TB1300
SAP Business One – Software Development Kit (SDK)

SAP Business One
2010 / Q2

- Version 92
- Material number: 50099230
Copyright 2010 SAP AG. All rights reserved.

Neither this training manual nor any part thereof may be copied or reproduced in any form or by any means, or translated into another language, without the prior consent of SAP AG. The information contained in this document is subject to change and supplement without prior notice.

All rights reserved.

Trademarks:

- Lotus ScreenCam ® is a registered trademark of Lotus Development Corporation.
- Vivo ® and VivoActive ® are registered trademarks of RealNetworks, Inc.
- ARIS Toolset ® is a registered Trademark of IDS Prof. Scheer GmbH, Saarbrücken
- Adobe ® and Acrobat ® are registered trademarks of Adobe Systems Inc.
- TouchSend Index ® is a registered trademark of TouchSend Corporation.
- Visio ® is a registered trademark of Visio Corporation.
- IBM ®, OS/2 ®, DB2/6000 ® and AIX ® are a registered trademark of IBM Corporation.
- Indeo ® is a registered trademark of Intel Corporation.
- Netscape Navigator ®, and Netscape Communicator ® are registered trademarks of Netscape Communications, Inc.
- OSF/Motif ® is a registered trademark of Open Software Foundation.
- ORACLE ® is a registered trademark of ORACLE Corporation, California, USA.
- INFORMIX ®-OnLine for SAP is a registered trademark of Informix Software Incorporated.
- UNIX ® and X/Open ® are registered trademarks of SCO Santa Cruz Operation.
- ADABAS ® is a registered trademark of Software AG
- The following are trademarks or registered trademarks of SAP AG; ABAP/4, InterSAP, RIVA, R/2, R/3, R/3 Retail, SAP (Word), SAPaccess, SAPfile, SAPfind, SAPmail, SAPoffice, SAPscript, SAPtime, SAPtronic, SAP-EDI, SAP EarlyWatch, SAP ArchiveLink, SAP Business Workflow, and ALE/WEB. The SAP logo and all other SAP products, services, logos, or brand names included herein are also trademarks or registered trademarks of SAP AG.
- Other products, services, logos, or brand names included herein are trademarks or registered trademarks of their respective owners.
The following are required prerequisites for attending this course:

- Knowledge of SAP Business One standard business processes. This prerequisite can be met by completing the courses in the “Product essentials Learning Map”
- Basic knowledge and experience with Microsoft .NET technology – ideally Visual Basic .NET since the exercises in the course will be performed with VB .NET and code examples are also provided in VB .NET only
- Basic knowledge of and experience with software development processes
- Students must bring their own laptop with SAP Business One software at release 8.8 installed. **No training system will be provided for participants.** Participants will be required to install a local demo database on their laptop during the class.

**Note:** this course assumes that participants have basic general accounting and IT skills.
Target Audience

The target audience for this course is an external SAP Business One consultant who will be developing additional functionality in or for SAP Business One.

Duration: 4 ½ days

User notes

- These training materials are not a teach-yourself program. They complement the explanations provided by your course instructor. Space is provided on each page for you to note down additional information.
- There may not be sufficient time during the course to complete all the exercises. The exercises provide additional examples that are covered during the course. You can also work through these examples in your own time to increase your understanding of the topics.
Contents:
- Course Goals
- Course Objectives
- Course Content
- Course Overview Diagram
- Main Business Example
This course will prepare you to:

- Know the basics of the SAP Business One SDK as well as details that are important for a general understanding of the SDK
- Create a partner package that contains enhancements to the SAP Business One Software
Course Objectives

After completing this course, you will be able to:

- Develop simple additional functions (add-ons) using the Data Interface (DI) Application Programming Interface (API) and develop simple enhancements using the User Interface API (UI API)
- Use the User-Defined Objects (UDO) feature
- Use SDK components in (customer) projects
- Modify business processes with the SDK
- Create and deliver an add-on installation package, including using the license mechanism
- Find and work with the SDK relevant information resources
- Know SAP solution certification requirements

To ensure that you retain the knowledge gained in this course and successfully complete the certification examination, we recommend that you consolidate the content in your own time after the course.

We also recommend joining the developer community on the SAP Community Network to seek and provide help in everyday SDK challenges.
Course Content

Preface

Unit 1  Course Overview
Unit 2  SDK Introduction
Unit 3  The Data Interface API
Unit 4  User-Defined Objects
Unit 5  The User Interface API
Unit 6  Add-On Packaging, Add-On Administration & Licensing

Appendices

- 1 Course Overview
- 2 SDK Introduction
- 3 The Data Interface API
- 4 User-Defined Objects (UDO)
- 5 The User Interface API
- 6 Packaging, Add-On Administration and Licensing

Appendices:
- Contain guidance how to implement the “Course Project”
- Include information about available tools
- Provide an overview on SDK installation matters and support processes
- Provide more details about some features that are only mentioned briefly in the User Interface API unit

The last one is supposed to refresh – or provide – details e.g. about the “Formatted Search” feature
Glossary

- **API** – Application Programming Interface
  Technology name for approaching application through an interface

- **COM** – Component Object Model
  Microsoft specific technology / Model for interfaces

- **SDK** – Software Development Kit
  A package that enables developers to implement own modules – here to build solutions that interface with SAP Business One (i.e. COM objects, services, and other tools)

- **Interface**
  An access point to exchange data with e.g. an application

- **Software Solution Partner (SSP)**
  Also known as ISV (Independent Software Vendor) implements solution(s) based on SAP Business One and SAP Business One SDK

- **Channel Partner (CP)**
  Sells and customizes SAP Business One. Often uses SAP Business One SDK for customer projects only.
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.
Introduction: Unit Objectives

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

- The SAP Business One Software Development Kit
- Data Interface API
- User Interface API
- SAP Business One integration for SAP NetWeaver
Course Overview Diagram

1 Course Overview
2 SDK Introduction
3 The Data Interface API
4 User-Defined Objects (UDO)
5 The User Interface API
6 Packaging, Add-On Administration and Licensing
SAP Business One is implemented as a two-layer architecture. The system is based on a Microsoft SQL Server database where data is stored centrally. The business logic is mostly processed on the client software (fat client).

In detail, the client software consists of a graphical user interface and the business object classes connecting to the database.

There are several built-in integration capabilities, interfaces and customization features (see “Adaptation”, MS Office Integration” + “Interfaces”):

* Besides all the adaptation capabilities accessible for customers, SAP Business One SDK enables partners to implement a solution extending SAP Business One using APIs and other features.
* DI Server e.g. enables partners to use SAP Business One data in a Browser without the need to install any SAP Business One component on the client or the application server of the web-based application.
* The UDO feature is a further step to ease creation of additional functionality inside SAP Business One.

Licenses are also managed centrally. Partners can use the mechanism for own purposes.
## Introducing SAP Business One SDK: Unit Overview Diagram

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topic 1: Introducing SAP Business One SDK</strong></td>
</tr>
<tr>
<td>Topic 2: Introducing Data Interface API</td>
</tr>
<tr>
<td>Topic 3: Introducing User Interface API</td>
</tr>
<tr>
<td>Topic 4: SAP Business One integration for SAP NetWeaver</td>
</tr>
<tr>
<td>Topic 5: Introduction to the Course Project</td>
</tr>
</tbody>
</table>
After completing this topic, you will be able to:
- Describe purpose and components of the SDK
- Explain SDK packages and licenses shortly
- Tell where to find further information or seek help
- Use test tools available on the SAP Community Network
The SAP Business One client software consists of a graphical user interface and the business object classes connecting to the database.

The source code of SAP Business One is not accessible by third parties. This guarantees a single version of SAP Business One with approved stability, functionality and upgrade functionality.

If you want to extend and change the functionality of SAP Business One, you can use the built-in tools for adjustments such as User defined fields and tables, formatted search, etc.). If your enhancements need more, You can use the SAP Business One Software Development Kit.

With this SDK, you can

- add industry-specific functions
- add other functions you deem necessary
- create interfaces to third-party tools.

SDK gives access to Business One internals via a set of programmatic interfaces based on COM: every development environment supporting COM can be used.

Many add-ons can be executing together with Business One changing its standard behavior: add-ons live in separate address spaces than Business One.

Nevertheless you should not underestimate the power of the customization tools!

Check-out the Appendix „More exercises and solutions“ for more information about „formatted search“ „queries“ and „alerts“.
The different application programming interfaces (APIs) included in the Software Development Kit use open Microsoft standards that allow access to a lot of business objects provided by SAP Business One.

API runtimes are installed with the SAP Business One client application – except DI Server which is part of the SAP Business One Server Tools installation.

You can access SAP Business One

- on business data level through the Data Interface API (DI API). Most SAP Business One business objects are exposed in this API. They can be accessed by external programs. If you prefer using Java, use “Java Connector” to access DI API.
- on business data level through DI Server (Data Interface). DI Server is a DCOM service that runs on the SAP Business One server and accepts XML data packed in SOAP (Simple Object Access Protocol) “envelopes”.
- on user interface level: The User Interface API (UI API) provides access to a running application where you can add or modify forms, and provide your own event handlers to actively influence the existing business logic.

In addition to that you can define your own business objects (User-Defined Objects (UDO)) that are joined to the SAP Business One business object collection.

The SDK ships with

- Sample Code
- Documentation
- Utilities
The SDK consists of 2 major packages:

1) The Runtimes (i.e. the APIs)
   - The runtimes (DI API + UI API) are installed together with the SAP Business One client application
   - DI API can be installed stand-alone (separate installer package available)
2) The documentation (samples, helpfiles, utilities) named „SDK Installation“

“SDK” Installation
- Development package for partners – includes SDK help, samples, tools

(SDK) Licenses
- SDK Implementation License
  - For customer specific implementation (usage of UI API only)
  - SDK Implementation license (99999 licenses) included in Professional User license
- SDK Development License / Add-On Solution License
  - Need development or solution license to use UI&DI API (we will talk about that later)
  - Partner has to order SDK Development License to start development
- AddOn Access User License
  - Allows to work via UI&DI API – cannot be used to do anything in the B1 application
- Indirect Access User License
  - Allows to work via DI only – cannot be used to do anything in the B1 application
  - New in version 8.8: Add-on connection requires user having an SAP License!

There’s only one set of APIs – no debug / release…

According to the remark above you could see the SAP Business One Software Development Kit to be available in three “versions”:

- The SDK Installation is a full version suitable for development of additional components by partners or customers. It contains documentation and examples.
- The SDK Implementation Version basically is just the general authorization to use UI API, if at least a Professional User license has been installed.
- The SDK DI API / Runtime Installation is required if customers want to run additional functions provided by a partner using DI API. It is installed with the client.

- In the past „Compatibility License for Add-Ons“ existed – to allow partners to work without using the SAP license mechanism for some time. While this is still possible – there’s no license for this purpose yet, but the users need any payable SAP license to connect to SAP Business One’s SDK starting with version 8.8.
You can get more information on the service marketplace via http://service.sap.com/smb.

Another valuable source of information about the SAP Business One SDK is currently the SAP Developer’s Network. You can access it under http://www.sdn.sap.com. There is a Discussion forum where hot topics regarding the SAP Business One SDK are discussed.

Please note:
- You can find additional information in Appendix 3 of this course material.
What you can find on SDN: People like you...

Free registration to Discussion Forums
What you can find on SDN: Technical information...

- **Developer Area** – includes:
  - Links to tools, articles, FAQ etc

**FAQ**

![SAP Business One screen shot](image-url)

**SAP Business One**

**Get Your Hands with Business One - Starter Community Post**

Would like to have involved with the followers of METRO or any about ABAP or Business One? Then this is your chance!

- User Forum: SAP Community Network
- SAP Business One: SAP Community Network

**SAP Business One (B1) is been as Strange**

No hooking up!

Thanks for your active collaboration during the SAP Business One B1 Hackathon! As you might have noticed, this project is still under development and needs more input from our community. We will release this project in 2009, and we will announce the SAP Business One B1 Hackathon on social media channels. In the meantime, you can still contribute to this project by sharing your ideas and suggestions. We are looking forward to your feedback and continued support.

**Business One Knowledge Center**

- [Database Setup](link)
- [SAP Business One and EDI Server](link)

**SAP Organizational Change Management Toolkit**

Tell us what you think of updates for educational purposes. Participate in discussions and share your feedback.

**Articles**

- [Import(exports) data as xml or as new xml](link)
- [Import(exports) data as xml or as new xml](link)
What you can find on SDN: Development tools

- B1 DB Browser
- B1 Test Composer
- B1 Form Checker
- B1 Code Generator
Introducing SAP Business One SDK: Topic Summary

You should now be able to:

- Describe purpose and components of the SDK
- Explain SDK packages and licenses shortly
- Tell where to find further information or seek help
- Use test tools available on the SAP Community Network
<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: Introducing SAP Business One SDK</td>
</tr>
<tr>
<td><strong>Topic 2: Introducing Data Interface API</strong></td>
</tr>
<tr>
<td>Topic 3: Introducing User Interface API</td>
</tr>
<tr>
<td>Topic 4: SAP Business One integration for SAP NetWeaver</td>
</tr>
<tr>
<td>Topic 5: Introduction to the Course Project</td>
</tr>
</tbody>
</table>
Introducing Data Interface API (DI API): Topic Objectives

After completing this topic, you will be able to:

- Explain what DI API is high-level
- Tell how DI API is used
- Know about DI Server
- Explain the User-Defined Objects concept high-level
SAP Business One SDK – Data Interface API / DIServer

Provides objects and methods (add, update etc.) to work on data level – installing the SAP Business One client application is not required

Provides access to business objects (e.g. master data and transactional data) and cross functionalities (services)

Performs the same checks as the SAP Business One client application

Links existing third-party solutions “as-is”

Use COM capable development tools (e.g. Microsoft Visual Studio)

- DI API is meant to be used by partners only!
- To use the DI API, you must either use a development environment and programming language that support Microsoft COM (component object model) technology and is released by SAP.
- Alternatively – just using JavaConnector (JCo) – you can use a Java development environment like Eclipse
- The following development environments are released by SAP:
  - Microsoft Visual Basic .NET (or Microsoft Visual Studio 6.0)
  - Microsoft Visual C++ .NET (or Microsoft Visual Studio 6.0 for C++)
  - Microsoft C# .NET
- Other development environments supporting COM technology might work but SAP does not provide support for them. See SAP Note 615987 for a complete list of development environments released by SAP.
- Note: SDK does not contain a development environment or source code editors. This is to give you the flexibility to choose the environment you prefer.
- Supported platforms: https://websmp209.sap-ag.de/~sapidb/011000358700001241092005/
- Note: SAP highly recommends that you install the latest Support Packages for the supported platforms. See SAP Note 628155 for a complete overview of supported platforms.
- The UDO feature is supported by DI API as far as meta data are concerned.
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 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 by default 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…
The DI Server is designed to run on a server machine and supplies a light-weight SOAP-based access layer
- 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
- Potential Solution to 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.

Limitations:
Meta data operations not supported
Different support for transaction handling than plain DI API

- 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!
The SAP Business One architecture now allows to add own Business Objects for your own purposes to the applications object collection.

As a consequence you can register your objects to participate in some most important functionalities („Services“) offered by the SAP Business One application as stated above. Thus you don’t have to reimplement the functionality in your application needed for supplying the Search function or adding data to the database (with some preconditions).

A lot more details will be covered in the unit dedicated to the User Defined Object feature.

We would like to emphasize that this already brings a lot of benefit to you – even without using the Implementation DLL feature!
Introducing Data Interface API (DI API): Topic Summary

You should now be able to:

- Explain what DI API is high-level
- Tell how DI API is used
- Know about DI Server
- Explain the User-Defined Objects concept high-level
Introduction

Topic 1: Introducing SAP Business One SDK
Topic 2: Introducing Data Interface API
**Topic 3: Introducing User Interface API**
Topic 4: SAP Business One integration for SAP NetWeaver
Topic 5: Introduction to the Course Project
Introducing User Interface API (UI API): Topic Objectives

After completing this topic, you will be able to:

- Explain what UI API is high-level
- Tell how UI API is used
To use UI API, you must either use a development environment and programming language that support Microsoft COM (component object model) technology and is released by SAP.

- UI API has no Java libraries
- Often you also use DI API and UI API in the same Add-On / 3rd party application
- The UDO feature is supported by UI API
- The following development environments are released by SAP:
  - Microsoft Visual Studio 6.0 for Visual Basic (VB) and Microsoft Visual Basic .NET
  - Microsoft Visual Studio 6.0 for C or C++ and Microsoft Visual C++ .NET
  - Microsoft C# .NET
- Other development environments supporting COM technology might work but SAP does not provide support for them. See SAP Note 615987 for a complete list of development environments released by SAP.
- Note: SDK does not contain a development environment or source code editors. This is to give you the flexibility to choose the environment you prefer.
**User Interface API – Use Cases**

**User Interface API is usually used to:**

- Reach a “seamless” integration of additional functionality with SAP Business One (usually requested by customers)
  - …including hooking on SAP Business One standard processes
  - …including adding own GUI elements into SAP Business One standard forms
  - …including adding own forms and plugging the corresponding data behind

- Manipulate SAP Business One standard functionality (when standard options do not apply to the customer’s processes (or the branch the customer works in))
  - …including hiding SAP Business One GUI elements
  - …including blocking SAP Business One events
You should now be able to:
- Explain what UI API is high-level
- Tell how UI API is used
## Introducing SAP Business One SDK: Unit Overview Diagram

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: Introducing SAP Business One SDK</td>
</tr>
<tr>
<td>Topic 2: Introducing Data Interface API</td>
</tr>
<tr>
<td>Topic 3: Introducing User Interface API</td>
</tr>
<tr>
<td><strong>Topic 4: SAP Business One integration for SAP NetWeaver</strong></td>
</tr>
<tr>
<td>Topic 5: Introduction to the Course Project</td>
</tr>
</tbody>
</table>
After completing this topic, you will be able to:
- Explain the purpose of B1iSN
- Tell what connectivity types it supports
- Describe how to set it up
- …and how to build your own scenarios
- Talk about getting information about errors
B1iSN – Solution for seamless integration between SAP Business One and:
SAP R/3 and/or SAP ERP (ECC 6.0) and/or SAP Business One and/or ...

Key benefits of B1iSN:
- Rapidly connects subsidiaries running SAP Business One to headquarters (and other subsidiaries)
- Standardizes and unifies business processes across the business ecosystem
B1iSN Connectivity Types

B1iSN 2007 provides many of out-of-the-box connectivity types:
- SAP Business One (DI, SQL)
- SAP ERP (RFC / ALE/XI-PI)
- SAP NetWeaver BW (RFC / SOAP)
- Database systems (SQL)
- HTTP any
- File (CSV, Offset)
- Web Services (In/Out, Sync/Async)

For each connectivity type multiple systems can be set up (many to many)

Connectivity types are represented in B1iSN via System Types

- BW = Business Warehouse
- Mayn connectivity types available
B1iSN Simple sample for deployment

- Notification Mechanism: Creating Events for change in SAP B1 (table SBO-COMMON.SEVT)
- EventSender: Sending Events to SAP B1iSN
- DI Proxy: The Data Channel between SAP B1 and SAP B1iSN
- B1iSN Server: The Integration Server

- Typical setup:
- B1iSN installed on a different machine
High level – only the part in pink color is missing
Graphical overview of BizStep

- A green box says that a specific element is relevant and correct.
- A yellow box shows elements that are "not correct."
- A red box is displayed in case the item relevant for the scenario, mandatory but not yet specified or missing in the repository.
Message Log – shows what is going on:

What went well

Whether problems arised

Click on the hyperlink to see the „message“ at this stage
You should now be able to:

- Explain the purpose of B1iSN
- Tell what connectivity types it supports
- Describe how to set it up
- …and how to build your own scenarios
- Talk about getting information about errors
<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: Introducing SAP Business One SDK</td>
</tr>
<tr>
<td>Topic 2: Introducing Data Interface API</td>
</tr>
<tr>
<td>Topic 3: Introducing User Interface API</td>
</tr>
<tr>
<td>Topic 4: SAP Business One integration for SAP NetWeaver</td>
</tr>
<tr>
<td><strong>Topic 5: Introduction to the Course Project</strong></td>
</tr>
</tbody>
</table>
After completing this topic, you will be able to:

- Explain the course project
**Course Project - Description: Video Library Module**

**Business case:**
Add a small module that will enhance SAP Business One application functionality to manage a video library.

**Features:**
- A movie is considered as an Item with some specific properties
- Add a new DVD
- DVD Availability Check
- Rent DVD
- Return DVD

Within the Course Project we will create an Add-On using UI API and DI API. The following slides show how these forms could look like…
A Movie is considered as an Item Master Data with some specific properties. Adding a DVD will insert an item master data as well. DVD Code = Item Code…

- After you have passed the UI API section of this course you may have a couple of ideas how to improve this form. You are encouraged to try to apply them!
Check the DVD availability before rental.

After you have passed the UI API section of this course you may have a couple of ideas how to improve this form. You are encouraged to try to apply them!
Course Project – Rent/Return DVD

Rent DVD: The stock no. of DVD will decrease.

Return DVD: The stock no. of DVD will increase.

- After you have passed the UI API section of this course you may have a couple of ideas how to improve this form. You are encouraged to try to apply them!
A Movie is considered as an Item Master Data. The form displays a list of the movie Rental.

- After you have passed the UI API section of this course, you may have a couple of ideas how to improve this form. You are encouraged to try to apply them!
You should now be able to:
- Explain the course project
Introduction: Unit Summary

You should now be able to describe and explain:
- The SAP Business One Software Development Kit
- The components of the SAP Business One Software Development Kit
- How to use SDK in general
- Data Interface API
- User Interface API
- How SAP Add-Ons and applications use SDK
- SAP Business One Integration for SAP NetWeaver
1-45

Exercises

Unit: Introduction

Topic: Specification for the Course Project

The following pages contain details about the functionality you should implement in the Course Project:

- Add Meta Data (additional user defined fields to the SAP Business One company database) and Master Data
- Create new Menus and Forms for your add-on
- Create an Add-On (which will be running “inside” SAP Business One) which will allow you to control your DVD Store
- Create an installer for this Add-On

You want to develop additional functionality for SAP Business One. Add a small module that will enhance SAP Business One application functionality to manage a DVD Store.

Pre-requisite: Use a non-continuous stock system

Pre-requisite: For usability we will only stock 1 DVD per title

1-1 Adding MetaData

1-1-1 Add the following UserFields to the Item Master Data Table (OITM).

- Aisle Number – Indicates in which aisle the movie is stored.
  - Field Name: AISLE
  - Field Description: Aisle Number
  - Field Type: db_Numeric
  - Field EditSize: 2

- Rented – Indicates weather the movie is rented or not. Holds 2 “valid values”: Y/N.
  - Field Name: RENTED
  - Field Description: Rented/Available
  - Field Type: db_Alpha
  - Field EditSize: 1
CardCode – In case the movie is “Rented”. This field will hold the CardCode of the customer who rented it otherwise it will be empty.

Field Name: CARDCODE

Field Description: Card Code

Field Type: db_Alpha

Field EditSize: 20

1-2 Define settings for master data

1-2-1 Open the Item Groups table in SAP Business One. You can find this under Administration -> Setup -> Stock Management -> Item Groups. Define the DVD categories you wish to use e.g. Horror, Comedy, Drama, Animation, Romance, Science Fiction etc.

1-2-2 Create three new price lists and assign the associated fixed prices for the DVDs depending on the price list set. The window can be found under Stock Management -> Price Lists -> Price Lists. Price list are called

- Weekly rental
- 1 night rental
- 3 night rental

1-3 Creating DVD Store Add-On

1-3-1 Create a new project and add the UI API and the DI API to the project references.

1-3-2 Connect with your Add-On to the UI and to the DI API using the multiple add-on feature.

1-3-3 Add the following Menu Items to SAP Business One Menu collection:

Sub Menu: DVD Store

Menu Items: Members Master Data

DVD Master Data

DVD Availability Check

Rent DVD

Return DVD
1-3-4 **Members Master Data Form**

1-3-4-1 Each new DVD store member is represented by an entry in OCRD (Business Partner Master Data)

-3-4-2 **Functionality:** When clicking on the Members Master Data menu the Business Partner Master Data form opens – this is the standard SAP Business One form.

1-3-4-3 Add some new DVD store members.

1-3-4-4 **Additional Functionality:** Add a new tab to the Business Partner Master Data form called ‘Rental History’. Create a matrix on this new tab – this will record the DVDs previously rented by this customer. Display Invoice No, Date and DVD name for all previous rentals by this customer. The screen should look as follows:

![Rental History Matrix](image)

Hint: You will need to query the Invoice tables

1-3-4-5 **Functionality:** Allow the user to sort by DVD Name

1-3-5 **Add DVD Form**

1-3-5-2 Each new DVD is represented by an entry in OITM (Item Master Data). Each time we add a new DVD we will also add a Goods Receipt to add the new DVD to Stock. Therefore it is more efficient to create our own user form.
1-3-5-2 **Functionality:** Draw the “Add New DVD” form (do it through code or use the screen painter).

![Add New DVD form](image)

1-3-5-3 **Functionality:** When the user clicks on Add a new Item will be created in OITM via the DI.

- ItemCode = DVD Code
- ItemName = DVD Name
- U_AISLE = DVD Aisle
- ItemGroup = DVD Category
- Price List = DVD Price List (Weekly, 1 night rental, 3 night rental)

1-3-5-4 Also via the DI create a Goods Receipt (oInventoryGenEntry) to add the new DVDs to stock

- Price List = DVD Price List
- ItemCode = DVDCode
- Quantity = No. of DVDs. For ease of usability we will only add 1 DVD per title.
1-3-6 DVD Availability Check Form

1-3-6-1 This was already partly created in the UI Exercises so you can reuse some of the code. This form will allow you to search for a particular DVD and check its availability.

1-3-6-2 **Functionality**: Draw the “DVD Availability Check” form (do it through code or use the screen painter).

1-3-6-3 Note all are Edit Text except:

   1-3-6-3-1 DVD Name is linked to a Choose from List (OITM)

   1-3-6-3-2 DVD Category is a combo box linked to Item Groups already defined

1-3-6-4 **Functionality**: Data bind each field to its associated column in the database (DBDataSource = OITM)

1-3-6-5 **Functionality**: When clicking on the Choose button or selecting tab in DVDName a Choose from List window will open with all available DVDs from the OITM table in the database. Select the DVD you want to view and the remaining fields will be auto filled based on that selection.

1-3-6-6 **Functionality**: When clicking Rent DVD the Rent DVD window opens.
1-3-7 Rent DVD Form

1-3-7-1 This form will enable the use to rent a particular DVD to a member. Rental is different from normal sales process. A virtual warehouse will be created for those rented DVD. When a DVD is rented, this DVD will be transferred from main warehouse to the virtual rented warehouse with stock transfer, and a manual JE similar to A/R invoice or a service invoice will be added via the DI. Once the DI manual AR JE or Service Invoice has been added we will use the UI to open the Incoming Payment screen, select the Customer and select the Payment Means so the Payment can be completed by the user.

1-3-7-2 **Functionality:** Draw the “Rent DVD” form (do it through code or use the screen painter).

![Rent DVD Form](image)

1-3-7-3 **Functionality:** Customer (OCRD) and DVD (OITM) are both combo boxes and should be automatically filled when the form is open. (Hint: use RecordSet)

1-3-7-4 **Functionality:** User selects Customer and DVD Name. When the user clicks on Rent
- Stock level: A stock transfer of this DVD with Quantity of 1 will be added from main warehouse to rented warehouse via the DI API.
- Financial level: A manual AR JE or AR service invoice will be added via the DI API.

Hint: Ensure error checking is done e.g. DVD is available to rent, Combo boxes have been selected etc.

1-3-7-5 **Functionality:** Store the CardCode and DocNum of the newly added Invoice in two variables (Hint: use GetNewObjectCode method to retrieve DocNum)
1-3-7-6 **Functionality:** Via UI:

1-3-7-6-1 Open the Incoming Payment window by simulating a click on the Incoming Payment menu under Banking.

1-3-7-6-2 Fill the Code field on the Incoming Payment screen with the Customer value you saved in the variable when Invoice was added via the DI. The invoices to be paid will appear in the Matrix.

1-3-7-6-3 Loop through the rows (Invoices) in the matrix until value in DocNum cell equals the value in the DocNum variable saved after adding the Manual AR JE or AR Service Invoice via DI.

1-3-7-6-4 When matching DocNum is found select the Selected checkbox.

1-3-7-6-5 Simulate a click on the Payment Means icon.

1-3-7-6-6 The user will now be able to process the Payment for the DVD rental.

1-3-7-7 **Functionality:** Via DI Update OITM as DVD is now rented

    U_Rented = Y

    U_CardCode = Customer

1-3-8 **Return DVD Form**

1-3-8-1 This form will enable the use to return a DVD to the store. It will create a Goods Receipt via the DI to return DVD to stock also.

1-3-8-2 **Functionality:** Draw the “Return DVD” form (do it through code or use the screen painter).
1-3-8-3 **Functionality**: User selects DVD Name. When the user clicks on Return a Stock Transfer for the DVD (OITM) being transferred from Rented warehouse to Main warehouse via the DI API. This DVD will then be returned back into stock.

Hint: you will need to check what customer is currently renting the DVD first.

1-3-8-4 **Functionality**: Via DI Update OITM as DVD is now back in stock.

\[ U_{\text{Rented}} = N \]

\[ U_{\text{CardCode}} = "" \]

1-4 **Add-On Administration**

1-4-1 Create and installation package for your Add-on. (Use the SAP Business One Development Environment (B1DE) toolset for this purpose)

1-4-2 Create an ard file. (Use B1DE – and try “manually”)

1-4-3 Register your Add-On.

1-5 **License**

1-5-1 Request a BASIS license for you add-on and include the Add-On Identifier in your code.
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.
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
- 1 Course Overview
- 2 SDK Introduction
- 3 The Data Interface API (short look on JCo + DI Server)
- 4 User-Defined Objects (UDO)
- 5 The User Interface API
- 6 Packaging, Add-On Administration and Licensing
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.
## DI API Introduction: 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>
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
DI API Introduction – Business Objects

- Represent records in the SAP Business One company database – often distributed across multiple tables
- Represent the functionality of the SAP Business One application
- Provide access to data and enable to modify the data (GetByKey, Read, Add, Update, Remove,...)
- Rules and checks (including authorizations) apply – regardless whether data are handled through the application or DI API / DI Server Business Objects

<table>
<thead>
<tr>
<th>Example: Object model of the BusinessPartners Master Data Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusinessPartners</td>
</tr>
<tr>
<td>UserFields</td>
</tr>
<tr>
<td>ContactEmployees</td>
</tr>
<tr>
<td>Browser (DataBrowser)</td>
</tr>
<tr>
<td>Addresses (BPAddresses)</td>
</tr>
<tr>
<td>AccountReceivablePayables (BP AccountReceivablePayables)</td>
</tr>
<tr>
<td>BPPaymentMethods</td>
</tr>
<tr>
<td>BPWithholdingTax</td>
</tr>
<tr>
<td>BPPaymentDates</td>
</tr>
<tr>
<td>BPBankAccounts</td>
</tr>
</tbody>
</table>

- A lot of business objects contain collections of additional objects like UserFields and more
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).
The Infrastructure objects do not represent SAP Business One data.

Company object
- Represents an SAP Business One Company database on Microsoft SQL Server
  - 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 through records of a certain object type (e.g., business partners) in conjunction with Recordset
- SBObob: Exposing extended / supplemental functionality

Meta Data Objects
- UserTablesMD: Create user tables
- UserKeysMD: Define an index for a user table
- UserFieldsMD: Create user fields (add to 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.
This slide focuses on the database connection part when connecting to a company database with DI API!

In addition you always have to supply the SAP Business One user code + password into the properties UserName and Password!

In case the connect method fails:
- Check all the properties.
- Use the “GetLastError” method to retrieve the error code and string. You can find details about the error code in the SDK documentation.
- Reassign OBSCommon user (note# 694413).

Please note:
Starting with B1 8.8 DB credentials are kept centrally – and are administrated via License Service.
For backward compatibility reasons DI API still supports supplying credentials for connection.

Relevant Properties:
- UseTrusted
- DBUserName
- DBPassword

UseTrusted = False

Connect with DB user.
Change values for the following properties:
DBUserName, DBPassword.

UseTrusted = False
DBUserName = "<Valid DB user (e.g. sa)>"
DBPassword = "<The password>"

UseTrusted = True

Connect with Windows user account (MSSQL only!).
Change value for the following property:
UseTrusted = True
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.
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)
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…
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

<table>
<thead>
<tr>
<th>Topic 1: DI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topic 2: Business Objects</strong></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>
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.
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?
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
Working with XML – Relevant methods and properties

Company object
- `oCompany.GetBusinessObjectFromXML (FilePath_OR_XMLString, Index)`
- `oCompany.GetXMLElementCount (FilePath_OR_XMLString)`
- `oCompany.GetObjectType (FilePath_OR_XMLString, Index)`
- `oCompany.GetBusinessObjectXmlSchema (ObjectType)`

XML export type – determines whether or not e.g. to export read-only data
- `oCompany.XmlExportType = e.g. xet_ExportImportMode`
- Please note: ONLY with xet_ExportImportMode data are exported in a manner that allows to import them again.

Working with XML as an XML string (not as an XML file)
- `oCompany.XMLAsString = True`

Business objects
- `oBusinessObject.SaveXML (FilePath_OR_XMLString)`
- `oBusinessObject.Browser.ReadXML (FilePath_OR_XMLString)`
  Use ReadXML to update an existing object

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.)
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.
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.

```vbnet
Dim sFileName As String = "c:\temp\BPs.xml"
Dim lEcount, ii As Long

' Get the number of Business object in the file ...
lEcount = oCompany.GetXMLElementCount(sFileName)

' Loop through the objects; when finding the first BusinessPartner 'object: load it, add it to the DB.
For ii = 0 To lEcount–1
    If oCompany.GetXMLObjectType(sFileName, ii) = _
        SAPbobsCOM.BoObjectTypes.oBusinessPartners Then

    "Read" the Business object data into the object…
    ' Please note:
    ' If the format is not OK you might run into an exception!
    oBP = oCompany.GetBusinessObjectFromXML(sFileName, ii)

    iRetVal = oBP.Add()
    ' …handle error…
    End If
    Next ii
```
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.

---

**Transaction Handling: Overview**

The Data Interface API supports two different types of transactions:

**Single Transaction (default)**
- Each data operation performed on a business object starts a transaction
- Depending on the result (success or failure), the system automatically issues a commit or a rollback

**Global Transaction**
- Allows perform several data operations and then a full commit or rollback based on specific criteria.
- If any of the data operations fails the global transaction will be rolled-back entirely
- Start and end of global transaction can be managed by using the Company object:
  - Company.StartTransaction()
  - Boolean Company.InTransaction
  - Company.EndTransaction([wf_RollBack / wf_Commit])

---
Transaction Handling: Flow Chart of Global Transactions

- Start Transaction

  Run a list of operations on the DB

  Did ALL operations succeed?

  Yes? ...then you can still choose:

  - Rollback?
  - Commit?

  End Transaction

- NO! At least one operation failed

  Automatic Rollback already happened, Transaction has been terminated

  Stop processing and handle error!

  Please note: “InTransaction” property of Company object holds info whether or global transaction is still active.

- 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 next Unit)
- 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-ef5268b00aaaf
- 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.
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
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…
## Non-Business Objects: 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>
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
RecordSet Object

Purpose:
- Temporary solution to work with SAP Business One 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

- Purpose:
  - Temporary solution for partners that need to work with objects that aren’t exposed (yet) with the DI API.
  - Very risky – mostly no validations, BE CAREFUL!!!
  - We recommend strongly to use the RecordSet object only for data reading purposes!

- 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.
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.

<table>
<thead>
<tr>
<th>DataBrowser Object – Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object:</strong></td>
</tr>
<tr>
<td>&lt;Business Object&gt;</td>
</tr>
<tr>
<td><strong>Methods:</strong></td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td><strong>Properties:</strong></td>
</tr>
<tr>
<td>Browser =&gt; RecordSet</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

You can call the DataBrowser object using the *Browser* property for all business objects.

Enables data navigation through all objects of a certain object type.

Easy to use – direct access to business object properties.

You cannot create a new DataBrowser object, it is invoked as a *Browser* property of a business object.

Example: Walk through all business partners.
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...
# Meta Data Objects: 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>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>
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…
Add User-Defined Tables via „User-Defined Tables-Setup“ form

Use context menu to remove a User-Defined Table (please note that there are some prerequisites!)

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.
Meta Data Objects: UserTablesMD

Use the UserTablesMD object to create a user defined table via DI API

```vbscript
'Object variable
Dim oUTables As SAPbobsCOM.UserTablesMD
'Create instance of UserTablesMD object
oUTables = oCompany.GetBusinessObject(oUserTables)

'Check whether table already exists
If oUTables.GetByKey("TB1_Table") Then
    oUTables = Nothing
    Exit Sub
Else
    oUTables.TableName = "TB1_Table"
    oUTables.TableDescription = "TB1300 test table"
    lRet = oUTables.Add()
End If

'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…
oUTables = Nothing
```

- 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…
Add User-Defined Fields to Tables

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Description</th>
<th>Type</th>
<th>GFI</th>
<th>Mand.</th>
<th>Index</th>
<th>Linked To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Data</td>
<td>Master Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>User Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Entities</td>
<td>Business Partner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sales Rep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key Contact</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact Person</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cargo Customer Declaration Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Employees</td>
<td>Employees</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GL Accounts</td>
<td>GL Accounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Groups</td>
<td>Item Groups</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Items</td>
<td>Items</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price Lists</td>
<td>Price Lists</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Codes</td>
<td>Project Codes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sales Employee</td>
<td>Sales Employee</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Users</td>
<td>Users</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warehouses</td>
<td>Warehouses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marketing Docs</td>
<td>Marketing Docs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payments</td>
<td>Payments</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
Use the UserFieldsMD object to create user defined fields

```
'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
lRet = 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
```

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.
### Defining a User-Defined Field

#### Title and Description

#### Type and Structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric</td>
<td>Regular, Address, Telephone, Text</td>
</tr>
<tr>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Date/Time</td>
<td>Date, Time</td>
</tr>
<tr>
<td>Units and Sums</td>
<td>Price, Sum, Unit, Quantity</td>
</tr>
<tr>
<td>General</td>
<td>Link, Picture</td>
</tr>
</tbody>
</table>

#### Additional Attributes

- Valid values (optional)
- Default value
- Mandatory (requires Default value)

- 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.
**Linking User-Defined Fields to User-Defined Tables**

Please remember:
When you create a User-Defined Table, two fields in the database are created by default: Code and Name

Please note:
- The data in the field will be taken from the field Code in the linked User-Defined Table.
- …therefore the field has to be alphanumerical and 8 characters long.
- You cannot link a User-Defined Field to other tables than User-Defined Tables (e.g. you can't link a User-Defined Field to the Business Partners table OCRD!)

- 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!
**User-Defined Fields in SAP Business One GUI**

- **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.ColumnAlias = "FieldName1"

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

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

' Add the key
lRet = oUKeys.Add()

' IMPORTANT: Only one handle to a user table or field or key object
' should be alive at the same time
In the toolbar menu go to “Tools” → “User-Defined Windows”

When linked to a User-Defined Field → Choose “Define new”

- 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…
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
You are now ready for:
- a MetaData objects exercise…
### 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>5</td>
<td>DI API Services</td>
</tr>
<tr>
<td>6</td>
<td>Java Connector (optional)</td>
</tr>
<tr>
<td>7</td>
<td>DI Server (optional)</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will be able to:

- Explain how to use DI API Services
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.
You should now be able to:
- Explain how to use DI API Services
### Java Connector (optional): 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>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><strong>Topic 6: Java Connector (optional)</strong></td>
</tr>
<tr>
<td>Topic 7: DI Server (optional)</td>
</tr>
</tbody>
</table>
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 APIInterface (SAPbobsCOM.dll)

Implementation (OBServerDLL.dll)

…can deal with COM (For DI API only!)

Server

SBO-Common

Observer.dll

Company

1

2

3
Java Connector (optional) – Details

Class / package hierarchy

- `java.lang.Object`
  - `com.sap.smb.sbo.util.ConvertUtil`
  - `com.sap.smb.sbo.api.SBOCOMUtil`
  - `com.sap.smb.sbo.api.SBOErrorMessage`
- `java.lang.Throwable (implements java.io.Serializable)`
  - `java.lang.Exception`
    - `com.sap.smb.sbo.util.NestingException`
    - `com.sap.smb.sbo.api.SBOCOMException`
  - `com.sap.smb.sbo.wrapper.util.WrapperUtil`

General remarks:

- All Interfaces are contained in the package `com.sap.smb.sbo.api`
- Check the Java Connector help file for more details.
- Important difference to DI API:
  - Objects to add new records are created using “new<Object name>” of the `SBOCOMUtil` class instead of using `ICompany` object’s “getBusinessObject”!
  - E.g. `newBusinessPartners` must be used when you want to add a business partner!

- There is an extra JCo help file. Below this, the help file for the data interface API holds as well.
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 = SBCOMUtil.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.getLastError();
    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 O/ITM";
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();
    }
}
Troubleshooting

- Test the issue in DI API first to check if it is the issue in DI

- SAP Notes
  - 1313297 : How to use SAP Business One Java Connector (JCO)
  - 1157304 : JCO_Failed connection to SBO produces memory leak
  - 1034147 : JCO_JVM shuts down with large payload

- 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)
## Java Connector (optional): 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>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>
At the conclusion of this topic, you will be able to:

- Use DI Server in principle
DI Server (optional) – Introduction

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.
**DI Server (optional) – Sample: Login**

Dim pCmp As New SAPbans10.CCompany
Dim ret As Long

pCmp.Server = "ASAFY"
pCmp.CompanyID = "SBDemo_US"
pCmp.UserName = "manager"
pCmp.Password = "manager"
pCmp.language = "en_US"
ret = pCmp.Connect()
**DI Server (optional) – Sample: Add Object**

```java
Dim ret As Long
Dim pBP As BusinessPartner
Set pBP = pExp.GetBusinessObject(oBusinessPartner)

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

ret = pBP.Add()
```

```xml
  <SOAP-ENV:Body>
    <S8ODL_Server:AddObject xmlns:S8ODL_Server="http://tumpuri.org/message/">
      <ObjectData><BO><AdminInfo><Object>2</Object></AdminInfo><BusinessPartners><row>
        <CardCode>Asaf</CardCode><CardName>Asaf Yankoni</CardName><CardType>C</CardType></row></BusinessPartners></BO></ObjectData>
    </S8ODL_Server:AddObject>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
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>“Function call efficiency”</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><strong>But please note:</strong> 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 “unlimited” number of connections (configurable) per database. Session pooling mechanism.</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>
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…
You should now be able to:
- Understand what the Data Interface API is
- Understand how the DI API exchanges data with SAP Business One
Unit: Data Interface API

Topic: Establish a Connection to SAP Business One

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

- Connect to an SAP Business One database

You want to develop additional functionality for SAP Business One.

In a first step, you want to create a simple program to connect to an existing SAP Business One database.

1-1 Log on to SAP Business One.

1-1-1 Note the name of one database you want to log on to.

1-1-1 Note one user in that database and the user's password.

1-2 Create a new Visual Studio project.

1-2-1 Within this project, create a form with two buttons on it. One of the buttons should be used to connect to the SAP Business One database, the other to disconnect from it.

1-2-2 Add a reference to the SAP Business One DI API COM library…

1-3 Code the connection to the SAP Business One database.

1-3-1 Define a variable for the Company object – ensure it is defined as a member of the add-on application class or globally.

1-3-2 Create a new Company object.

1-3-3 Set the properties needed to connect to the SAP Business One database.

1-3-4 Call connect on the Company object
1-4  Implement error handling and success handling.

1-4-1  If the connection succeeds, display a message box displaying a corresponding message.

1-4-2  If the connection failed, display the error message provided by the Company object.

1-5  Code the disconnection from the SAP Business One database.
Exercises

Unit: Data Interface API
Topic: Documents Object

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

- Work with Documents objects

Create a Sales Order in Business One. Via the DI create an Invoice based on this Sales Order and later create an Incoming Payment for that Invoice

2-1 On your Visual Studio project create a new button called “Invoice and Payment”

2-2 In Business One create an Order for a particular customer and a particular item.

2-2-1 First you must create a new Document object instance for the Invoice. Then you set the properties of the Documents object and the Documents_Lines ensuring the BaseEntry, BaseLine and BaseType are set.

2-2-2 Add the whole document. In the case of success, you should bring up a message box telling the user the number of the newly added Sales Invoice using the method GetNewObjectCode. In case of any error, you should display a message box with an error message.

The Documents object must be created with the GetBusinessObject method of the company object you are connected to. Look in the online help of the GetBusinessObject method for the correct object type. Which one must be used?

To access the Documents_Lines object, look at the properties of the Documents object.
To create a document based on a document you need to utilize the properties BaseEntry (DocEntry of Base document), BaseType (in this case Sales Order), BaseLine (line you wish to copy to target document).

2-2-3 Finally you should release the document object variables.

2-3 Create the Incoming Payment for this Invoice

2-3-1 Create a new Payments object instance for the Incoming Payment. Then you set the properties for the CardCode, Invoice DocEntry, and we will pay via cash so we will use the properties CashAccount and CashSum.

2-3-2 Add the whole document. In the case of success, you should bring up a message box telling the user the number of the newly added Payment using the method GetNewObjectCode. In case of any error, you should display a message box with an error message.

2-3-3 Finally you should release the document object variables.
Exercises

Unit: Data Interface API

Topic: XML

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

- Work with XML

Create data as XML and checkout how to use this process to add new data to the SAP Business One database.

3-1 On your Visual Studio project create a new button called “Working with XML”

3-2 Save the Invoice created in the Documents exercise as XML.

3-2-1 Try all settings for XmlExportType property on the Company object and find the differences.

Have a look at the DI-API Help file

3-2-2 Save the Invoice document created in the previous exercise in Xml format

Use the GetAsXml or SaveXml methods of the Documents object (verify all business objects have the same methods)

3-2-3 Test also the method GetBusinessObjectXmlSchema of the Company object. What kind of information does it save?
3-3 Modify the XML data obtained before and add it to the SAP Business One database.

Use the method GetBusinessObjectFromXML of the Company object

3-3-1 Try all files generated above and check the errors (exceptions) for details.
Exercises

Unit: Data Interface API
Topic: Transactions

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

- Work with transactions

Create an Order via DI API and later create an Invoice that is based in that Order, Documents exercise done before.

This time open a transaction before and close it afterwards.

4-1 Log on to a SAP Business One Company as shown in the first exercise.

4-2 Open a transaction (StartTransaction of the Company Object).

4-3 Perform the same actions as you did in the Documents exercise.

4-4 Close the transaction (EndTransaction of the Company Object).

4-5 Play e.g with the TaxCode (or VatGroup – depending on the localization!) property to see if and how the transaction fails. Also use wrong data (e.g. non-existing CardCode etc.) to see the reaction (as discussed in the presentation).
Exercises

Unit: Data Interface API
Topic: Using General Objects

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

- Use the data browser object to browse through a set of data
- Use the record set object

Create an application to navigate through all customers.
You will use the Browser property of the BusinessPartners object.
Add the navigation buttons to your form and provide the coding so that the user can browse through the customers.

5-1 On your Visual Studio project create a new button called “General Objects”

5-2 Create a new form in your Visual Studio application containing a text box where you will show the Business Partners Card Code and four buttons: first, previous, next and last.

5-3 Create a Recordset object and set the Browser property of the BusinessPartners object to this Recordset.

There is a code sample in the DI-API Help documentation.
5-3-1 Add the code to all four of the buttons so that the user can navigate backwards and forward through the customers. Be sure that your application only includes customers, not Leads or Vendors (Suppliers).

Use the DoQuery method of the RecordSet object with the appropriate SQL query.

5-4 Test your changes. Be sure to include the following scenarios:

5-4-1 Click the “First Record” button ( ), then click it again. Try the same thing with the “Last Record” button ( ).

5-4-2 Click the “First Record” button ( ), then click the “Previous Record” button ( ).

5-4-3 Click the “Last Record” button ( ), then click the “Next Record” button ( ).

5-4-4 If any of these scenarios raises an error, add code that will fix the error. Then test the application again.
At the conclusion of this exercise, you will be able to:

- Work with Meta Data objects in the DI API
- Create user-fields and user-tables in the SAP Business One database.
  - Use the UserTableMD Object to create User Tables
  - Use the UserFieldMD Object to create User Fields
  - Use the specifications for the User-Defined Table and the User-Defined Fields within from the “Course Project Exercise” (see end of the course’s “Introduction” section)

6-1 As a first small exercise add a User-Defined Field to the item table (OITM) through DI API. On your Visual Studio project create a new button called “UDF and UDT”

6-1-1 Use namespace “TB1_” as a prefix…

6-2 Add a User-Defined Table (use namespace “TB1_” as a prefix…), but do not add any fields to the table yet.

Table name: TB1_VIDS
Table description: Video Management

You will need to create an instance of the UserTablesMD object in order to add a field to the User Table. It is recommended that after you create your table you set this object variable to “Nothing” so that its properties do not inadvertently carry forward to the next table or field you are creating.

6-3 Test your application by opening the “Manage User Fields” window in SAP Business One. Check to see that the table was added.
6-4 Remove the User-Defined Table (in the SAP Business One application) you just created before. Enhance your application with the capability to remove the User-Defined Table through DI API – and then test your application to see that you can also add and delete the User-Defined Table in SAP Business One.

6-5 Add the following User-Defined Fields to your new User-Defined Table:

You will need to create an instance of the UserFieldsMD object in order to add a field to the User Table. It is recommended that after you create each field, you set this object variable to “Nothing” so that its properties do not inadvertently carry forward to the next field you are creating. Do the same thing at the end of the last user field added.

Aisle Number – Indicates in which aisle the movie is stored.
   Field Name: AISLE
   Field Description: Aisle Number
   Field Type: db_Numeric
   Field EditSize: 2

Section – Indicates the section the movie is stored in the aisle.
   Field Name: SECTION
   Field Description: Section Number
   Field Type: db_Alpha
   Field EditSize: 20

Rented – Indicates weather the movie is rented or not. Holds 2 “valid values”: Y/N.
   Field Name: RENTED
   Field Description: Rented/Available
   Field Type: db_Alpha
   Field EditSize: 1

CardCode – In case the movie is “Rented” this field will hold the CardCode of the customer who rented it otherwise it will be empty.
   Field Name: CARDCODE
   Field Description: Card Code
   Field Type: db_Alpha
   Field EditSize: 20

6-6 Test your application and make sure all your fields were added successfully.
6-7 Write data into the User-Defined Table.

6-7-1 Add about 15 records to your new User-Defined Table.

- In order to add a record, you will need to use the UserTable object. The name of this object is a bit misleading – the UserTable object actually corresponds to a record within a user table.

- When referring to specific fields within a User Table record, you must prefix the fieldname with “U_”. For example, if you have created a User-Defined Table object variable called pRecord, you could set the value of the “Make” field by adding this line of code: 
  pRecord.UserFields("U_Make").Value = “Ford”

The “Code” and “Name” must each be unique within the User Table. The “Code” is the Primary Key used to retrieve a record.

6-7-2 Your User-Defined Table could look like this:
Optional Exercise

Unit: Data Interface API
Topic: Services

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

- Work with Service Type objects

Use CompanyService to change the background color of forms for a particular company…

7-1 On your Visual Studio project create a new button called “Service Object”

7-2 Get CompanyServices object.

7-3 Get structure which reflects information in table OADM.

7-4 Set the background color to purple.

7-5 Call the method which updates the information in the SAP Business One database. See the effect in the SAP Business One application.

Please note that only forms opened after changing the background color will reflect this change.
Unit: Data Interface API

Topic: Establish a Connection to SAP Business One

At the conclusion of this exercise, you will be able to:
- Connect to an SAP Business One database

You want to develop additional functionality for SAP Business One. In a first step, you want to create a simple program to connect to an existing SAP Business One database.

1-1 Log on to SAP Business One.

1-1-1 Note the name of one database you want to log on to E.g. SBODemo_UK

1-1-2 Note one user in that database and the user's password E.g. Manager, Manager

1-2 Create a new Visual Studio project.

1-2-1 Within this project, create a form with two buttons on it. One of the buttons should be used to connect to the SAP Business One database, the other to disconnect from it.
1-2-2 Add a reference to the SAP Business One DI API COM library...

Click *Project -> Add Reference* and click on the COM tab

1-3 Code the connection to the SAP Business One database.

1-3-1 Define a variable for the Company object – ensure it is defined as a member of the add-on application class or globally. Suggestion: Create a new module and put the Company variable there. Since this is a separate module you need to either specify the module in each call or add a declaration so the other form/modules can see this.

```
Public oCompany As SAPbobsCOM.Company
```

1-3-2 Create a new Company object.

```
oCompany = New SAPbobsCOM.Company
```

1-3-3 Set the properties needed to connect to the SAP Business One database.

```
oCompany.Server = "Your server name"
oCompany.CompanyDB = "SBODemo_UK"
oCompany.UserName = "manager"
```
oCompany.Password = "B1Admin"

oCompany.DbServerType = SAPbobsCOM.BoDataServerTypes.dst_MSSQL2005

oCompany.DbUserName = "sa"

oCompany.DbPassword = "sapass"

oCompany.LicenseServer = "Your license server name"

Note DBUserName and DBPassword are not required in Version 8.8.

1-3-4 Call connect on the Company object

   retVal = oCompany.Connect

1-4 Implement error handling and success handling.

   1-4-1 If the connection succeeds, display a message box displaying a corresponding message.

   1-4-2 If the connection failed, display the error message provided by the Company object.

   If retVal <> 0 Then
       oCompany.GetLastError(retVal, retStr)
       MsgBox("Error " & retVal & " " & retStr)
   Else
       MsgBox("Connected to " & oCompany.CompanyName)
   End If

1-5 Code the disconnection from the SAP Business One database.

   If oCompany.Connected = True Then
       oCompany.Disconnect()
   End If

A further sample can be found in the SDK DI samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM DI/1.BasicOperations.
Solutions

Unit: Data Interface API

Topic: Documents Object

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

- Work with Documents objects

Create an Invoice via DI API and later create an Incoming Payment for that Invoice

2-1 On your Visual Studio project create a new button called “Invoice and Payment”

2-2 In Business One create an Order for a particular customer and a particular item.

2-2-1 First you must create a new Document object instance for the Invoice. Then you set the properties of the Documents object and the Documents_Lines ensuring the BaseEntry, BaseLine and BaseType are set.

```vba
Dim oInvoice As SAPbobsCOM.Documents

oInvoice = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oInvoices)

oInvoice.CardCode = "C2000"

oInvoice.Lines.BaseEntry = 8 'DocEntry of Sales Order
```
Add the whole document. In the case of success, you should bring up a message box telling the user the number of the newly added Sales Invoice using the method GetNewObjectCode. In case of any error, you should display a message box with an error message.

```
retVal = oInvoice.Add

If retVal <> 0 Then
    oCompany.GetLastError(retVal, retStr)
    MsgBox("Error " & retVal & " " & retStr)
Else
    MsgBox("Invoice number " & oCompany.GetNewObjectKey & " created")
    InvNum = oCompany.GetNewObjectKey
End If
```

Finally you should release the document object variables.

```
oInvoice = Nothing
retVal = ""
retStr = ""
```
2-3 Create the Incoming Payment for this Invoice

2-3-1 Create a new Payments object instance for the Incoming Payment. Then you set the properties for the CardCode, Invoice DocEntry, and we will pay via cash so we will use the properties CashAccount and CashSum.

\[
\text{Dim } o\text{IncomingPytm }\text{As } \text{SAPbobsCOM.Payments}
\]

\[
o\text{IncomingPytm }= \text{oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oIncomingPayments)}
\]

\[
o\text{IncomingPytm}.\text{CardCode }= \text{"C2000"}
\]

\[
o\text{IncomingPytm}.\text{Invoices.DocEntry }= \text{InvNum}
\]

\[
o\text{IncomingPytm}.\text{CashAccount }= \text{"SYS00000000076"}
\]

\[
o\text{IncomingPytm}.\text{CashSum }= \text{"14.10"}
\]

Note: CashAccount “SYS…” uses an internal account number in a database where account segmentation is used. If Account segmentation is not used – just use the visible account numbers.

2-3-2 Add the whole document. In the case of success, you should bring up a message box telling the user the number of the newly added Payment using the method GetNewObjectCode. In case of any error, you should display a message box with an error message.

\[
\text{retVal }= \text{oIncomingPytm.Add}
\]

\[
\text{If } \text{retVal }<> 0 \text{ Then}
\]

\[
\text{oCompany.GetLastError(retVal, retStr)}
\]

\[
\text{MsgBox("Error " }& \text{retVal }& \text{ " " }& \text{retStr})
\]

\[
\text{Else}
\]

\[
\text{MsgBox("Incoming Payment number " }& \text{oCompany.GetNewObjectKey }& \text{ " added")}
\]

\[
\text{End If}
\]
Finally you should release the document object variables.

\[ o\text{IncomingPynt} = \text{Nothing} \]

\[ \text{retVal} = "" \]

\[ \text{retStr} = "" \]

Another sample exercise can be found in the SDK samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM DI/5.OderAndInvoice.
Unit: Data Interface API

Topic: XML

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

- Work with XML

Create data as XML and checkout how to use this process to add new data to the SAP Business One database.

3-1 On your Visual Studio project create a new button called “Working with XML”
3-2  Save the Invoice created in the Documents exercise as XML.

3-2-1  Try all settings for XmlExportType property on the Company object and find the differences.

The 4 property types are

<table>
<thead>
<tr>
<th>XML ExportType</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>oCompany.XmlExportType = SAPbobsCOM.BoXmlExportTypes.xet_AllNodes</td>
<td>Export to XML all fields (both read only and read/write fields) from the database.</td>
</tr>
<tr>
<td>oCompany.XmlExportType = SAPbobsCOM.BoXmlExportTypes.xet_ExportImportMode</td>
<td>Export to XML only valid fields that support XML import (read/write fields only) from the database.</td>
</tr>
<tr>
<td>oCompany.XmlExportType = SAPbobsCOM.BoXmlExportTypes.xet_NodesAsProperties</td>
<td>Export to XML all fields as properties from the database.</td>
</tr>
<tr>
<td>SAPbobsCOM.BoXmlExportTypes.xet_ValidNodesOnly</td>
<td>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.</td>
</tr>
</tbody>
</table>

3-2-2  Save the Invoice document created in the previous exercise in Xml format

```vba
Dim oInvoice As SAPbobsCOM.Documents  
oInvoice = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oInvoices)

If oInvoice.GetByKey(5) = False Then  
oCompany.GetLastError(retVal, retStr)  
MsgBox("Failed to Retrieve Invoice" & retVal & " " & retStr)  
Exit Sub
Else

'Save the object as an xml file
oInvoice.SaveXML("C:\Program Files\SAP\SAP Business One SDK\Samples\CourseXML\Invoice.xml")
```

3-2-3  Test also the method GetBusinessObjectXmlSchema of the Company object. What kind of information does it save?

```vba
Dim schema As String  
schema = oCompany.GetBusinessObjectXmlSchema(SAPbobsCOM.BoObjectTypes.oInvoices)  
MsgBox(schema)
```

This method retrieves the XML schema used to define the structure and content of the object.
3-3 Modify the XML data obtained before and add it to the SAP Business One database.

```vbnet
oInvoice = oCompany.GetBusinessObjectFromXML("C:\Program Files\SAP\SAP Business One SDK\Samples\CourseXML\Invoice.xml", 0)
retVal = oInvoice.Add
If retVal <> 0 Then
    oCompany.GetLastError(retVal, retStr)
    MsgBox("Error " & retVal & " " & retStr)
Else
    MsgBox("Invoice number " & oCompany.GetNewObjectKey & " created")
End If
```

3-3-1 Try all files generated above and check the errors (exceptions) for details.

*Similar exercises can be found in the SDK samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM DI/7.SaveXML and COM DI/8.LoadFromXML*
Solutions

Unit: Data Interface API  
Topic: Transactions

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

- Work with transactions

Create an Order via DI API and later create an Invoice that is based in that Order, Documents exercise done before.

This time open a transaction before and close it afterwards.

There is no additional “Solution” to this exercise.
Solutions

Unit: Data Interface API
Topic: Using General Objects

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

- Use the data browser object to browse through a set of data
- Use the record set object

Create an application to navigate through all customers.

You will use the Browser property of the BusinessPartners object. Add the navigation buttons to your form and provide the coding so that the user can browse through the customers.

5-1 On your Visual Studio project create a new button called “General Objects”

```
5-2 Create a new form in your Visual Studio application containing a text box where you will show the Business Partners Card Code and four buttons: first, previous, next and last.
```
5-3 Create a Recordset object and set the Browser property of the BusinessPartners object to this Recordset. Be sure that your application only includes customers, not Leads or Vendors (Suppliers).

```
oRecordSet = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.BoRecordset)
oBusinessPartner = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oBusinessPartners)
oRecordSet.DoQuery("Select CardCode from OCRD where CardType = 'C'")
oBusinessPartner.Browser.Recordset = oRecordSet
```

5-3-1 Add the code to all four of the buttons so that the user can navigate backwards and forward through the customers.

For example to move First use the following code. It needs to be changed slightly for the other 3 actions.

```
If oBusinessPartner.Browser.BoF = False Then
  oBusinessPartner.Browser.MoveFirst()
  FillField()
End If
```

5-4 Test your changes. Be sure to include the following scenarios:

5-4-1 Click the “First Record” button, then click it again. Try the same thing with the “Last Record” button.

5-4-2 Click the “First Record” button, then click the “Previous Record” button.

5-4-3 Click the “Last Record” button, then click the “Next Record” button.

5-4-4 If any of these scenarios raises an error, add code that will fix the error. Then test the application again.

A similar exercise can be found in the SDK samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM DI/1.BasicOperations
Solutions

Unit: Data Interface API

Topic: Meta Data

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

- Work with Meta Data objects in the DI API

Create user-fields and user-tables in the SAP Business One database.

- Use the UserTableMD Object to create User Tables
- Use the UserFieldMD Object to create User Fields

6-1 As a first small exercise add a User-Defined Field to the item table (OITM) through DI API. On your Visual Studio project create a new button called “UDF and UDT”

6-1-1 Use namespace “TB1_” as a prefix…

```vba
Dim oUDF As SAPbobsCOM.UserFieldMD
oUDF = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oUserFields)
oUDF.TableName = "OITM"
oUDF.Name = "TB1_Course"
oUDF.Description = "Course UDF"
oUDF.Type = SAPbobsCOM.BoFieldTypes.db_Alpha
oUDF.Editsize = 20

RetVal = oUDF.Add
If RetVal <> 0 Then
    oCompany.GetLastError(RetVal, retStr)
```
6-2 Add a User-Defined Table (use namespace “TB1_” as a prefix…), but do not add any fields to the table yet.

```vbscript
Dim oUsrTble As SAPbobsCOM.UserTablesMD
oUsrTble = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oUserTables)

oUsrTble.TableName = "TB1_DVD"
oUsrTble.TableDescription = "DVD Management"
retVal = oUsrTble.Add
If retVal <> 0 Then
    oCompany.GetLastError(retVal, retStr)
    MsgBox("Error " & retVal & " " & retStr)
Else
    MsgBox("UDT Added")
End If
oUsrTble = Nothing
```

6-3 Test your application by opening the “Manage User Fields” window in SAP Business One. Check to see that the table was added.

6-4 Remove the User-Defined Table (in the SAP Business One application) you just created before. Enhance your application with the capability to remove the User-Defined Table through DI API – and then test your application to see that you can also add and delete the User-Defined Table in SAP Business One.

```vbscript
If oUsrTble.GetByKey("TB1_DVD") = True Then
    retVal = oUsrTble.Remove
End If
If retVal <> 0 Then
    oCompany.GetLastError(retVal, retStr)
    MsgBox("Error " & retVal & " " & retStr)
Else
    MsgBox("UDT Removed")
End If
```
Add the following User-Defined Fields to your new User-Defined Table:

Aisle Number – Indicates in which aisle the movie is stored.
    Field Name: AISLE
    Field Description: Aisle Number
    Field Type: db_Numeric
    Field EditSize: 2

Section – Indicates the section the movie is stored in the aisle.
    Field Name: SECTION
    Field Description: Section Number
    Field Type: db_Alpha
    Field EditSize: 20

Rented – Indicates whether the movie is rented or not.
    Holds 2 “valid values”: Y/N.
    Field Name: RENTED
    Field Description: Rented/Available
    Field Type: db_Alpha
    Field EditSize: 1

CardCode – In case the movie is “Rented”
    This field will hold the CardCode of the customer who rented it otherwise it will be empty.
    Field Name: CARDCODE
    Field Description: Card Code
    Field Type: db_Alpha
    Field EditSize: 20

Same process as adding a user-defined field to a System table except we use the correct notation for a User Defined Table i.e. using @

`oUDF.TableName = "@TB1_DVD"`

Test your application and make sure all your fields were added successfully.
Write data into the User-Defined Table.

Add about 15 records to your new User-Defined Table.

```vbnet
Dim oUserTable As SAPbobsCOM.UserTable
oUserTable = oCompany.UserTables.Item("TB1_DVD")
oUserTable.Code = "1"
oUserTable.Name = "Avatar"
oUserTable.UserFields.Fields.Item("U_AISLE").Value = "2"
oUserTable.UserFields.Fields.Item("U_SECTION").Value = "Science Fiction"
oUserTable.UserFields.Fields.Item("U_RENTED").Value = "N"

retVal = oUserTable.Add
If retVal <> 0 Then
    oCompany.GetLastError(retVal, retStr)
    MsgBox("Error " & retVal & " " & retStr)
Else
    MsgBox("Record Added")
End If

oUserTable = Nothing
```

Your User-Defined Table could look like this:

![User-Defined Table Image]

A similar solution can be found in the SDK samples (in the SDK Folder – see Appendix “SDK Installations” for more information), ...
\COM UI DN\COM UI DNVB.NET\UIDIBasicApp\CreateUserTables
Solution to Optional Exercise

Unit: Data Interface API
Topic: Services

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

- Work with Service Type objects

Use CompanyService to change the background color of forms for a particular company...

7-1 On your Visual Studio project create a new button called “Service Object”

7-2 Get CompanyServices object.

7-3 Get structure which reflects information in table OADM.

7-4 Set the background color to purple.
Call the method which updates the information in the SAP Business One database. See the effect in the SAP Business One application

```vba
Dim oCompanyService As SAPbobsCOM.CompanyService
Dim oCompanyInfo As SAPbobsCOM.CompanyInfo
Dim oCompanyAdminInfo As SAPbobsCOM.AdminInfo

oCompanyService = oCompany.GetCompanyService
oCompanyAdminInfo = oCompanyService.GetAdminInfo
oCompanyAdminInfo.CompanyColor = 3

oCompanyService.UpdateAdminInfo(oCompanyAdminInfo)
```

A solution (+ more sample code around services) can be found in the SDK samples (in the SDK Folder – see Appendix “SDK Installations“ for more information),

**COM DI/ 11. Basic Company Settings**
User Defined Objects (UDOs)

Contents:
- SAP Business One Objects
- Explain why UDOs may make sense
- Implementing UDOs step-by-step
- Use DI API's GeneralService to maintain UDO data
User Defined Objects: Unit Objectives

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

- Describe the SAP Business One Objects
- Why UDOs may make sense
- Implement UDOs step-by-step
- Use UDOs within an Add-on
- 1 Course Overview
- 2 SDK Introduction
- 3 The Data Interface API (short look on JCo + DI Server)
- 4 User-Defined Objects (UDO)
- 5 The User Interface API
- 6 Packaging, Add-On Administration and Licensing
At the conclusion of this topic, you will be able to:

- Explain the architecture of UDOs
- Describe available services that reduce development efforts
User-Defined Business Objects – Benefits

- User-Defined Business Objects will be added to the SAP Business One application objects collection.
- User-Defined Business Objects come with a set of basic functionalities (named “services”) which are common for any Business Object in SAP Business One.

May be a very good solution to add new business logic to the SAP Business One application.
- Fast way to develop Add-ons since a major part of the implementation is provided automatically.
- Fast way to develop any Add-On working with data from User-Defined Tables with database format used for UI.
**Object Types**

SBO application supports 2 main types of objects:
- Master Data Objects – e.g. Business Partner
- Documents – e.g. Sales Order

The Document object supports methods that are not implemented in Master Data objects like:
- Document numbering (Serial Numbers)
- Close
## Predefined Services for Business Objects

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Add a new record of the object to the DB.</td>
</tr>
<tr>
<td>Update</td>
<td>Update the fields of the object in the DB.</td>
</tr>
<tr>
<td>Find</td>
<td>Supports “Choose From List” for the object.</td>
</tr>
<tr>
<td>Close</td>
<td>Only relevant for „Document Data“ type User-Defined Objects</td>
</tr>
<tr>
<td>Cancel</td>
<td>Only changes the record’s status to “Cancel = Y”.</td>
</tr>
<tr>
<td>Delete</td>
<td>Master Data – deleting record, Doc – no effect.</td>
</tr>
<tr>
<td>Manage Series</td>
<td>Relevant to document objects. Adding the object to the Document Numbering form and managing the series for that object.</td>
</tr>
<tr>
<td>History</td>
<td>Creates a log table for the object and saves its history.</td>
</tr>
<tr>
<td>Default Form</td>
<td>Creates a default form for the object which manages all the services.</td>
</tr>
<tr>
<td>Year Transfer</td>
<td>Copying the tables and the records in the Year Transfer operation (only released for the Netherlands and Israel).</td>
</tr>
</tbody>
</table>

- The above are SAP Business One services available for partners new object.
Information flow between Add-ons and SAP Business One using DI API and UI API

- In the current situation the partner usually needs to implement the connection between the DI and the UI API.
After running the UDO wizard your object is registered to SAP Business One services.
Flow between Add-ons and SAP Business One using UI API and UDOs including Impl. DLL

Predefined service register for the object
1. Add, Update...
2. History log
3. Series
4. User signature
5. Year Transfer

GUI Layer

<table>
<thead>
<tr>
<th>System Form</th>
<th>User Defined Form</th>
</tr>
</thead>
</table>

Business Layer

<table>
<thead>
<tr>
<th>System Objects</th>
<th>User Defined Object</th>
</tr>
</thead>
</table>

DB Layer

<table>
<thead>
<tr>
<th>System Table</th>
<th>User Defined Table</th>
</tr>
</thead>
</table>

UI Add-on

Create user form and connect it to the Defined object

UDO implementation
You should now be able to:
- Explain the architecture of UDOs
- Describe available services that reduce development efforts
At the conclusion of this topic, you will be able to:

- Implement UDOs step-by-step
- Use DI API's GeneralService to maintain UDO data
You can go through “Order Meal” sample provided with the UDO documentation.

Choose from the UDO documentation:

- SAP Business One – User Defined Object → Samples → Document Type Sample – Meal Ordering Object → Stage...
Define Base Tables

- Create User Table/s with User Fields that will hold the data for your new business object.
- Use SAP Business One application
  (Tools ➔ User Defined Fields ➔ Manage User Fields ➔ User Tables)
- Use the DI API Metadata object
  (UserTablesMD and UserFieldsMD)
- Do not forget to choose the suitable object type.
The registration wizard helps you to register your User Defined Objects.

The registration is per company.

Choose from SAP Business One menu:

*Tools ➔ User Defined objects ➔ Registration Wizard*

- Add a new object: Inserts a new User Defined Object
- Update an existing object: Updates an existing object
- Unregister an existing object: Removes the Object registration (UDO)
- Delete an existing object: Removes the Object registration and clears the object’s tables.
Set A Unique ID for your object
- (use namespace)
- Set the object type
- Set the Header/Parent table

Register for services.
- Add and Update are the basic services and cannot be deselected.
UDO Registration Steps 4 – Fields for “Find” and 5 – select child/son tables

If the Find service was selected:

- Select the fields from the parent table to be displayed in the find form.

Select the Child tables of the object.

- Only suitable tables are displayed in the list.
UDO Registration Step 6 – Optionally define “Default Form”

If the Default Form service was checked:

- Select fields from parent table to be displayed in the default form.
- Select fields from one child table.
UDO Registration Step 7 – Optional Implementation DLL

Set the Extension DLL file (optional).

- Please refer to slide number 21 for further information about the extension DLL.
The UDO wizard provides the option to create a default UI Form. Use this option in case:
- You need to test on your object.
- You need a quick solution.

Load your UI form:
Tools → Default Forms → Your form

Limitation:
- Only 1 child table supported.
How to define a UDO through DI API?

```vba
Dim oUserObjectMD As UserObjectsMD
oUserObjectMD = _
 oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oUserObjectsMD)

oUserObjectMD.Code = "TT_MD"
oUserObjectMD.Name = "TEST_MD"

oUserObjectMD.ObjectType = _
  SAPbobsCOM.BoUDOObjType.boud_MasterData
oUserObjectMD.TableName = "T_MD"  ' Main user table (same type as the UDO)

oUserObjectMD.ChildTables.TableName = "T_MD1"  ' First child user table

' Add 2nd line; first line in a already exists by default
oUserObjectMD.ChildTables.Add()
oUserObjectMD.ChildTables.TableName = "T_MD2"  ' Second child user table
```

- Use the new interface for creating a form
How to define a UDO through DI API?
(continued)

Dim c_Yes As SAPbobsCOM.BoYesNoEnum = BoYesNoEnum.tYES

' Configure Services
oUserObjectMD.CanCancel = c_Yes
oUserObjectMD.CanClose = c_Yes
oUserObjectMD.CanCreateDefaultForm = c_Yes ' Need to specify columns
oUserObjectMD.CanDelete = c_Yes
oUserObjectMD.CanFind = c_Yes ' Need to specify columns
oUserObjectMD.CanLog = SAPbobsCOM.BoYesNoEnum.tNO
oUserObjectMD.CanYearTransfer = SAPbobsCOM.BoYesNoEnum.tNO
oUserObjectMD.ManageSeries = c_Yes

' Columns added in the ChooseFromList form, repeat this 3 lines for each column
oUserObjectMD.FindColumns.ColumnAlias = "Code"
oUserObjectMD.FindColumns.ColumnDescription = "Code"

' ... add Columns for Default Form in the same way ...
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "U_MyName"
oUserObjectMD.FindColumns.ColumnDescription = "My Name."

' Add the UDO
lRetCode = oUserObjectMD.Add()
Steps to writing your own object’s business logic unit:

- Write a class that inherits from CSBOBusinessObject.
- Export CreateObject function (dll entry point).
- Implement Destroy and Close functions (pure virtual).
- Overwrite any desired virtual function.
- Call the base class functions to get the default behavior.
- Use the interface functions to do your work.
- Register the dll in the registration wizard.

Important:

- You must implement in C++.
- You can only register one DLL per UDO.
- Pay attention to the namespaces to avoid conflicts
- When a user activates a UDO, the SAP Business One application loads the DLL in memory.

…find a description how to implement such a DLL file step-by-step in the notes below…

You are able to overwrite the implementation for your object

- In case you want to add actions to the default behavior.
- In case you want to replace the default behavior.
Include the header files:

- SboBusinessObject.h - Defines CSboBusinessObject, base Class for SBO objects.
- SboDataAccessGate.h - Defines CSboDataAccessGate, BD interface.
- SboCondition.h - Elements for query conditions.
- SBO_Types.h
- __SBOERR.h - Application errors definition.
- _AppObjects.h - List of SBO objects id’s.
The new interface includes the GeneralService and a set of 4 supporting objects

One interface is good for all UDOs (Master Data and Document)

The new interface provides access to UDO data:
- Add records
- Find records
- Delete records
- Cancel / Close document
- Invoke partner method (to invoke a custom method written in an implementation DLL for your UDO)
- GetDataInterfaceFromXMLFile / GetDataInterfaceFromXMLString (creates object from XML file or string)
- Get/Set property – for Getting and Setting table field values (for most fields that are auto generated, only Get is implemented)
DI General Service – Objects

- **General Data** – Represents a single row in a database table of a UDO, or in a child table of the UDO
- **GeneralDataParams** – Holds the keys to rows in database tables linked to a UDO data. This object is used to pass keys to and from GeneralService methods
- **GeneralCollectionParams** – A collection of GeneralDataParams objects
- **GeneralDataCollection** – A collection of GeneralData objects, each of which represents a row in a child user table for a specific row of the main table of a UDO
- **InvokeParams** – Holds a single, string value. This object is used to pass a parameter to or receive a return value from the Invoke method of the GeneralService service.
Sample is for Document UDO type, Master Data is quite similar
DI General Service – Code Sample
Update Document

SAPbobsCOM.GeneralService oDocGeneralService;
SAPbobsCOM.GeneralData oDocGeneralData;
SAPbobsCOM.GeneralDataCollection oDocLinesCollection;
SAPbobsCOM.GeneralData oDocLineGeneralData;
SAPbobsCOM.GeneralDataParams oGenralParameter;

// Retrieve the relevant service
oDocGeneralService = (SAPbobsCOM.GeneralService)oCompService.GetGeneralService("MyDocUDO");

// Get by key - header record
oGenralParameter = (SAPbobsCOM.GeneralDataParams)oDocGeneralService.GetDataInterface
(SAPbobsCOM.GeneralServiceDataInterfaces.gsGeneralDataParams);

oGenralParameter.SetProperty("DocEntry", "1");

// Update - Add Lines to the child tables, Insert Values to the Lines properties
oDocLinesCollection = (SAPbobsCOM.GeneralDataCollection)oDocGeneralData.Child("SAP_DOCL");

// Add Line
oDocLineGeneralData = oDocLinesCollection.Add();
oDocLineGeneralData.SetProperty("U_ItemCode", "Item2");
oDocLineGeneralData.SetProperty("U_Quantity", "2");
...

// Update DocTotal in the header
oDocGeneralData.SetProperty("U_DocTotal", "50");

// Update the MD UDO
oDocGeneralService.Update(oDocGeneralData);
DI General Service – Code Sample
Delete Document

SAPbobsCOM.GeneralService oDocGeneralService;
SAPbobsCOM.GeneralData oDocGeneralData;
SAPbobsCOM.GeneralDataParams oGeneralParameter;

// Retrieve the relevant service
oDocGeneralService = (SAPbobsCOM.GeneralService)oCompService.
GetGeneralService("MyDocUDO");

// Get by key – header record
oGeneralParameter = (SAPbobsCOM.GeneralDataParams)oDocGeneralService.GetDataInterface(SAPbobsC
OM.GeneralServiceDataInterfaces.gsGeneralDataParams);
oGeneralParameter.SetProperty("DocEntry", "3");

// Delete whole record (Header and Lines)
oDocGeneralService.Delete(oGeneralParameter);

- Similar code for Cancel and Close (which are relevant for document object type only!)
SAPbobsCOM.GeneralService oDocGeneralService;
SAPbobsCOM.GeneralCollectionParams oDocsCollectionParams; //List

// Retrieve the relevant service

oDocGeneralService = (SAPbobsCOM.GeneralService)oCompService.GetGeneralService("MyDocUDO");

// Get the List

oDocsCollectionParams = oDocGeneralService.GetList();


MessageBox.Show("There are " + oDocsCollectionParams.Count + " documents");
Refer to the blog:

**Simple Sample Blog** (Accessing UDO in DI API):

**How to use UDO services in DI Server:**

- Code sample can be downloaded via a link provided in the blog
Features related to UI API are going to be discussed in the UI API unit.
You should now be able to:
- Implement UDOs step-by-step
- Use DI API's GeneralService to maintain UDO data

- Connecting has already been practiced in the introduction unit…
User Defined Objects: Unit Summary

You should now be able to:
- Describe SAP Business One Objects
- Why UDOs may make sense
- Implementing UDOs step-by-step
- Use DI API's GeneralService to maintain UDO data
Exercises

Unit: User Defined Object

Topic: Basics

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

- Add a User Defined Table of type UDO
- Register your UDO
- Fill your UDO with data

1-1 Define the User defined table

1-1-1 Take the table you defined in the DI Exercises (Exercise 6) TB1_DVD and define it a user defined object type Master Data. You can do this via the SAP Business One application or via the DI API as highlighted in the DI Exercises.

You will first need to delete the user table created previously and recreate it as a User Defined Object. If you wish to keep the data you can first export it to excel and re-enter it in after again.

1-1-2 Define the user defined fields again for this table (from DI exercises)
1-2 Register the UDO

This can be done via the SAP Business One Objects Registration Wizard or via the DI API

UDO Code: TB1_DVDAvail

UDO Name: TB1_DVDAvailability

Select services Cancel, Delete and Find

Find columns Code, Name, U_Section, U_Aisle, U_Rented, U_CardCode

1-3 Enter data into the UDO using the General Service
Solutions

Unit: User Defined Object
Topic: Basics

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

- Add a User Defined Table of type UDO
- Register your UDO
- Fill your UDO with data

1-1 Define the User defined table

1-1-1 Take the table you defined in the DI Exercises (Exercise 6) TB1_DVD and define it a user defined object type Master Data. You can do this via the SAP Business One application or via the DI API as highlighted in the DI Exercises.

1-1-2 Define the user defined fields again for this table (from DI exercises)
1-2 Register the UDO

```vba
Dim oUserObjectMD As SAPbobsCOM.UserObjectsMD

oUserObjectMD = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oUserObjectsMD)

oUserObjectMD.Code = "TB1_DVDAvail"
oUserObjectMD.Name = "DVDAvailability"
oUserObjectMD.ObjectType = SAPbobsCOM.BoUDOObjType.boud_MasterData
oUserObjectMD.TableName = "TB1_DVD"

oUserObjectMD.CanCancel = SAPbobsCOM.BoYesNoEnum.tYES
oUserObjectMD.CanClose = SAPbobsCOM.BoYesNoEnum.tYES
oUserObjectMD.CanDelete = SAPbobsCOM.BoYesNoEnum.tYES
oUserObjectMD.CanFind = SAPbobsCOM.BoYesNoEnum.tYES

oUserObjectMD.FindColumns.ColumnAlias = "Code"
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "Name"
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "U_SECTION"
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "U_AISLE"
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "U_RENTED"
oUserObjectMD.FindColumns.Add()
oUserObjectMD.FindColumns.ColumnAlias = "U_CARDCODE"
oUserObjectMD.FindColumns.Add()

retVal = oUserObjectMD.Add()
```

1-3 Enter data into the UDO using the General Service

```vba
Dim oGeneralService As SAPbobsCOM.GeneralService
Dim oCompanyService As SAPbobsCOM.CompanyService
Dim oGeneralData As SAPbobsCOM.GeneralData

oCompanyService = oCompany.GetCompanyService
oGeneralService = oCompanyService.GetGeneralService("TB1_DVDAvail")

oGeneralData = oGeneralService.GetDataInterface(SAPbobsCOM.GeneralServiceDataInterfaces.gsGeneralData)
oGeneralData.SetProperty("Code", "32")
oGeneralData.SetProperty("Name", "Gran Torino")
oGeneralData.SetProperty("U_SECTION", "Drama")
oGeneralData.SetProperty("U_AISLE", "8")
oGeneralData.SetProperty("U_RENTED", "Y")
oGeneralData.SetProperty("U_CARDCODE", "Kim Kingston")

oGeneralService.Add(oGeneralData)
```
The User Interface API

Contents:
- API overview
- Establishing a connection to the user interface
- Working with system forms
- Creating and working with custom forms
- Menus
- Event handling
The User Interface API: Unit Objectives

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

- Explain what the User Interface API is
- Explain how to establish a connection to a running SAP Business One application
- Explain how the API interacts with the SAP Business One client
- Add menu entries
- Work with existing SAP Business One forms
- Create forms and integrate them into SAP Business One GUI

Steps:

- General introduction
- Connecting to User Interface API (UI API)
- Implementing functionality required to ensure seamless integration (events, menus etc)
- Modifying existing forms (how and when)
- Developing and connecting own forms
- …how to connect own forms to data from the database
- 1 Course Overview
- 2 SDK Introduction
- 3 The Data Interface API (brief look at JCo + DI Server)
- 4 User-Defined Objects (UDO)
- 5 The User Interface API
- 6 Packaging, Add-On Administration and Licensing
You want to:

- Perform additional checks in SAP Business One
- Enhance SAP Business One by seamlessly integrating additional functionality
The User Interface API

**Topic 1: UI API Introduction**

- Topic 2: Add-On Basics
- Topic 3: Creating Forms
- Topic 4: ItemEvents, Event Filtering (and more)
- Topic 5: Menus
- Topic 6: Data Binding
- Topic 7: Use UDO in Add-On
- Topic 8: Additional Events
- Topic 9: Additional Objects
- Topic 10: UI API – Additional Information
UI API Introduction: Topic Objectives

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

- How User Interface API works
- How to connect to the SAP Business One application through UI API
The UI API exposes user interface elements of the SAP Business One front-end:
- Respond to internal events in the SAP Business One client application
- Add or modify menus
- Add new forms
- Modify existing forms
- Get or set values on a form

By using the event mechanism a 3rd party application can react to user interactions with the SAP Business One application.

Summary: UI API provides the capability for seamless integration maintaining the uniform “look and feel” of SAP Business One.
UI API Introduction – Characteristics

- DCOM executable running on the client machine – 1 instance per Windows session
- Connected to all instances of the SAP Business One application
- Gives access to user interface elements within the SAP Business One application via a COM interface
- Sends events (usually originating) from SAP Business One GUI elements (items) to your event sink / event handler

- Enables add-on executables to customize or extend the SAP Business One client application
- Relatively low-level – most large-scale/complex changes require significant programming effort
Multiple add-ons (from several vendors) may be used alongside the SAP Business One application to provide a complete solution.

- 3rd party applications can modify SAP Business One GUI through UI API.
- 3rd party applications can get events from SAP Business One through UI API to react to user interaction.
UI API add-ons are launched by SAP Business One and then have to connect to the UI API within a timeout limit of approx. 10 seconds.

“SAP Business One checks for registered add-ons”

- Add-On Administration settings control which add-ons are started for a user – Refer to the „Add-On Packaging, Administration & Licensing” unit
When SAP Business One is launched, it starts the UI API and connects with it before any add-ons are started.

If add-ons are registered to start within a user session:

1. SAP Business One links to the UI API and establishes an event-sink for them

2. SAP Business One starts 3rd party application A and passes a command line parameter to it
   2.a 3rd party application A creates an Application object that has a counterpart in the UI API
   2.b 3rd party application A provides an event sink for events to be fired from the Application object on the UI API side
   2.c The UI API application object registers itself in the IAppLink object for bidirectional communication

3. The same set of steps is repeated for 3rd party application B
SAP Business One starts add-on applications registered for automatic start-up in the order determined by the system administrator (see section “Creating a package”).

The add-ons establish connections to the UI API and register event sinks.

When an event occurs in SAP Business One UI, it is passed to add-ons which created event sinks for such events, one at a time.
Every add-on action is reflected in the SAP Business One application via the UI API.

The same mechanism is used in the other direction, from the SAP Business One application to add-ons.
The User Interface API (UI API) is a collection of COM objects that provide access to:

- forms
- controls within these forms
- menus

Note: The “Main Menu” and the status bar are forms as well
The Application object (reflecting the IApplication interface on the UI API DCOM server side) provides access to forms, controls within forms, menus and the main window (desktop).

- “Forms” and “Menus” are collections accessible via the Application object.
  - „Menus“ holds a snapshot of the menu items currently available – both visible and not visible.
  - „Forms“ holds the collection of the currently available forms – both visible and not visible.
  - It is not possible to explicitly display a new instance of a specific type of system form, e.g. Sales Quotation. New system forms can be instantiated indirectly, for example by activating the corresponding menu item.

- „Form“ object is a representation of a form, both system and user-defined
  - „Item“ object represents a window control - contents, position, size, visibility and other attributes can be modified
  - „DataSources“ collection - objects which hold data for form items, designed to provide efficient data handling separately from UI presentation

- Desktop object/property - use it to change e.g. the background image
You can view technical information related to forms, items (controls), and corresponding database tables/fields by selecting View > System Information and mouse over the items (controls) in question.

The information is shown in the lower left corner of the screen:

- Form Type (string, but appears as a number for system forms)
- Item UID (string, but appears as a number for system items)
- Pane - current layer linking items with folders (tabs) – see later in this unit
- Database table name
- Database field name
- Menu item unique id

Note:

- Database details are not available for items which display information that is:
  - Calculated within the user interface
  - Sourced from more than one database field – for example amounts combine the float value with the currency code, e.g. „EUR 7.59“. The information in the status bar will contain a „variable“ ID.

- The information displayed relates to the position of the mouse pointer, not necessarily the item which currently has input focus
Most UI API code is event driven. Events are usually fired in response to user actions within the SAP Business One application.

AppEvent, ProgressBarEvent + StatusBarEvent will always be forwarded to add-ons, i.e. they cannot be filtered out.

Other types of events can be filtered (ItemEvent) or must be added to the (ItemEvent) event filter (click on menu).

Note:
- It is the developer’s responsibility to make sure that their code will handle each application event successfully
- AppEvent events must be processed by your application. Refer to the Standards & Guidelines document for more information
Your application connects to the SAP Business One client through the SboGuiApi object using the `Connect` method.

- If a connection has been established, the `GetApplication` method of the SboGuiApi object grants access to an application instance, which must be used to access the application’s containers (for example, menus or forms) for event manipulation and for property settings.

- The user interface can be accessed using the objects that exist within User Interface API objects.

- Using the connection string supplied by the SAP Business One application as a command line parameter for `Connect()` makes sure that the add-on gets connected to the correct instance of the SAP Business One application.
In VB.NET, the SBO_Application object has to be declared with the WithEvents modifier for it to support event handling.

The connection to the UI API requires:

- **Connection String**
  - Development mode - supplied by SAP; preferably use in MS Visual Studio debug project settings as “command line parameter” rather than hard-coding it. See the “How to” section in the SDK Helpcenter.
  - Runtime mode - supplied by the SAP Business One application as a command line parameter

- An **Add-On Identifier String** - allows the SAP Business One License Service to recognize your Add-On. To create the identifier, use the Add-On Identifier Generator available from the SAP Business One application (License Administration).

- Licensing and AddOnIdentifiers are discussed later in the course
Add-ons generally only need re-compilation to run on newer versions of the UI API

- Always test an add-on thoroughly before deploying in a productive environment
- Due to bug fixes in the SDK, incorrect code might “work” on an older version, but it might encounter exceptions after an upgrade
You should now be able to explain:
- how the User Interface API works
- how to connect to the SAP Business One application through UI API
Now do the first UI API exercise and try to connect in exercise 1-1.
## Add-On Basics: Unit Overview Diagram

<table>
<thead>
<tr>
<th>The User Interface API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: UI API Introduction</td>
</tr>
<tr>
<td><strong>Topic 2: Add-On Basics</strong></td>
</tr>
<tr>
<td>Topic 3: Creating Forms</td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>Topic 5: Menus</td>
</tr>
<tr>
<td>Topic 6: Data Binding</td>
</tr>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
<tr>
<td>Topic 9: Additional Objects</td>
</tr>
<tr>
<td>Topic 10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
Add-On Basics: Topic Objectives

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

- Explain how to use single sign-on to connect to both the User Interface API and the Data Interface API
- Explain how to use the option to get the Data Interface API connection parameters through UI API
- Explain the use of events to ensure that your add-on is synchronized with SAP Business One
Add-On Basics
Multiple Add-Ons using DI API

SAP Business One version 2007 introduced the option to share a DI API connection across add-ons:

Call `Application.Company.GetDICompany()` in UI API to get a reference to the DI API Company object.

SAP Business One Application (Loading OBServerDLL at 1st add-on connection)  DI API

UI API

UI API + DI API connection; memory consumption > ~1.5MB

Add-on 1 (first add-on to connect)

UI API + DI API connection; mem. > ~1.5MB

Add-on 2

UI API + DI API connection; mem. > ~1.5MB

Add-on N
Add-On Basics
Single Sign-On

- In SAP Business One prior to version 2007, each UI API add-on needing the DI API must have its own DI API connection.

- DI API connection can be set up reusing the logon information of the existing UI API connection.

User credentials are not directly available.

Documentation and sample code is included in the SDK help: Getting Started -> Single Sign On.

“Single Sign-On” is still a valid option to connect to both UI API and DI API, but the “Multiple Add-On” feature should be preferred.
Add-On Basics
Single Sign-On (Code example)

*After connecting to UI, but before connecting to DI:
  *Acquire the connection context cookie from the DI API
    Dim sCookie As String
    sCookie = oDICompany.GetContextCookie

*Retrieve the connection context string from the
  *UI API using the acquired cookie.
    Dim conStr As String
    conStr = SBO_Application.Company.GetConnectionContext(sCookie)

*Set the connection context information to the DI API.
  ret = oDICompany.SetSboLoginContext(conStr)
  If Not ret = 0 then
    Exit Sub 'the operation has failed.
  End If

*Establish the connection to the company database.
  ret = oDICompany.Connect()
Add-On Basics
Details about AppEvents (mandatory!)

Language Change
- Occurs when the user changes the display language in the company settings
  (Administration > System Initialization > General Settings)
  => The Modules menu is rebuilt and any additional add-on menus removed. You must
  handle the Language Change events in your add-on and reapply your menu changes
  using the new language selected by the user.

Shutdown of SAP Business One / Company Change / Add-On shutdown in
Add-On Manager (“UIServerTermination”)
- Shutdown occurs when the user closes the SAP Business One application
- Company change occurs when the user selects another company within the same user
  interface session
- UIServerTermination is fired when an AddOn is requested to stop through Add-On
  Manager (Administration > Add-ons > Add-on Manager)
  => You must do clean-up work (remove menus (UI Server Termination), close windows,
  …) and stop your add-on (e.g. call End in VB.NET)
Add-On Basics: Topic Summary

You should now be able to:
- Explain how to use single sign-on to connect to both the User Interface API and the Data Interface API
- Explain how to use the option to get the Data Interface API connection parameters through UI API
- Explain the use of events to ensure that your add-on is synchronized with SAP Business One
Add-On Basics: Exercise

You should now be ready to add these basic features to your Add-On in an exercise:

- “Single Sign-On” (or the alternative way to connect to DI API as well)
- Handlers for the (mandatory) AppEvents
- …more Event handlers – if you like…
# Creating Forms: Unit Overview Diagram

## The User Interface API

<table>
<thead>
<tr>
<th>Topic 1: UI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Add-On Basics</td>
</tr>
<tr>
<td><strong>Topic 3: Creating Forms</strong></td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>Topic 5: Menus</td>
</tr>
<tr>
<td>Topic 6: Data Binding</td>
</tr>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
<tr>
<td>Topic 9: Additional Objects</td>
</tr>
<tr>
<td>Topic 10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
Creating Forms: Topic Objectives

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

- Explain how to create new forms and items
- Use Screen Painter Add-On to design forms
- Save and load forms using XML
Creating Forms – User Forms

User form is a form that you add to Business One using the UI API

There are several ways to create a user form

- Code it step-by-step
- Use the Screen Painter add-on

You must assign a type and a unique ID (UID) which must be prefixed with your company’s namespace, e.g. SAP_AsstMD stands for the form type of Fixed Asset Master Data in SAP Fixed Asset Add-On.
Creating Forms – General Remarks

- Adding items to user forms
  - Unique id “1” and “2” will inherit Business One’s behavior for “OK” and “Cancel” buttons
  - Positioning
  - LinkTo property

- Default tab order is based on the order in which items are added to a form, but can be changed later on; see UI API helpfile or Appendix 4 for details

- DataSources will improve performance

- XML layout improves form load speed
Using FormCreationParams is the preferred method, although version 6.5 style still works:

- Dim oForm As SAPbouiCOM.Form
- Dim creationPackage As SAPbouiCOM.FormCreationParams

Create the FormCreationParams object
creationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectType.cot_FormCreationParams)

'Specify the parameters in the object
creationPackage.UniqueID = "MP_MyFormID"
creationPackage.FormType = "MP_MyFormType"
creationPackage.BorderStyle = SAPbouiCOM.BoFormBorderStyle.fbs_Fixed

' Add the form to the SBO application
oForm = SBO_Application.Forms.AddEx(creationPackage)

'Set the form title and visibility
'oForm.Title = "Hello World"
oForm.Visible = True
Creating Forms – Create Items on the Form (Sample)

```vba
Dim oItem As SAPbouiCOM.Item
Dim oButton As SAPbouiCOM.Button

'Add button, buttons with UID 1 and 2 should be OK and Cancel
oItem = oForm.Items.Add("1", it_BUTTON)
oButton = oItem.Specific
oButton.Caption = "&OK"
'Set Size and Location:
oItem.Top = 200
oItem.Left = 20
oItem.Width = 70
oItem.Height = 19

oItem = oForm.Items.Add("2", it_BUTTON)
oButton = oItem.Specific
oButton.Caption = "&Cancel"
'Set Size and Location:
oItem.Top = 200
oItem.Left = 95
oItem.Width = 70
oItem.Height = 19
```

- For more information, see the User Interface API online help for the Form object
Creating Forms – Screen Painter

Best method for initial layout design

Screen Painter
- easy-to-use graphical form design tool
- add-on application and part of the SDK tools
- independent on any development environment
- lets you create forms with SAP Business One look and feel
- generates XML form definitions which load fast and are easy to use with UI API’s XML handling features

Available to install like any other add-on
Once running, launch it from the menu: Tools > Screen Painter
- The Screen Painter is a graphical design tool that enables you to quickly and easily create user forms for SAP Business One
- The Screen Painter is part of SDK and installs and runs as an add-on application
- Launch it from Tools > Screen Painter
Creating Forms – Working with XML

Why use XML?
A series of operations is replaced by a single batch operation. This means less code and better performance!

Saving a form layout to an XML file
sXML = oForm.GetAsXML() 'get XML string
oXML.loadXML(sXML) 'load XML into DOM document obj.
oXML.save (App.Path & ":Form.xml") 'save file

Updating ANY form (or loading a user form) from an XML file
SBO_Application.LoadBatchActions (oXMLDoc.xml) 'load string through one call

Preferred user form loading mechanism
oFormCreationParams.XmlData = oXMLDoc.xml
Use with creation params; preferred over LoadBatchActions due to greater flexibility/control
Creating Forms – Save, Load or Update using XML (Sample)

```vba
Dim oXMLDoc As New Xml.XmlDocument '…when using .NET's System.Xml
Dim oForm As SAPbouiCOM.Form
Dim xmlData As String
Dim m_sPathToFormXML As String = "c:\xml\xml_UpdateSample.xml"

'1) Save: get XML resource from Quotation form
  oForm = SBO_Application.Forms.GetForm("149", 1)
  If oForm Is Nothing Then Exit Sub

  xmlData = oForm.GetAsXML()

'2) Load or Update: load the xml file into the XML document object
  oXMLDoc.Load (m_sPathToFormXML)

  'upload the xml... (preferrably to update a form...)
  SBO_Application.LoadBatchActions (oXMLDoc.InnerXml)

  'eventually check for errors and warnings
  SBO_Application.GetLastBatchResults()
  oXMLDoc = Nothing
```

- New forms should not be created using the LoadBatchAction method of the Application object (see next slide for the preferred method)
- Existing forms can be modified using LoadBatchAction with “action” = “update” (see UI API help file or SDN Developer Area for SAP Business One for more)
- Add your changes to the XML and keep the XML e.g. in a resource file
- Since LoadBatchAction just takes an XML string you could of course just load the XML as a text file – or from the DB (as a string); using XML libraries facilitates handling and makes it possible to modify the XML before loading
Creating Forms – Loading Forms using XML
(Sample)

Dim oForm As SAPbouiCOM.Form
Dim creationPackage As SAPbouiCOM.FormCreationParams
Dim oXMLDoc As New Xml.XmlDocument ’…when using .NET’s System.Xml

’Create the FormCreationParams object
creationPackage = SBO_Application.CreateObject(_
SAPbouiCOM.BoCreatableObjectType.cot_FormCreationParams)

’Please note: These parameters override corresponding data in the XML
creationPackage.UniqueID = “MP_MyFormID”
creationPackage.FormType = “MP_MyFormType”
creationPackage.BorderStyle = SAPbouiCOM.BoFormBorderStyle.fbs_Fixed

’Just a sample for an XML string describing a form… same as used for LoadBatchActions
oXMLDoc.Load(“C:\XML\Sample.srf”) creationPackage.XmlData = oXMLDoc.InnerXml

’Add the form to the SBO application
oForm = SBO_Application.Forms.AddEx(creationPackage)

’Set the form visible (can be set in XML too)!
oForm.Visible = True
You should now be able to:
- Explain how to create new forms and items
- Use Screen Painter Add-On to create forms
- Save and load forms using XML
You are now ready for:
- Hands-on in an exercise about Screen Painter and XML handling features of the UI API...
## Item Events: Unit Overview Diagram

### The User Interface API

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UI API Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Add-On Basics</td>
</tr>
<tr>
<td>3</td>
<td>Creating Forms</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>5</td>
<td>Menus</td>
</tr>
<tr>
<td>6</td>
<td>Data Binding</td>
</tr>
<tr>
<td>7</td>
<td>Use UDO in Add-On</td>
</tr>
<tr>
<td>8</td>
<td>Additional Events</td>
</tr>
<tr>
<td>9</td>
<td>Additional Objects</td>
</tr>
<tr>
<td>10</td>
<td>UI API – Additional Information</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will be able to:
- Handle ItemEvents
- Use event filtering
- Manipulate SAP Business One forms
Form item types are the same as controls in Visual Basic forms from a user perspective - except the LinkedButton which is specific to SAP Business One

Technically, the SAP objects are unrelated to VB form controls

Examples of form item types are:

- Button
- CheckBox
- ComboBox
- EditText
- LinkedButton
- Grid
- Matrix (most tables in system forms)
- OptionBtn ("Radio Button")
- PictureBox
- StaticText
The Form.PaneLevel property is used with the Item.FromPane and Item.ToPane values to create multiple panes or “layers” within a form, in which different items are visible on different panes. Typically used with Folder tabs to display different items on different tabs.

Set FromPane and ToPane properties for each item:
- If both properties are set to 0, the item will be visible on all panes.
- Example: If item oEdit1.FromPane = 1 and oEdit1.ToPane = 3, then the item will be visible when oForm.PaneLevel is 1, 2, or 3.

Properties which are common to all items are directly available in the Item object:
- Top, Left, Width, Height properties
- Update method

Other members depend on item type (ComboBox, Matrix, etc). These are available through the Item’s "Specific" property:
- String property (EditText item)
- Selected property (ComboBox item)
- ValidValues property (ComboBox item)
- Columns property (Matrix item)
- Layout property (Matrix item)
The code snippets

- `oEdit = oItem.Specific
  oEdit.String = "Hello World"

and

- `oItem.Specific.String = "Hello World"

are equivalent.

In the latter case, however, IntelliSense will not automatically display EditText members, so it is easier to use a reference to the specific object (e.g. EditText) to work with.

Note:

- Setting the String property of the EditText item will (technically) cause a COM event to be fired to UI API. Changing many properties from add-on code results in a lot of calls through the UI API and may cause performance issues. It is recommended to set the value through DataSource. Refer to DataBinding with Datasources for details.
ItemEvent – General Remarks

- Occurs when a UI event takes place on a form OR any of its items (controls)
- Examples of Item Events: LostFocus, GotFocus, FormActivate, FormLoad, Click, ItemPressed, …
- ItemEvent handler (function):
  Private Sub SBO_Application_ItemEvent ( _
    ByVal FormUID As String, _
    ByRef pVal As ItemEvent, _
    ByRef BubbleEvent As Boolean _
  ) Handles SBO_Application.ItemEvent

- The data structure „pVal“ contains a large number of data providing details regarding the calling situation
- BubbleEvent specifies whether the event will continue to be processed by SAP Business One

- BubbleEvent
- BubbleEvent specifies whether the event will continue to be processed by SAP Business One
- Default value is True
- By setting BubbleEvent = False, you are canceling the event. This is similar to setting Cancel = True in a VB application.
- BubbleEvent is only valid when BeforeAction = True
**ItemEvent – Flow Of Control**

**SAP Business One Client**
- Form in SAP Business One application
- SAP Business One event handler

**Add-on**
- My event handler

**Flow**
- ItemEvent
- BeforeAction = True
- ActionSuccess = False / True
- BeforeAction = False
- return Control
The parameter BubbleEvent is available for the ItemEvent as well as for the MenuEvent.
This example adds a button to a Business Partner Master Data form when it loads.

Do not confuse UI API Item object with Items in general collections (such as UI API Items or Forms).

Note: Changes to SAP system forms occur only at runtime and are not persisted in any way. The method shown uses explicit low-level code – the alternative is to use XML batch actions.
FormDataEvent - Sample

The FormDataEvent occurs when the application performs the following actions on forms connected to business objects:

- Add, Update, Delete, Load (via browse, link button, or find) form data.

```
Private Sub SBO_Application_FormDataEvent(ByVal BOInfo As SAPbouiCOM.BusinessObjectInfo, ByVal BubbleEvent As Boolean) Handles SBO_Application.FormDataEvent
    If (BOInfo.BeforeAction = True) Then 'Before Event
        'Do something
    Else 'After event
        Dim oForm As SAPbouiCOM.Form = SBO_Application.Forms.Item(BOInfo.FormUID)
        Dim oBusinessObj As SAPbouiCOM.BusinessObject = oForm.BusinessObject
        Dim uid As String = oBusinessObj.Key

        If (BOInfo.Type = "2") Then
            Dim BP1 As SAPbobsCOM.BusinessPartners
            BP1 = oCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners)
            BP1.Browser.GetByKeys(BOInfo.ObjectKey)
            Dim cardCode As String = BP1.CardCode
        End If
    End If
End Sub
```

- Handling this event will make sure that your add-on is called whenever data are displayed or changed.
- Several types of lower-level events may cause a FormDataEvent. Add-on code is clearer when it handles a FormDataEvent instead of a mixture of ItemEvents and MenuEvents which underlie it.
Event Filtering – Motivation

A lot of form events are forwarded to add-ons, including:
- `et_ITEM_PRESSED` a button released/pressed
- `et_FORM_LOAD` SAP Business One application opened a form
- `et_KEY_DOWN` a key was pressed
- `et_GOT_FOCUS/et_LOST_FOCUS` an item got/lost focus
- `et_CLICK` “Mouse Up” on editable item

All menu click events are forwarded to add-ons...
- `et_MENU_CLICK` “Mouse Up” occurred on menu item (not a sub-menu!) in SAP Business One application
  This event must be included in a filter if an add-on is to handle MenuEvents

⇒ by default, all add-ons receive all events in the event handlers they implement
  -> this takes time even for events to which the add-on does not respond
⇒ filtering (capturing) only the events that need to be handled improves performance

- `et_ITEM_PRESSED` and `et_FORM_LOAD` are often used to add additional validation checks before saving a document or to manipulate a form before it’s shown (e.g. make some fields invisible depending on business logic)
- `et_KEY_DOWN` might be useful for a special kind of “help“ (when key „X“ is pressed some detail is shown)
- other form events are usually used less frequently
By default, the UI API receives all events triggered by the SAP Business One application.

Without event filtering, all events are sent to your add-on application. Your event handler is getting called each time an event is raised. This can result in poor performance overall.

If you use event filtering, only the selected events are sent to your add-on application. Significantly fewer COM calls will be made and performance improves.

AppEvents are not affected.

Note:

Once you define an EventFilter, add it to the EventFilters object and assign it to the Application object, your add-on will start to only receive events specified in the filters.

To continue to receive MenuEvents, don’t forget to include et_MENU_CLICK in the filter.
Event Filtering - Sample

The add-on will receive only the following events:
- et_ITEM_PRESSED for all forms
- Other forms:
  - Purchase Order - all events (et_ALL_EVENTS)
  - Sales Order - et_KEY_DOWN and et_ITEM_PRESSED

NOTE: To make sure that MenuEvents are sent to the add-on et_MENU_CLICK needs to be added to the event filter too!

- Event is filtered by event type and form type.
- The add-on notifies the list of required events through the SetFilter() method of the Application object
- The event list contains event types for:
  - form events, listing all form types for which they will be raised
  - menu click event
- The event list cannot contain:
  - AppEvents (aet_ShutDown...)
  - ProgressBarEvents
  - StatusBarEvents

- Note: Most UI API events are notified twice – before they take place in the user interface (BubbleEvent = True) and after they have taken place
Filtering Events: Code Example

1) create a new EventFilters object
   oFilters = New SAPboui.COM EventFilters

2) add an event type to the container
   (this method returns an EventFilter (<> EventFilters) object)
   oFilter = oFilters.Add(et_CLICK)

3) assign the form types on which the event should be processed
   oFilter.AddEx("139") 'Sales Order Form
   oFilter.AddEx("142") 'Purchase Order Form

... add a second event type to the container
   oFilter = oFilters.Add(et_KEY_DOWN)

... assign the form type on which this event should be processed
   oFilter.AddEx("139") 'Sales Order Form

4) set the event filters object to the application
   SBO_Application.SetFilter(oFilters)

- You can remove a particular form type from the filter by using RemoveEx(„FormType“)
- You can also remove all filters through a Reset() of the Filters collection
Want to find the “right” event? Use the Event Logger!

- EventLogger is part of the SAP Business One Development Environment (B1DE) toolset – and available on SDN... (see unit „Introduction“)
- Easily identify the events fired by the UI API depending on user actions
- Check the information given by SAP Business One for each event – including available event types (ItemEvent, MenuEvent, AppEvent etc).
You should now be able to:
- Handle ItemEvents
- Use event filtering
- Manipulate SAP Business One forms
ItemEvents, Event filtering, and more: Exercise

You are now ready for:
- Hands-on handling of ItemEvents etc. in an exercise…
## Menus: Unit Overview Diagram

### The User Interface API

<table>
<thead>
<tr>
<th>Topic 1: UI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Add-On Basics</td>
</tr>
<tr>
<td>Topic 3: Creating Forms</td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
</tbody>
</table>

### Topic 5: Menus

<table>
<thead>
<tr>
<th>Topic 6: Data Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
<tr>
<td>Topic 9: Additional Objects</td>
</tr>
<tr>
<td>Topic 10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will be able to:

- Add and remove menu items
- Describe menu event handling
The Menus object is a collection of MenuItem objects
- It holds all currently visible menu items
- You can add your own menus
- You can enable/disable/remove menus
MenuEvent provides notification of menu click events
- You can use it to open user forms or to perform other operations
- …or to capture (and eventually block) the opening of system forms at a very early stage
- Please note that clicks on toolbar buttons are represented as menu events as well
Dim oMenus As SAPbouiCOM.Menus
Dim oMenuItem As SAPbouiCOM.MenuItem
Dim oCreationPackage As SAPbouiCOM.MenuCreationParams

' Get the menus collection from the application
oMenus = SBO_Application.Menus

oCreationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectType.cot_MenuCreationParams)

' Point on the module sub menu
oMenuItem = SBO_Application.Menus.Item("43520")
oMenus = oMenuItem.SubMenus

' Set SubMenu values into the MenuCreationPackage object
oCreationPackage.Type = SAPbouiCOM.BoMenuType.mt_POPUP
oCreationPackage.UniqueID = "SM_VID"
oCreationPackage.String = "Video Store"
oCreationPackage.Image = sPath & "VID.bmp"
oCreationPackage.Position = 8 ' Some valid position; check-out what happens, if it is invalid.

Try ' If the menu already exists this code will fail
    oMenuItem = oMenus.AddEx(oCreationPackage) ' Add the SubMenu item
Catch err As Exception ' Error Handling
    SBO_Application.MessageBox(err.Message)
End Try
Note:

- If you are reusing a MenuCreationParams object for several menu items, set all properties, including Position, every time – AddEx() does not change/increase any properties implicitly.
You cannot add top-level menu items (i.e. at the same level as “File”, “Edit”, “Modules”, “Help”, …)

If you add a menu item with sub-menus to menu “Modules”, it will automatically appear in the “Main Menu” form.

If you link menus to a form, they will appear under the “Goto” top level menu.

Every menu item has its unique id

- You can export menu items to XML to find out the particular IDs.
- You can use “System information” to find it – just let the mouse pointer hover over the menu item.

Context or „right-click“ menus can be modified when handling the RightClickEvent (see later in this unit – or in the UI API help file).
Private Sub SBO_Application_MenuEvent (ByRef pVal As SAPbouiCOM.MenuEvent, ByRef BubbleEvent As Boolean)
Handles SBO_Application.MenuEvent

If pVal.BeforeAction Then
    SBO_Application.MessageBox("Menu item: " + pVal.MenuUID + " sent BEFORE SAP Business One processes it.", bmt_Long, True)
    '// to stop SAP Business One from processing this event
    '// unmark the following statement
    '// BubbleEvent = False
Else
    SBO_Application.MessageBox("Menu item: " + pVal.MenuUID + " sent AFTER SAP Business One processed it.", bmt_Long, True)
End If

End Sub
You should now be able to:
- Add and remove menu items
- Describe menu event handling
Menus: Exercise

You are now ready to:
- Add new menus in SAP Business One and
- Handle Menu Events in an exercise…
# Data Binding: Unit Overview Diagram

## The User Interface API

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: UI API Introduction</td>
</tr>
<tr>
<td>Topic 2: Add-On Basics</td>
</tr>
<tr>
<td>Topic 3: Creating Forms</td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>Topic 5: Menus</td>
</tr>
<tr>
<td><strong>Topic 6: Data Binding</strong></td>
</tr>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
<tr>
<td>Topic 9: Additional Objects</td>
</tr>
<tr>
<td>Topic 10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
Data Binding: Topic Objectives

At the conclusion of this topic, you will be able to:
- Bind data to form items
Data Binding: Characteristics and Motivation

- DataSources serve as **containers for data within a form** - they are not necessarily linked directly to the database

- DataSources **improve performance** because frequent manipulation of data values does not necessarily require frequent updates of the user interface

- **Some items (e.g. Matrix, Grid) should be bound** to a data source

- **Some items (e.g. Checkbox) have to be bound** to a data source – some items may not even be displayed unless they are bound to a data source
There are 3 types of data sources

**DBDataSource** – linked to a database table, represents tabular data (you can only use 1 table + only set conditions – no sorting etc.)

**UserDataSource** – acts as a container for data within the form, can be connected e.g. to an EditText or a column in a Matrix

**DataTable** – two methods:
- Populate with SQL statement (so that you can use joins, sorting etc.)
- **OR** (no mixing possible)
  - Define the Columns of the DataTable one-by-one and fill through code…

DataTables are mostly used in conjunction with Grid or ChooseFromList objects
Using data binding, you can easily add data to matrix columns. The SAP Business One Software Development Kit provides several objects that support data binding to form items.
To create a data-bound form:

- Define the form
- Define data sources within the form
- Link data sources to matrix columns or individual items/controls
- Populate data source values – this will display the data in the data bound items
The Form object contains a collection of DataSources which holds all data sources within the form.

- A DBDataSource object represents a database data source (i.e. a table in the SAP Business One scope of tables) attached to a form.

- A table can be attached only once to a form using method Add of the DBDataSources collection.

- User data sources can also be attached to a form using the Add method of the UserDataSources collection. For more information, see the documentation for the UserDataSources collection.

- A DataTable can be used to read data from any database/table or be used in the same way as a UserDataSource.

- In conjunction with a Grid item, the DataTable enables the display of tabular data with collapse/expand functionality.

```
'Add a DBDataSource to the form
oForm.DataSources.DBDataSources.Add("OUSR")
'Add a UserDataSource
oForm.DataSources.UserDataSources.Add("udsRemarks", dt_LONG_TEXT, 30)
'Add a DataTable
oForm.DataSources.DataTables.Add("MyDataTable")
```
Having added a data source to a form, then specify which form items to link to it.

For a simple item such as an EditText, the item’s Specific property contains the DataBound object.

Use its SetBound method to bind the item to a data source.

For matrices, data is binded column-by-column.
Dim oColumnDBS As SAPbouiCOM.Column
Dim oColumnUDS As SAPbouiCOM.Column
oMatrix = Form.Items.Item("Matrix1").Specific
oColumns = oMatrix.Columns

'DBDataSource: Binding a field / alias of the table to a column
oColumnDBS = oColumns.Item("UserName")
oColumnDBS.DataBind.SetBound (True, "OUSR", "U_NAME")

'UserDataSource: Bind a UserDataSource (UID) to a column
oColumnUDS = oColumns.Item("Remarks")
oColumnUDS.DataBind.SetBound (True, "", "udsRemarks")

'DataTable: Bind a DataTable object to a Grid
oGrid.DataTable = Form.DataSources.DataTables.Item("MyDataTable")
This code fragment will populate a matrix from table OUSR based on the data binding of individual matrix columns.

The Query method retrieves all data. Optionally, a Conditions argument can be specified to implement a WHERE clause.

The matrix can be populated row-by-row using the AddRow method or populated in one step with LoadFromDataSource. When some matrix columns are user data bound, LoadFromDataSource is only useful if all rows contain the same value for any user data bound column.

To reference a user data source and set its value:

```vbnet
Dim oUserDataSource As SAPboui.Com.DBDataSource
Dim oMatrix As SAPboui.Com.Matrix

' getting the data sources bound to the form
oDBDataSource = oForm.DataSources.DBDataSources.Item("OUSR")

' getting the matrix on the form
oMatrix = oForm.Items.Item("Matrix1").Specific

oMatrix.Clear()

' Querying the DB DataSource - i.e. load data from DB
oDBDataSource.Query()

' Adding the data to the matrix
oMatrix.LoadFromDataSource()
```
To populate your DataTable “manually”:

```vba
Dim oDataTable As SAPbouiCOM.DataTable

' getting the data sources bound to the form
oDataTable = oForm.DataSources.DataTables.Item("MyDataTable")

' Querying the DataTable
oDataTable.ExecuteQuery("Select CardCode, DocDate from OINV")

' Columns of the Grid will be added and populated automatically
```

- To populate your `DataTable` “manually”:
- Dim `oDataTable` as SAPbouiCOM.DataTable
- Dim `oCol` As SAPbouiCOM DataColumn
- ‘ Add the columns to the Grid manually
  - `oDataTable` = `oForm`.DataSources.DataTables.Item("MyDataTable")
  - `oCol` = `oDataTable`.Columns.Add("XX_Col0", SAPbouiCOM.BoFieldsType.ft_AlphaNumeric)
  - `oCol` = `oDataTable`.Columns.Add("XX_Col1", SAPbouiCOM.BoFieldsType.ft_AlphaNumeric)
  - `oCol` = `oDataTable`.Columns.Add("XX_Col2", SAPbouiCOM.BoFieldsType.ft_AlphaNumeric)
- ‘ Add a first row
  - `oDataTable`.Rows.Add()
- `oCol` = `oDataTable`.Columns.Item("XX_Col0")
- `oCol`.Cells.Item(0).Value = "MyVal0"
- `oCol` = `oDataTable`.Columns.Item("XX_Col1")
- `oCol`.Cells.Item(0).Value = "MyVal1"
- `oCol` = `oDataTable`.Columns.Item("XX_Col2")
- `oCol`.Cells.Item(0).Value = "MyVal2"
- `oGrid`.DataTable = `oForm`.DataSources.DataTables.Item("MyDataTable")
**Data Binding: DataSources on System Forms**

**IMPORTANT**

- DataSources are only populated with data already stored in the database
- Updates have to be committed to the database
- DataSources on system forms cannot be changed (there are plans to allow changing at least user-defined fields in version 9.0)

- ItemEvents such as `et_DATASOURCE_LOAD` and `et_MATRIX_LOAD` only occur for user forms, not system forms
Data Binding: DataSources on User Forms

- **Navigation**
  - When navigating between records, set a condition for the DBDataSource – or the DataTable

- **Values**
  - When you need to display values in a different format than stored in the database, use UserDataSource:
  - Run the query (e.g. via DI API or DBDataSource), format the data as required and then store the values in UserDataSources

- **Clearing form items**
  - Set the condition of DBDataSources so that the results are empty
  - Set UserDataSource values to ""
  - Set UI item strings directly to "" only as a last resort
You should now be able to:
- Bind data to form items
Data Binding: Exercise

You are now ready for:
- Hands-on data binding in an exercise…
Use UDO in Add-On: Unit Overview Diagram

The User Interface API

| Topic 1: UI API Introduction |
| Topic 2: Add-On Basics |
| Topic 3: Creating Forms |
| Topic 4: ItemEvents, Event Filtering (and more) |
| Topic 5: Menus |
| Topic 6: Data Binding |
| **Topic 7: Use UDO in Add-On** |
| Topic 8: Additional Events |
| Topic 9: Additional Objects |
| Topic 10: UI API – Additional Information |
At the conclusion of this topic you will know how to use UDO features in UDO forms:

- Connected UDO to form
- Default buttons
- Number series
How to use your new UDO within an Add-on?

**Connect Form to a UDO:**

```
creationPackage = SBO_Application.CreateObject
(SAPbouiCOM.BoCreatableObject_Type.cot_FormCreationParams)

creationPackage.FormUID = "MathExamsID"
creationPackage.Type = "SM_MathExam"

' Need to set the parameter with the object unique ID
creationPackage.**ObjectType** = "SM_MATHGRADES"

oForm = SBO_Application.Forms.AddEx(creationPackage)
```
## Default Buttons on UDO Form

<table>
<thead>
<tr>
<th>Service</th>
<th>UI form support</th>
</tr>
</thead>
</table>
| Add/Update/Find    | oItem = oForm.Items.Add("1", SAPbouiCOM.BoFormItemTypes.it_BUTTON) oButton = oItem.Specific  
Do not set the caption for this button |
|                    | (automatic event handling for the OK buttons)                                                                                                 |
| Cancel             | oItem = oForm.Items.Add("2", SAPbouiCOM.BoFormItemTypes.it_BUTTON) oButton = oItem.Specific  
Do not set the caption for this button |
<table>
<thead>
<tr>
<th>Service</th>
<th>UI form support</th>
</tr>
</thead>
</table>
| Manage Series | `create a combo box for the series relevant for this document type
oItem = oForm.Items.Add("SeriesName", BoFormItemTypes.it_COMBO_BOX)
oComboBox = oItem.Specific` |
|     | `fill the combo with relevant series
oComboBox.ValidValues.FillWithSeries(True, False, 0)
oComboBox.DataBind.SetBound(True, "@MATH", "Series")` |
|     | `edit text the hold the document number (related to the selected series)
oItem = oForm.Items.Add("DocNum", SAPbouiCOM.BoFormItemTypes.it_EDIT)
oEditText = oItem.Specific
oEditText.DataBind.SetBound(True, "@MATH", "DocNum")` |
|     | `***** later e.g. in the event handler *****************************
`get the “next serial number” from the selected series in add mode
strSeries = oComboBox.Selected.Value
lNum = oForm.BusinessObject.GetNextSerialNumber(CLng(strSeries))` |
|     | `set the “next serial number” it into the document number field
oEditText.String = CStr(lNum)"` |
You should now be able to use UDO features in UDO forms:

- Connected UDO to form
- Default buttons
- Number series
### The User Interface API

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: UI API Introduction</td>
</tr>
<tr>
<td>2: Add-On Basics</td>
</tr>
<tr>
<td>3: Creating Forms</td>
</tr>
<tr>
<td>4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>5: Menus</td>
</tr>
<tr>
<td>6: Data Binding</td>
</tr>
<tr>
<td>7: Use UDO in Add-On</td>
</tr>
<tr>
<td><strong>8: Additional Events</strong></td>
</tr>
<tr>
<td>9: Additional Objects</td>
</tr>
<tr>
<td>10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
At the conclusion of this topic you will know about further events in addition to the basic ones such as ItemEvent:

- ProgressBarEvent
- StatusBarEvent
- RightClickEvent
- ReportDataEvent
- PrintEvent
**ProgressBarEvent / StatusBarEvent**

**ProgressBarEvent**
Occurs when a progress bar (in the status bar) is created, stopped or released

Public Event ProgressBarEvent
    ( ByVal pVal As ProgressBarEvent, ByRef BubbleEvent As Boolean )

ProgressBarEvent - holds all the relevant information about the event – essentially the type:
- pbet_ProgressBarCreated
- pbet_ProgressBarStopped
- pbet_ProgressBarReleased

**StatusBarEvent**
Occurs when a message is displayed in SAP Business One’s status bar

Public Event StatusBarEvent ( ByVal Text As String, ByVal MessageType As BoStatusBarMessageType )
By default all menu entries from Edit, Data and Goto menus in the SAP Business One application are displayed in the context or right-click menu.

RightClickEvent is raised when the user right-clicks an item.

To add/remove menus to/from the context menu of an item:
- ...catch RightClickEvent ‘Before’ and
- ...add menus to Edit, Data, Goto menus in the SAP Business One application

In the ‘After’ event user should retrieve changes and/or remove menu changes that should only be temporary.

Code sample – Add menu:
```vbnet
Private Sub SBO_Application_RightClickEvent(ByVal contextMenuInfo As SAPbouiCOM.contextMenuInfo, ByVal BubbleEvent As Boolean) Handles SBO_Application.RightClickEvent
    If (contextMenuInfo.BeforeAction = True) Then
        Dim oCreationPackage As SAPbouiCOM.MenuCreationParams = SBO_Application.CreateObject(BoCreatableObjectType.cot_MenuCreationParams)
        oCreationPackage.Type = SAPbouiCOM.BoMenuType.mt_STRING
        oCreationPackage.UniqueID = "MyMenu1"
        oCreationPackage.String = "My Menu1"
        oCreationPackage.Enabled = True
        Dim oMenuItem As SAPbouiCOM.MenuItem = SBO_Application.Menus.Item("1280")
        Dim oMenus As SAPbouiCOM.Menus = oMenuItem.SubMenus
        oMenus.AddEx(oCreationPackage)
    End If
End Sub
```

Code sample – Remove menu:
```vbnet
Private Sub SBO_Application_RightClickEvent(ByVal contextMenuInfo As SAPbouiCOM.contextMenuInfo, ByVal BubbleEvent As Boolean) Handles SBO_Application.RightClickEvent
    If (contextMenuInfo.BeforeAction = True) Then
        Dim menuItem As SAPbouiCOM.MenuItem = SBO_Application.Menus.Item("768")
        Dim menuItem1 As SAPbouiCOM.MenuItem = menuItem.SubMenus.Item("773")
        menuItem1.Enabled = False
    End If
End Sub
```

Code sample – Cleanup:
```vbnet
Private Sub SBO_Application_RightClickEvent(ByVal contextMenuInfo As SAPbouiCOM.contextMenuInfo, ByVal BubbleEvent As Boolean) Handles SBO_Application.RightClickEvent
    If (contextMenuInfo.BeforeAction = False) Then
        Dim menuItem As SAPbouiCOM.MenuItem = SBO_Application.Menus.Item("768")
        Dim menuItem1 As SAPbouiCOM.MenuItem = menuItem.SubMenus.Item("773")
        menuItem1.Enabled = True
    End If
End Sub
```
Right Click Menu – Details

RightClickEvent - Fires ‘Before’ and ‘After’ events

RightClickEvent   (ByRef contextMenuInfo As ContextMenuInfo, ByRef BubbleEvent As Boolean)

ContextMenuInfo – holds all parameters for the event

- String FormUID – form unique id
- BoEventTypes EventType – event type
- String ItemUID – item unique id
- String ColUID – column unique id. Default value is -1
- String Row – row number. Default value is -1
- Boolean BeforeAction – indicates if the event is ‘Before’ or ‘After’
- Boolean ActionSuccess – relevant only for ‘After’ event, indicates whether B1 application action succeeded
ReportDataEvent / PrintEvent

ReportDataEvent (and subsequently PrintEvent) occur when an end-user performs one of the following actions:
- Clicking on Print or Print Preview icons
- Sending documents to print using “Document Printing” option
- A document is sent to print by the “Document Generation Wizard”

**ReportDataEvent**

```
ReportDataEvent (ByRef eventInfo As ReportDataInfo,
                 ByRef BubbleEvent As Boolean)
```

In “BeforeAction = True” for this event the add-on has to signal that it wants to get report data in XML format. It does so by calling RegisterForReport():

```
eventInfo.RegisterForReport (True)
```

**PrintEvent**
You should now be able to describe:
- ProgressBarEvent
- StatusbarEvent
- RightClickEvent
- ReportDataEvent
- PrintEvent
### Additional Objects: Unit Overview Diagram

**The User Interface API**

<table>
<thead>
<tr>
<th>Topic 1: UI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Add-On Basics</td>
</tr>
<tr>
<td>Topic 3: Creating Forms</td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>Topic 5: Menus</td>
</tr>
<tr>
<td>Topic 6: Data Binding</td>
</tr>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
</tbody>
</table>

**Topic 9: Additional Objects**

<table>
<thead>
<tr>
<th>Topic 9: Additional Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 10: UI API – Additional Information</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will know about additional objects in addition to the basic building blocks such as Form and Item:

- Grid
- ChooseFromList
- FormSettings
The Grid is planned to replace the longer-established Matrix as a tabular control.

Grid is a view of a DataTable:
- The Grid is responsible for the visual settings
- The DataTable is responsible for the data behind the user interface
- Grid & DataTable synchronise automatically
  - Data changes flow from Grid to DataTable and vice versa
  - Meta-Data / Structural changes are synchronized from the DataTable to its managed Grids

The Grid enables expand/collapse
DataTable is a type of DataSource

Object Model

- Grid
  - Rows (GridRows)
  - DataTable
  - RowHeaders
  - Columns (GridColumn)
  - GridColumns

Object Model

- DataSource
  - DBDataSources
  - UserDataSources
  - DataTables
# DataTable versus Grid

<table>
<thead>
<tr>
<th>DataTable</th>
<th>Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural actions</strong></td>
<td><strong>Structural actions</strong></td>
</tr>
<tr>
<td>Execute query (+data)</td>
<td>Control of column display types &amp; properties</td>
</tr>
<tr>
<td>Load from XML (+data)</td>
<td>Expose collapsible view mechanism of</td>
</tr>
<tr>
<td></td>
<td>existing data</td>
</tr>
<tr>
<td>Add columns</td>
<td>Row selection methods</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Actions</strong></td>
<td><strong>Data Actions</strong></td>
</tr>
<tr>
<td>Add rows</td>
<td>Set cell value by display type</td>
</tr>
<tr>
<td>Set cell value by data type</td>
<td></td>
</tr>
</tbody>
</table>
ChooseFromList (CFL) is the ability to use the built-in lookup functionality form from a trigger item.

- The CFL form displays a list of objects (of the same type) as a result of a simple query.
- New functionality will enable developers to apply filters to CFL objects which were defined for system forms.
- No need to develop lookup forms from scratch.
**ChooseFromList - Details**

- Use CreationParams mechanism to create a ChooseFromList (CFL) object
- Set a condition (same object as for DBDataSources)
- Connect a CFL-capable item to a CFL (EditText, EditTextColumn, Button):

### EditText, EditTextColumn

- **Property ChooseFromListUID as String (read-write)**
  - Sets the item to be the trigger item for the CFL UID
  - (using a wrong ID will cause an exception)

- **Property ChooseFromListAlias as String**
  - Alias – Field in database that will be compared in query (using a wrong alias will cause an exception)
  - Remark: Set the alias after setting CFL UID.

### Button

- **Property ChooseFromListUID as String (read-write)**
  - Sets the item to be the trigger item for the CFL UID
  - (using a wrong ID will cause an exception)
How to handle the Event et_CHOOSE_FROM_LIST

ChooseFromList Event “inherits” from ItemEvent

=> It comes as an ItemEvent, but the structure passed to the event handler is different!

■ BeforeAction = True

Sent before the ChooseFromList form is opened

If BubbleEvent = FALSE the CFL form will not open – as you would expect

■ BeforeAction = False

Sent after the user made his choice (select) or pressed “Cancel” in the CFL form

Properties:

■ ChooseFromListUID as String (read-only)

   Note: For a CFL that was opened from “Find” – the UID of the CFL will be -1

■ SelectedObjects as DataTable (read-only)

   The result is valid/available during the after event only
   All manipulation of the data must be completed during the event

Code sample available in SDK samples

XML support for ChooseFromList:

XML

<Form >
  <ChooseFromLists>
    <ChooseFromList UniqueID="1" ObjectType="2" MultiSelection="0" IsSystem="1">"
      <conditions>
        <condition alias="CardType" bracket_close_num="0" bracket_open_num="0"
          compare_fields="0" compared_field_alias="" cond_end_val="" cond_value="C"
          operation="1" relationship="0" use_result="0"/>
      
      <conditions>
    </ChooseFromList>
  </ChooseFromLists>

  <Items>
    <Item uid="5" ...type="EditText">
      <Specific ChooseFromListUID="" ChooseFromListAlias="CardCode"/>
    </Specific>
  </Item>

  .......... <Column ChooseFromListUID="MyCFL" ChooseFromListAlias="kk" > (EditText/linkbutton)
                            </Column>

  <Items>

4-106
ChooseFromList – Limitations and Restrictions

- Possible trigger item types are Button, EditText and EditTextColumn (use of other types will throw an exception)
- The user CFL is opened the same way as a system CFL:
  - EditText/Edit Column /Link Column – by {TAB}
  - Button – By press
- The table of the CFL object is the header table. Therefore the condition is applied to the header table.
- There will be no automatic copy between the resulted DataTable to any DataSource. Explicit code must be written to do this.
- A user-CFL form will always open, even if there is only one match or no match at all
- Find mode – Executing find also opens a CFL form but there is no trigger item
- CFL form – cancel on ‘new’ button will not raise an “after event”
- There’s a 1:1 relationship between the trigger item and the CFL
- System CFL limitations
  - We can’t see the system conditions on CFL. We can only see the Add-On Conditions.
  - System CFL is not editable - the only change that is allowed is adding conditions
  - You cannot change the trigger item of system CFL
- Changing the trigger item of a user CFL
  - When new item is bound to a CFL the old one is overridden
  - When replacing CFL – the old connection of both is overridden
The setting button enables users to configure the way a matrix in a form will be displayed.
Every column can be toggled as visible and/or active.
User Form Settings – Overview (continued)

Form Settings / Form Preferences (see DI API)

- Saving preferences
  - Form settings are updated when a form is closed
  - The preferences are held in memory (application cache) until the application is closed or another database selected
  - When the application is closed the updated preferences are saved to the database (table CPRF)

- Loading preferences
  - Application caches form preferences as it logs in to the company database
  - When a form is loaded, it loads and applies the settings from cache
  - User Forms – Preferences are applied automatically only when layout is loaded from XML

Form

Property Settings as FormSettings (read-only)

Only on user forms, exception is raised on system forms

- Beware of “unexpected” behavior when multiple forms of the same types are open simultaneously or when the user is logged in multiple sessions
Form Settings Default Behavior

The default behavior expects a form with a “grid”:
- The grid is set as the default grid for the settings
- The Settings menu will be enabled for the form
- The row format and expand line will be enabled for the grid

Disabling this functionality:
- The Settings functionality is on by default
- To disable it from an add-on, disable the form settings menu item (ID 5890)
- To disable the row format and expand line, set EnableRowFormat = False
You now know how to use:

- Grid
- ChooseFromList
- FormSettings
## The User Interface API

<table>
<thead>
<tr>
<th>Topic 1: UI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Add-On Basics</td>
</tr>
<tr>
<td>Topic 3: Creating Forms</td>
</tr>
<tr>
<td>Topic 4: ItemEvents, Event Filtering (and more)</td>
</tr>
<tr>
<td>Topic 5: Menus</td>
</tr>
<tr>
<td>Topic 6: Data Binding</td>
</tr>
<tr>
<td>Topic 7: Use UDO in Add-On</td>
</tr>
<tr>
<td>Topic 8: Additional Events</td>
</tr>
<tr>
<td>Topic 9: Additional Objects</td>
</tr>
</tbody>
</table>

**Topic 10: UI API – Additional Information**
Go to SDN to find People, Information and Tools

Developer Area – Includes: Links to tools, articles etc

Free registration to Discussion Forums

FAQ

B1 DB Browser

B1 Test Composer

B1 Form Checker

B1 Code Generator
Check “what’s going on” using B1TE’s .NETProfiler

- B1TE (“SAP Business One Test Environment”) is available on the SDN (see unit “Introduction”)
- Traces calls to SDK APIs and any other .NET objects
- Marks deprecated SDK API calls
- Only available for add-ons using Microsoft .NET (uses Profiling API of MS .NET)
Check your Forms using B1TE’s Form Checker

- Checks a form against the B1 programming guidelines and UI standards & guidelines
- Lists all the possible issues encountered in a form itself
- Can check XML layout definitions as well as any forms shown in the application
Add-On Testing – Using SAP Business One Test Composer (B1TC)

- Simple way to test add-ons
- Records, replays and checks values
- Can perform batch tests and selected tests in a batch
User Interface API – Use Cases

User Interface API is most often used to:

- Achieve a „seamless“ integration of additional functionality with SAP Business One (usually requested by customers), including
  - linking into SAP Business One standard processes
  - adding custom GUI elements into SAP Business One standard forms
  - adding custom forms with user-defined data links

- Manipulate SAP Business One standard functionality (when standard options do not apply to the customer's or industry processes, including
  - hiding SAP Business One GUI elements
  - blocking SAP Business One events

=> Changes to standard functionality must be documented!
You should now be able to:

- Explain what the User Interface API is
- Explain how to establish a connection to a running SAP Business One application
- Work with existing SAP Business One forms
- Create forms and integrate them into SAP Business One GUI
- Add menu entries
- Explain how the API interacts with the SAP Business One client
Exercises

Unit: User Interface API

Topic: Basics

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

- Connect to a SAP Business One Application
- Display a MessageBox in SAP Business One
- Use Single Sign-On and the “Multi Add-On” feature
- React to AppEvents

You want to use the SAP Business One User Interface API for actively manipulating Process flow. As a first step you have to connect to the Application actually running.

1-1 Implement a connection to a running SAP Business One application.

1-1-1 Create a new Visual Basic project

1-1-2 Define the variables you need for a connection to a running SAP Business One application.

You will need at least two variables, one for the SboGuiApi object and one for the Application object

1-1-3 Connect to the SAP Business One SboGuiApi and get a handle to the running application.

1-2 Display a MessageBox within SAP Business One.

There is a Method of the Application object to display message boxes within SAP Business One

1-2-1 The method to display a MessageBox has several optional parameters. Check them out.

The lecture will continue after you have implemented this.; the remaining pieces of this exercise will be covered in the next steps.
1-3 Use the Single-Sign-On feature (and/or the Multiple Add-On feature) to connect to DI API as well.

1-4 Define the AppEvent handler – and implement the handling of these events (which are mandatory to be handled).

To define Event Handlers in Microsoft Visual Studio .NET please check the content on the drop-down comboboxes – which are displayed just above the source code…
Exercises

Unit: User Interface API
Topic: Creating Forms

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

- Create a form within SAP Business One

You want to create a form which is displayed in the SAP Business One main window.

2-1 Create a new form within SAP Business One. The form should contain the following items:

- Input field for DVD Name (will be linked to a Choose from list)
- Input field for DVD Aisle
- Input field for DVD Section
- Input field for DVD Rented
- Input field for Rented To
- OK button
- Cancel button.
- Rent DVD button.

This (the screenshot below) is the final goal, but you will only get data when you have gone through the “Databinding” lesson as well; in this exercise we will only focus on the form’s layout…
2-2 Enhance your program so that the form will be saved as an XML file.

2-3 Change your program. The form should now be loaded from the XML file you have created in the last step. Display the form in the SAP Business One window.

2-4 Use the tools from the B1TE toolset (essentially Form Checker) to check whether you have designed your form(s) according to some important UI guidelines…

Some helpful data for designing Forms

(See ScreenDesignGuidelines.pdf for more information!)

<table>
<thead>
<tr>
<th>Form</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>413px</td>
</tr>
<tr>
<td>Width</td>
<td>557px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controls common</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance to left edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to right edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to top edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to bottom edge</td>
<td>5px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Button</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>19px</td>
</tr>
<tr>
<td>Width</td>
<td>65px</td>
</tr>
<tr>
<td>Spacing</td>
<td>5px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label Field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>14px</td>
</tr>
<tr>
<td>Width</td>
<td>Depends on text</td>
</tr>
<tr>
<td>Horizontal spacing</td>
<td>&gt;=12px (ungrouped)</td>
</tr>
<tr>
<td>Vertical spacing</td>
<td>1px (grouped)</td>
</tr>
<tr>
<td></td>
<td>&gt;=3px (ungrouped)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>14px</td>
</tr>
<tr>
<td>Width</td>
<td>Enough to show complete field value</td>
</tr>
<tr>
<td>Horizontal Spacing</td>
<td>&gt;=12px (ungrouped)</td>
</tr>
<tr>
<td>Vertical spacing</td>
<td>1px (grouped)</td>
</tr>
<tr>
<td></td>
<td>&gt;=3px (ungrouped)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Help</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical distance to input field</td>
<td>1px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Form width minus 2 times 5px to the left and right edge</td>
</tr>
<tr>
<td>Number of rows</td>
<td>&lt;=7 rows, add scrollbars if more necessary</td>
</tr>
</tbody>
</table>
**Exercises**

Unit: User Interface API  
Topic: Additional Event Handling

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

- Handle SAP Business One events.

You want to actively influence the SAP Business One dialogues. Therefore you need to handle SAP Business One events.

You only want to receive the events you are interested in.

3-1 Catch FormLoad event for Order form

Look for the FormLoad event into the possible events thrown by SAP Business One (Application, Menu or Item events)

Have a look into the Event parameters to find out whether the FormLoad event is coming from the Order form or from another form

3-2 Display the message "Caught Order FormLoad Event" when the FormLoad event for the Order form arrives (use the Application.Message method).

3-3 Catch the click event on the “Rent DVD” button you've added in the previous exercise. Again display a message once the event is caught.

3-4 Create a filter to only receive the events we are interested in.

To set a filter on the UI events you should use the EventFilter object.  
Don’t forget to assign the EventFilter to the application you are connected to.
Exercises

Unit: User Interface API
Topic: New Menu entries in SAP Business One window

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

- Add a menu entry in the SAP Business One main menu

The form of your Add-On solution should be displayed when the user chooses your new menu entry.

For that purpose you will create a new entry and handle the menu event.

4-1 Add a menu entry to the "Modules" menu called **DVD Store**. Those are the menu entries which are also displayed in the Main Menu. Add a new sub-menu to that menu entry called **DVD Availability**.

Ideally you use the specification for the menu of the “Course Project” example!

4-2 Handle the menu event: When you choose the menu entry, your form should be displayed.

4-3 Add another menu only visible when your form is open. This menu should appear under the GoTo menu.

Use the **Menu** collection property into the Form object to link the menus to your form.
Exercises

Unit: User Interface API
Topic: Data Binding

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

- Bind data to fields of a form within the SAP Business One window.

You have created a new form that is displayed within SAP Business One. Now you want the system to display data on that form.

5-1 Declare a DBDataSource and UserDataSource object. Link it to the form you created in the Creating Forms exercise.

5-2 Bind the form's items with the corresponding data from the User Defined table created in the DI exercises (TB1_DVD).

Use the Method "DataBind.SetBound" on each item to assign it
- the corresponding table and field name it is associated to
for the DBDataSources

5-3 Get the data from the data sources and display them in the corresponding fields on your form. Note you will first need to read the value selected by the user from the Choose from List (Item Event) and then fill all other fields accordingly.

Use the Query method into the DBDataSource object to filter the information you want to show.

5-4 Test your application.
Solutions

Unit: User Interface API

Topic: Basics

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

- Connect to a SAP Business One Application
- Send a Message in SAP Business One
- Use Single Sign-On and the “Multi Add-On” feature
- React to AppEvents

You want to use the SAP Business One User Interface API for actively manipulating Process flow. As a first step you have to connect to the Application actually running.

1-1 Implement a connection to a running SAP Business One application.

1-1-1 Create a new Visual Studio project for a windowless add-on application and add a reference to the SAP Business One DI API COM library and UI API COM library.

1-1-2 Define the variables you need for a connection to a running SAP Business One application.

```
Private WithEvents SBO_Application As SAPbouiCOM.Application
Dim SboGuiApi As SAPbouiCOM.SboGuiApi
Dim sConnectionString As String
```

1-1-3 Connect to the SAP Business One SboGuiApi and get a handle to the running application.

```
SboGuiApi = New SAPbouiCOM.SboGuiApi
sConnectionString = Environment.GetCommandLineArgs.GetValue(1)
SboGuiApi.Connect(sConnectionString)
SBO_Application = SboGuiApi.GetApplication()
```

1-2 Display a MessageBox within SAP Business One.

1-2-1 The method to display a MessageBox has several optional parameters. Check them out.

```
SBO_Application.MessageBox("Connected", 1, "Continue", "Cancel")
```
1-3 Use the Single-Sign-On feature (and/or the Multiple Add-On feature) to connect to DI API as well.

```vbnet
Private oCompany As SAPbobsCOM.Company
oCompany = New SAPbobsCOM.Company
oCompany = SBO_Application.Company.GetDICompany()
```

1-4 Define the AppEvent handler – and implement the handling of these events (which are mandatory to be handled).

```
Dim connectionstring As String
SBOUILAPI = New SAPbobsCOM.SBOUILAPI

// by following the steps specified above, the following statements should be sufficient for either development or
```

Solutions can be found in the SDK Help Center documentation and SDK samples (in the SDK Folder – see Appendix “SDK Installations” for more information).

COM UI / VB .NET / 01.HelloWorld
COM UI / VB .NET / 02.CatchingEvents
COM UI DI / VB .NET / Hello World
At the conclusion of this exercise, you will be able to:

- Create a form within SAP Business One

You want to create a form which is displayed in the SAP Business One main window.

2-1 Create a new form within SAP Business One. The form should contain the following items:

Some example code:

**Form Creation:**

```plaintext
creationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectObject.cot_FormCreationParams)

creationPackage.UniqueID = "TB1_DVDAvailability"
creationPackage.FormType = "TB1_DVDAvailability"
creationPackage.ObjectType = "TB1_DVDAvail"  'link form to your UDO

oForm = SBO_Application.Forms.AddEx(creationPackage)

oForm.Title = "DVD Availability Check"
oForm.Left = 336
oForm.ClientWidth = 280
oForm.Top = 44
oForm.ClientHeight = 200
```

---

**Unit: User Interface API**

**Topic: Creating Forms**

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

- Create a form within SAP Business One

You want to create a form which is displayed in the SAP Business One main window.

2-1 Create a new form within SAP Business One. The form should contain the following items:

Some example code:

**Form Creation:**

```plaintext
creationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectObject.cot_FormCreationParams)

creationPackage.UniqueID = "TB1_DVDAvailability"
creationPackage.FormType = "TB1_DVDAvailability"
creationPackage.ObjectType = "TB1_DVDAvail"  'link form to your UDO

oForm = SBO_Application.Forms.AddEx(creationPackage)

oForm.Title = "DVD Availability Check"
oForm.Left = 336
oForm.ClientWidth = 280
oForm.Top = 44
oForm.ClientHeight = 200
```
**Button creation:**

```vbnet
oItem = oForm.Items.Add("RentDVD", SAPbouiCOM.BoFormItemTypes.it_BUTTON)
oItem.Left = 200
oItem.Width = 65
oItem.Top = 170
oItem.Height = 19

oButton = oItem.Specific
oButton.Caption = "Rent DVD"
```

**Choose from List**

```vbnet
Dim oCFLs As SAPbouiCOM.ChooseFromListCollection
oCFLs = oForm.ChooseFromLists

Dim oCFL As SAPbouiCOM.ChooseFromList
Dim oCFLCreationParams As SAPbouiCOM.ChooseFromListCreationParams
oCFLCreationParams = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectType.cot_ChooseFromListCreationParams)
oCFLCreationParams.ObjectType = "TB1_DVDAvail" 'Note – this is the Code you gave in the wizard when you registgered the UDO  for TB1_DVD in the UDO exercises
oCFLCreationParams.UniqueID = "DVDCFL"
oCFL = oCFLs.Add(oCFLCreationParams)
```

**EditText Creation**

```vbnet
oItem = oForm.Items.Add("DVDNameT", SAPbouiCOM.BoFormItemTypes.it_EDIT)
oItem.Left = 90
oItem.Width = 163
oItem.Top = 25
oItem.Height = 14
oItem.LinkTo = "DVDNameL" 'link it to the associated Static

oEditText = oItem.Specific
oEditText.DataBind.SetBound(True, "", "DVDName")
oEditText.ChooseFromListUID = "DVDCFL"
```

2-2 Enhance your program so that the form will be saved as an XML file.

Firstly add a reference to Microsoft XML – this references .NET’s System.Xml library

```vbnet
oXmlDoc = New Xml.XmlDocument
sXmlString = oForm.GetAsXML
oXmlDoc.LoadXml(sXmlString)
oXmlDoc.Save("File location " & "DVDAvailability.xml")
```
2-3 Change your program. The form should now be loaded from the XML file you have created in the last step. Display the form in the SAP Business One window.

This code uses MSXML library – so this is just for demonstration. It’s preferable to use LoadBatchActions for updates and Forms.AddEx to load forms.

Private Sub LoadFromXML(ByVal Filename As String)
    Dim oXMLDoc As MSXML2.DOMDocument
    oXMLDoc = New MSXML2.DOMDocument

    oXMLDoc.load("File Location\" & Filename)
    SBO_Application.LoadBatchActions((oXMLDoc.xml)
End Sub

2-4 Use the tools from the B1TE toolset (essentially Form Checker) to check whether you have designed your form(s) according to some important UI guidelines…

Similar solution can be found in the SDK UI samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM UI/03.SimpleForm, 06.MatrixAndDataSources and 04.WorkingWithXML
Some helpful data for designing Forms

<table>
<thead>
<tr>
<th>Form</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>413px</td>
</tr>
<tr>
<td>Width</td>
<td>557px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controls common</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance to left edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to right edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to top edge</td>
<td>5px</td>
</tr>
<tr>
<td>Distance to bottom edge</td>
<td>5px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Button</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>19px</td>
</tr>
<tr>
<td>Width</td>
<td>65px</td>
</tr>
<tr>
<td>Spacing</td>
<td>5px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label Field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>14px</td>
</tr>
<tr>
<td>Width</td>
<td>Depends on text</td>
</tr>
<tr>
<td>Horizontal spacing</td>
<td>&gt;=12px (ungrouped)</td>
</tr>
</tbody>
</table>
| Vertical spacing | 1px (grouped)  
|                 | >=3px (ungrouped) |

<table>
<thead>
<tr>
<th>Input Field</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>14px</td>
</tr>
<tr>
<td>Width</td>
<td>Enough to show complete field value</td>
</tr>
<tr>
<td>Horizontal Spacing</td>
<td>&gt;=12px (ungrouped)</td>
</tr>
</tbody>
</table>
| Vertical spacing | 1px (grouped)  
|                 | >=3px (ungrouped) |

<table>
<thead>
<tr>
<th>Field Help</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical distance to input field</td>
<td>1px</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix Objects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Form width minus 2 times 5px to the left and right edge</td>
</tr>
<tr>
<td>Number of rows</td>
<td>&lt;=7 rows, add scrollbars if more necessary</td>
</tr>
</tbody>
</table>

Please note that more exhaustive information is available in the “User Interface: Standards and Guidelines” in the “SAP Business One Topic Search” on SAP Servicemarketplace at http://service.sap.com/smb/sbo/resources (October 2007).
Solutions

Unit: User Interface API

Topic: Additional Event Handling

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

- Handle SAP Business One events.

You want to actively influence the SAP Business One dialogues. Therefore you need to handle SAP Business One events.

You only want to receive the events you are interested in.

3-1 Catch FormLoad event for Order form

```
If pVal.FormType = "139" And pVal.EventType = SAPbouiCOM.BoEventTypes.et_FORM_LOAD And pVal.BeforeAction = False Then
End If
```

3-2 Display the message "Caught Order FormLoad Event" when the FormLoad event for the Order form arrives (use the Application.Message method).

```
SBO_Application.MessageBox("Caught Order FormLoad Event")
```

3-3 Catch the click event on the “Rent DVD” button you’ve added in the previous exercise. Again display a message once the event is caught.

```
If FormUID = "TB1_DVDAvailability" And pVal.ItemUID = "RentDVD" And pVal.EventType = SAPbouiCOM.BoEventTypes.et_ITEM_PRESSED Then
   SBO_Application.MessageBox("Caught click on Rent DVD button")
End If
```
Create a filter to only receive the events we are interested in.

```vbnet
Public oFilters As SAPbouiCOM.EventFilters
Public oFilter As SAPbouiCOM.EventFilter
oFilters = New SAPbouiCOM.EventFilters()
oFilter = oFilters.Add(SAPbouiCOM.BoEventTypes.et_ITEM_PRESSED)
oFilter.AddEx(“139”) 'Orders Form
oFilter.AddEx(“TB1_DVDAvailability”)
SBO_Application.SetFilter(oFilters)
```

A solution implementing events catching can be found in the SDK UI samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM UI/02.CatchingEvents.
Unit: User Interface API

Topic: New Menu entries in SAP Business One window

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

- Add a menu entry in the SAP Business One main menu

The form of your Add-On solution should be displayed when the user chooses your new menu entry.

For that purpose you will create a new entry and handle the menu event.

4-1 Add a menu entry to the "Modules" menu called **DVD Store**. Those are the menu entries which are also displayed in the Main Menu. Add a new sub-menu to that menu entry called **DVD Availability**.

```vba
Dim oMenus As SAPbouiCOM.Menus
Dim oMenuItem As SAPbouiCOM.MenuItem

oMenus = SBO_Application.Menus

Dim oCreationPackage As SAPbouiCOM.MenuCreationParams
oCreationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectType.cot_MenuCreationParams)
oCreationPackage.Type = SAPbouiCOM.BoMenuType.mt_POPUP
oCreationPackage.UniqueID = "TB1_DVDStore"
oCreationPackage.String = "DVD Store"
oCreationPackage.Enabled = True
oCreationPackage.Image = sPath & "dvd.bmp"
oMenus = oMenuItem.SubMenus

Try
    oMenus.AddEx(oCreationPackage)
End Try

Dim sPath As String
sPath = Application.StartupPath
sPath = sPath.Remove(sPath.Length - 3, 3)
oCreationPackage.Type = SAPbouiCOM.BoMenuType.mt_STRING
oCreationPackage.UniqueID = "TB1_DVDStore"
oCreationPackage.String = "DVD Store"
oCreationPackage.Enabled = True
oCreationPackage.Image = sPath & "dvd.bmp"
oMenus = oMenuItem.SubMenus
```

4-137
4-2 Handle the menu event: When you choose the menu entry, your form should be displayed.

```vbscript
If pVal.MenuUID = "TB1_Avail" And pVal.BeforeAction = False Then
    LoadFromXML("DVDAvailability.xml")
End If
```

```vbscript
Private Sub LoadFromXML(ByVal Filename As String)
    Dim oXMLDoc As MSXML2.DOMDocument
    Try
        oXMLDoc = New MSXML2.DOMDocument
        oXMLDoc.load("File location" & Filename)
        SBO_Application.LoadBatchActions(oXMLDoc.xml)
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

4-3 Add another menu only visible when your form is open. This menu should appear under the GoTo menu.

```vbscript
Dim oCreationPackage As SAPbouiCOM.MenuCreationParams
    oCreationPackage = SBO_Application.CreateObject(SAPbouiCOM.BoCreatableObjectType.cot_MenuCreatio
Dim oMenuForm As SAPbouiCOM.Form
    oCreationPackage.Type = SAPbouiCOM.BoMenuType.mt_STRING
    oCreationPackage.UniqueID = "TB1_TestMenu"
    oCreationPackage.String = "Test Menu"
    oMenuForm.Menu.AddEx(oCreationPackage)
```

A similar solution can be found in the SDK UI samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM UI/05.AddingMenuItems
Solutions

Unit: User Interface API
Topic: Data Binding

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

- Bind data to fields of a form within the SAP Business One window.

You have created a new form that is displayed within SAP Business One. Now you want the system to display data on that form

5-1 Declare a DBDataSource and UserDataSource object. Link it to the form you created in the Creating Forms exercise.

```csharp
oForm.DataSources.DBDataSources.Add("@TB1_DVD")
oForm.DataSources.UserDataSources.Add("DVDName", SAPbouiCOM.BoDataType.dtSHORT_TEXT)
```

Note the TB1_DVD table should be defined as Master Data UDO table and registered as a UDO

5-2 Bind the form's items with the corresponding data from the User Defined table created in the DI exercises (TB1_DVD).

Firstly for each Edit Text field bind the field to it’s corresponding column in the User defined table e.g.

```csharp
oEditText.DataBind.SetBound(True, "@TB1_DVDAVAIL", "U_AISLE")
```
5-3 Get the data from the data sources and display them in the corresponding fields on your form. Note you will first need to read the value selected by the user from the Choose from List (Item Event) and then fill all other fields accordingly.

```vba
If pVal.EventTypes = SAPbouiCOM.BoEventTypes.et_CHOOSE_FROM_LIST Then
    Dim oCFLEvent As SAPbouiCOM.ChooseFromListEvent
    Dim oCFL As SAPbouiCOM.ChooseFromList
    Dim CFLID As String
    Dim oForm As SAPbouiCOM.Form

    oCFLEvent = pVal
    CFLID = oCFLEvent.ChooseFromListUID
    oForm = SBO_Application.Forms.Item(FormUID)
    oCFL = oForm.ChooseFromLists.Item(CFLID)
    If oCFLEvent.BeforeAction = False Then
        Dim oDataTable As SAPbouiCOM.DataTable
        oDataTable = oCFLEvent.SelectedObjects
        Dim val As String
        Try
            val = oDataTable.GetValue(1, 0)
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
        If pVal.ItemUID = "DVDNameT" Then
            oForm.DataSources.UserDataSources.Item("DVDNameT").ValueEx = val
            oDBDataSource = oForm.DataSources.DBDataSources.Item("@TB1_DVD")
            oDBDataSource.Query()
        End If
    End If
End If
```

5-4 Test your application.

A similar solution can be found in the SDK UI samples (in the SDK Folder – see Appendix “SDK Installations” for more information), COM UI/03.SimpleForm and 06.MatrixAndDataSources
Add-On Packaging, Administration & Licensing

Contents:
- Add-On Administration
- Packaging
  - Creating a Package
- Licensing
  - Add-On Identifiers
Creating an Add-On: Unit Objectives

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

- List what you need to do to create an Add-On package
- Perform the steps that are necessary to register an Add-On
- Describe the SAP Business One license mechanism
- Explain the different Add-On Identifier types and their usage
1 Course Overview
2 SDK Introduction
3 The Data Interface API (short look on JCo + DI Server)
4 User-Defined Objects (UDO)
5 The User Interface API
6 Packaging, Add-On Administration and Licensing
Creating an Add-On: Business Example

You have developed an industry-specific solution for SAP Business One. Now you want to deliver this solution to your customers.
# Packaging Introduction: Unit Overview

## Diagram

**Add-On Packaging, Administration & Licensing**

<table>
<thead>
<tr>
<th>Topic 1: Packaging Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: License Concept</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will be able to:

- Explain how to package your solution
- Describe what to include in the Add-On package
- How to get the registration data file
- Describe how your solution will be registered in SAP Business One
In your Add-On documentation make sure that – beyond the documentation of any User-Defined Tables, Fields etc – document each event and form in the SAP Business One application that you handle or manipulate.

Include the following information:

- Expected situation (prerequisite)
- Action that is performed (change of data)
- Condition for a break in event chain (i.e. when you set parameter BubbleEvent = False)
- Situation that can be expected by possible successors (other Add-Ons) handling the same event

- Add-On components (including registration data file, setup etc.)
- Add-On Installer must be 1 (one!) executable file
- Provide Documentation
- Describe User-defined fields and tables
- Describe the User-defined objects you define in your Add-On (if applicable)
- List where you modify SAP Business One standard functionality (if appropriate):
  - E.g. list which Modules – or forms / items – you hide etc
- List where you interfere in the control flow of SAP Business One standard functionality (if appropriate) – i.e. list any Event you capture which originates from SAP Business One standard functionality:
  - recommended for all Add-Ons; mandatory for Add-Ons to be certified
  - especially where you might set BubbleEvent to False (Mandatory for all Add-Ons)
  - This is important since at the customer site various Add-Ons from various vendors might get engaged!
After you’ve finished to develop the add-on for your customer the most important thing is to deliver and install it correctly on your clients station.

Before installing the Add-on on the customer’s station make sure that:

- Customers station complies with the prerequisites for running SAP Business One. (the prerequisites are detailed available system setup documentation)
- All the relevant components are installed on your client station

We recommend you to create an installation package with one of the available tools in the market (for example: Microsoft package and deployment, Installed Shield) in this way you can be sure that all the relevant components (dll, OCXs, etc…) will be packed in the package – or to use the installation wizard which is included in the B1DE (SAP Business One Development Environment) toolset – which is available through SDN.

Do you remember (just because it happened often that partners disregarded that fact…)?

An Add-On can connect to UI API in two different modes:

- Development Mode
  - Using the predefined connection string (don’t confuse it with the Add-On Identifier!!!) within your code
  - 0030002C0030002C00530041005000420044005F00440061007400650076002C0050004C006F006D0056004900490056

- Customer Mode (Runtime mode)
  - Connect using the connection string that comes as the commandline parameter…

---

**Overview – Add-On deployment step-by-step**

<table>
<thead>
<tr>
<th>Develop / Create Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile the Add-On program</td>
</tr>
<tr>
<td>Create installation program</td>
</tr>
<tr>
<td>Create registration data file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register / Install / Administrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register (and install) the Add-On</td>
</tr>
<tr>
<td>Set Company Preferences</td>
</tr>
<tr>
<td>Set User Preferences</td>
</tr>
<tr>
<td>Assign Add-On Licenses to Users (if appropriate)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run or Stop the Add-On</td>
</tr>
</tbody>
</table>
Add-On Installer – AddOnInstallAPI.dll

The installation is always initiated by the SAP Business One client. The installer must handle a command line parameter:

"RecommendedPathForAddOn" | "PathForAddOnInstallAPI.dll"

You must use the AddOnInstallAPI.dll in your installer!

**Mandatory:**

- **EndInstallEx(String, Boolean)**
  Call this function at the end of the installation process. The boolean parameter allows to indicate success (= True) or failure (= False). The function returns an integer value; 0 signals success.

- **EndUnInstall(String, Boolean)**
  Call this function at the end of the uninstallation process. The boolean parameter allows to indicate success (= True) or failure (= False). The function returns an integer value; 0 signals success.

**Optional:**

- **SetAddOnFolder(path)**
  If you modified the default installation path as provided by SAP Business One

- **RestartNeeded()**
  Call this function if the setup program restarts the PC (see SDK documentation for more details!)

- See UI API helpfile for more details

- Please note the „EndInstall“ is deprecated!
The registration data file allows the application to identify your add-on and run it automatically when the application is launched.

In order to generate the license data file

1. Run the _AddOnRegDataGen.exe_ file, typically located at:

   C:\Program Files\SAP Manage\SAP Business One SDK\Tools\AddOnRegDataGen
2. Enter your partner information  
3. Enter the add-on information  
4. Enter the data of the add-on installer  
5. Choose Generate File to create the registration data file

- Please note that during installation exceeding the „Estimated Install Time“ will cause a message box to pop up. In the message box the user can confirm – or deny – the successful installation of the Add-On – at a later time.
AddOnRegDataGen.exe is batch capable.

Calling convention (commandline parameters):

```
AddOnRegDataGen.exe <xml info> <InstallerVersion>
<Installer> <Uninstaller> <Add-On-Exe>
```

**Sample:**

```
AddOnRegDataGen.exe MyAddOn.xml 1.0 setup.exe setup.exe
MyAddOn.exe
```

The `<xml info>`:

```
<AddOnInfo partnernmsp="ABC" contdata="my cont data"
addonname="My Add-On" addongroup="M" esttime="300"
instparams="" uncmdarg="" partnername="My Comp" />
```
Add-On Administration – Introduction

The Add-On Administration tool is designed to help administrators deploy and manage add-on applications on end-users workstations.

IMPORTANT:

The Add-On Administration tool and the current installation mechanism have been introduced in SAP Business One Version 2004. You might encounter older versions on the customer site. Check-out the appropriate information e.g. on SAP Service Marketplace /education.

With the Add-On Administrator you can:
- Register Add-Ons
- Set Company preferences
- Set User Preferences
- Remove Add-Ons
- Monitor Add-Ons
Add-On Administration – Overview

- Add-On Registration Process registers the Add-On applications in the SBO-Common DB on the SAP Business One server
  - Will be done before the first installation of an Add-On on a client machine.
  - The System Administrator registers the Add-On using Add-On Administration; this triggers the import of the installation package and the ARD file into the SBO-Common DB (table SARI).
  - Add-On installation can be started in one step with registration; otherwise it will be started at the next logon.

- Add-On Upgrade will be done by repeating the Add-On Registration Process.
  - Please note that the new Add-On version must be greater than the installed version (e.g. “1.1” instead of “1.0”).
  - In a first step the upgrade process will start the „uninstall“!
  - Add-On upgrade can be started in one step with registration; otherwise it will be started at the next logon.

- Add-On Administration is only available for users with Superuser privileges for the company.
Add-On Administration – Register installation package

In SAP Business One go to: Administration-> Add-On-> Add-On Administration

Choose Register Add-On

Registration Data File: Choose the data file of the Add-On.

Installation Package: Choose the installation file of the Add-On.

Check Assign to Current Company, if you like to assign the Add-On to the current company.

Check Activate for Company, if you like to start the installation on the PC you currently use.

Choose OK to close the window and register the Add-On

Once the Add-On is registered, it appears in the Available Add-On list on the Add-On Administration window.

Package has been copied into SBO-Common database (table SARI)...
Setting Company Preferences

The Add-On Administration tool lets you set different company-wide preferences for each company-assigned add-on.

To set these preferences, you must assign the add-on to the company (if you haven't done so already during the registration process) by moving the add-on from the Available Add-On list to the Company Assigned Add-On list using the icons.

The company preferences include:

- **Start-up Group** - Assigning a Start-up Group controls the start-up behavior and deployment of add-ons for all users connecting to a company.
- **Mandatory** - Add-on is needed to fulfill requirements of the customer specification; Add-On will be started automatically.
- **Automatic** - Add-on is started automatically by the SAP Business One application.
- **Manual** - Add-on is not started automatically by the SAP Business One application.

This setting can be changed per user – except for “Mandatory”.

- **Force install** - Forces the SAP Business One application to try again to install an Add-On that failed to install each time the end-user logs on to the company.
- **Event-receiving order** - This order is determined by the order (from top to bottom) in the Company Assigned Add-On list.
- **Active** – An Add-On can be temporarily deactivated through this setting.

See slide…
Add-On Administration – Send notification to Users

Client Install – A user can install new add-ons without logging again to Business one.

Administrator registers new add-ons and sends notification to users by clicking on the button on the Add-on Administration form.

![Add-On Administration Form]

You have new add-ons to install

<table>
<thead>
<tr>
<th>Message Header</th>
<th>Message Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>You have new add-ons to install</td>
<td>To install, go to Administration &gt; Add-Ons &gt; Add-On Manager &gt; Add-Ons Installer Tab</td>
</tr>
</tbody>
</table>

Send Notification  Cancel
Add-On Administration – Add-On Admin

Remarks

- Active flag overrides all other settings for the AddOn in a company:

  If the Add-On is marked as not active (the check box is not checked) the Add-On could be updated – without being installed and executed immediately.

- Extended log for Add-On Installation:
  - Add Windows environment variable AAdminLog
  - Values 0 (no log) … 3 (every message will be logged)
  - Logfile can be found in <Temp> folder of the Windows user (e.g. C:\Documents and Settings\<user>\Local Settings\Temp)
Add-On Administration – Add-On Manager

In SAP Business One go to: Administration → Add-On → Add-On Manager

Monitor Add-Ons in SAP Business One Client (current user).

Displays a list of Add-ons the user is allowed to run.

Displays Current Add-on Status: Connected, Disconnected, Failed

Notifies through Popup-Message in case Add-On failed.

Ability to Start Add-On manually within the SAP Business One application

Displays only the relevant information for current user

- In case an Add-On terminates the user will be informed about that fact including options to continue working or logoff from the current company.
Creating an installer using the B1DE – Toolset

The B1DE Package contains 2 installer wizards that you can use to create Add-on installers easily:

- SAP Installer Wizard – to generate the setup code to install and register an add-on with B1
- VB B1 Installer Wizard – to generate the .NET setup code to install and register an add-on with B1 (potentially better suitable for more sophisticated installation routines)

- In contrast to using B1DE to implement add-ons – no B1DE DLL files have to be shipped together with the installer.
- You can use the B1DE installer wizards without using B1DE for your add-on project!
You should now be able to:

- Explain how to package your solution
- Describe what to include in the Add-On package
- How to get the registration data file
- Describe how your solution will be registered in SAP Business One
Packaging Introduction: Exercise

Check-out the sample installation program in the SDK Folder (see Appendix “SDK Installations” for details about the SDK Folder)…

- Also try to install the Add-Ons you implemented in the exercises before and / or the Video Library course project you’ll develop later…
Additional Information: Certification Process

- This process is mandatory for Solution Partners (SPs / ISVs) and optional for Sale & Software Partners (SSPs)
- It includes specification of various technical information – above all what of and how the SDK’s interfaces are used
- Partners also have to describe test cases that characterize the solution.
- Certificators will approve documentation and test cases for the certification session – or request additional information – or more details.
- In the certification session data provided will be discussed and test cases will be checked.
# License Concept: Unit Overview Diagram

Add-On Packaging, Administration & Licensing

<table>
<thead>
<tr>
<th>Topic 1: Packaging Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: License Concept</td>
</tr>
</tbody>
</table>
At the conclusion of this topic, you will be able to:
- Describe the license concept for the SAP Business One Software Development Kit and Add-ons
- Describe the meaning of
  - License service and license file
  - Add-on Identifier
  - License mode
  - License Key Name (string)
- Order a license file
Your company has built a solution (Add-on) for SAP Business One. Now you want to add license checks for named users to your solution. To do so, you use the license concept of the SAP Business One Software Development Kit.
License Concept – Motivation

- Ensure Return on Investment (ROI)
- Restrict usage of an Add-On solution
- Re-use license mechanism of SAP Business One
  - No need for own license check programming
  - No need to set up a license infrastructure

Please note: Licenses for registered Add-Ons will be provided without further approval so far (Sept. 2007).

After a click on “License Overview” at the SAP Channel Partner Portal Quick Link
http://service.sap.com/licensekey a „License Report“ listing requested licenses for your registered Add-On solutions is displayed...
The license service is part of the SAP Business One server tools and can be installed on a central machine that can support multiple SAP Business One systems and Company Databases.

From version 2005 on, it is a CORBA service instead of COM service as of version 2004.

- You need to set port (default = 30000) + PC name (not IP address!) – in 2004 you could also use the IP address; no port could be specified (the port used was owned by DCOM).
- 2005 technology solves the domain problem with the DCOM service as of 2004.

When you connect to SAP Business One the system receives from the license service the modules the user is licensed. Whenever a form is clicked to be opened, it checks if the form is part of the license package the user has.

One named user can access multiple systems with just one license.

The license service does not need to be installed on a separate machine. It can of course also be installed on the SAP Business One database server.

The partner solution (= Add-on) identifies itself via SDK.

The license check for partner solutions is done via SAP Business One client resp. SDK DI API.

Two different SAP Business One database server (e.g. test system and productive system) can use one license service.

To simplify the handling of licenses it’s recommended to use one central license service for the whole SAP Business One system landscape.
Licensing Infrastructure – Technical Components

License Service / License Manager
- Part of „SAP Business One Server Tools“
- Can be installed on any computer
- Calculates a „Hardware Key“
- Provides logging capabilities to detect license-related issues
- Allows to set the port number it listens to
  - Actually the port that is specified is the port number of the „Naming Service“ that handles the initial connect to license service…
  - License service itself listens to that port number + 1
- Checks whether a session is still alive – to release concurrent user type licenses if necessary
- Responds to license checks triggered from any SAP Business One component
  - SDK components: Connect to component

Please note: “Named User” type license checks are performed per user code + database name (disregarding DB server + client PC)
  - It is only allowed to log on to SAP Business One once per user code / DB name!

- „Hardware Key“ available in the SAP Business One „About…“ screen and the Properties of the License Service
License File
- Specific for a particular License Service
- Generated through SAP Service Marketplace on request
- Includes licenses for:
  - All purchased SAP Business One components
  - Includes SAP components that are available for free
  - …and Add-On solutions
- Can be uploaded through License Service – or through the SAP Business One client application
- Issued per „localization“; customers who upgrade from previous versions will receive a „Global“ license that still allows to use any localization.

B1Upf.xml
- Keeps information regarding licenses assigned to specific user codes
- …is keeps this information independently from Company Database and even SAP Business One “Server”
Add-On Licensing – Step-by-step

ISV / Solution Partner
- Register Add-On solution
  - Currently use message to SAP Support for component SBO-SDK-AA
- Receive License Key Name
- Generate Add-On Identifier
- Use Add-On Identifier in source code to trigger check for specific Add-On Solution license

VAR / Sales & Service Partner / Customer
- Order SAP Licenses on the Channel Partner Portal http://service.sap.com/smb/sbo/order
- “Request” License Key from SAP (includes Add-On licenses)
- Install Add-On Solution
- Assign Licenses to users
Add-On Licensing – Terms

License Key Name
- Technical String returned by SAP upon registration of the Add-On Solution
- Starts with „BASIS“ followed by a 10-digit number
- Identical to SWPRODUCTNAME in the license file + extension (extension related to chosen DB type); e.g.:
  SWPRODUCTNAME=BASIS1234567890_MSS

Add-On Identifier
- Identifier that is used in Add-On code
  - „Add-On Identifier Generator“ in SAP Business One generates “Solution”, “Implementation” or “Development” Add-On Identifiers
    - “Solution” Add-On Identifier is generated from the License Key Name, allows to use DI API and UI API (Add-On License to be assigned to the user)
    - “Implementation” Add-On Identifier allows to use UI API only (concurrent user license)
    - “Development” Add-On Identifier allows to use DI API and UI API as well, but requires “SDK Development” License (concurrent user license)
- Use only the first 15 characters of the License Key Name (e.g. BASIS1234567890) to generate a “Solution” Add-On Identifier
Add-On Licensing – Remarks

ISV / Solution Partner
- Register Add-On solution
  - Registration during the License Key Request as “Partner Solution not listed” (just type in a name) will result in the solution being available for your direct customers only.
  - “SDK Development” License is prerequisite to register a Solution (it is checked whether this license has been ordered for any installation on the partner/customer number).
- An Add-On that uses SDK’s UI API only can run on “SDK Implementation” License.
  - Please note: Add-On Identifier is specific for the “System”; it has to be regenerated for other “Systems”.

VAR / Sales & Service Partner / Customer
- Can register “proprietary” Add-On solutions as well – “SDK Development” License is a prerequisite.
Add-On Licensing – License Key Request

- Use the Hardware Key supplied by License Manager or in the „About…“ dialog in SAP Business One when creating the „System“
- Choose from SAP licenses purchased (purchased licenses can be distributed across multiple “Systems” within an “Installation“)
- Choose Add-On Solution licenses as agreed with the SSP
- An email with an attached license file will be sent to the address entered for the „System“

Process for the License Key Request:
- Go to http://service.sap.com/licensekey
- Select the Installation Number a license file should be requested for. The respective systems for this Installation Number will be displayed
- To modify an existing license choose the respective System, change data and request a new license file.
- Go to “Request New System” Link to request an new license file for a new license landscape
- Fill in data and choose “Next Step”

In this screen the licenses of the different SAP Business One components and Certified and Uncertified Partner Solutions can be selected and will be included in the license file.

Customer specific solutions are shown in a personalized list for the partner only. If a customer runs a customer specific solution, the partner has to order the license file (with the same transaction). The partner can register his/her customer-specific solution via his/her license request form. His/Her solution is then shown in his/her personalized license request form and can be selected for a license file for the customer.

Also expiration dates can be set to give partners the possibility to send out demo or test licenses for their solutions.
Add-On Licensing – License Administration

The License Administration form allows:
- Administrators to maintain licenses and
- Grant users access to SAP Business One modules and Add-On solutions
- View content of license file
- Import a new license file to the license service
- Lock any users from the SAP Business One system
Add-On Licensing – License Administration

- SAP Business One and Add-On license can be maintained and controlled through the License Administration Form -> Administration -> License -> License Administration.
- Configurations can be maintained only for the Company Database the administration is currently logged on to.
- Indirect Access user is a valid SAP Business One license type, not authorized to any functionality inside the SAP Business One GUI application.
- No limitation on the number of add-ons assigned to one user
- Add-on licenses can only be assigned to users with a valid SAP Business One license type
- Registered Add-ons are displayed under External Licenses

Please note:
⇒ To use B1i(SN) two (free) licenses have to be assigned to the (technical) user „B1i“ in SAP Business One:
  - License type „B1i“
  - License type „B1iINDIRECT_MSS“
In general the Add-On Identifier String must be passed to the AddOnIdentifier property before calling the Connect() method of an API.

Sample code UI API

```vbnet
Dim b1GuiApi as New SAPbouiCOM.SboGuiApi
b1GuiApi.AddonIdentifier = "4CC5B8A4E0213A68489E38CB4052855EE8678 CD237F64D1C11C52706A541BD245D5E6E4050AE9B919FBE0FAB44F9"
b1GuiApi.Connect(sConnectionString)
```

Sample code DI API (for usage without UI API)

```vbnet
m_cmp = New SAPbobsCOM.Company
m_cmp.AddonIdentifier = "4CC5B8A4E0213A68489E38CB4052855EE8678 CD237F64D1C11C52706A541BD245D5E6E4050AE9B919FBE0FAB44F9"
lret = m_cmp.Connect()
```

- The Add-On Identifier String needs to be assigned to the Add-On Identifier Property before calling the connect method in the APIs
- Connections should be re-used to avoid wasting licenses for the same user.
- Add-on solutions using the UI and DI API should set the Add-On Identifier only in the UI API and first connect to the UI API and then to the DI API.
  - Another connection through the DI API would use up another license
- If the Add-On is assigned to the „Mandatory“ start group, a user that has not been assigned a license for this Add-On cannot logon to the particular company.
**Add-On Licensing – Technical details**

**Please note:**
- Add-On solutions using both, UI API and DI API in conjunction with the “single-sign on” feature have to leave the `AddOnIdentifier` property of DI's company object empty!
- When using the “Multi Add-on” feature to get the DI connection through UI API – the `Connect()` method won’t be called anyway.
- DI Server performs a license check when it starts.
- DI Server has a CPU-based license model!

**Please note further:**
- UI API has a functionality to check the License Status of a particular form for the logged on user:
  ```csharp
  Application.Company.GetFormLicenseStatus(…)
  ```
Regarding „Historical Licenses“:
- In the license file you will still find entries for „Implementation License“ and „Compatibility License“.
- „Compatibility License“ has been kept to support non-registered Add-ons technically.
- „Implementation License“ may not make much sense in this context. It has been kept for backward compatibility reasons though.

Please note (again):
To use UI API or DI API the user must have an SAP license assigned in addition (Indirect Access, Limited or Professional User) – no matter which SDK License type should be used!

<table>
<thead>
<tr>
<th>Licenses vs. Components</th>
<th>License Type</th>
<th>UI API</th>
<th>DI API</th>
<th>DI Server</th>
<th>Screen Painter</th>
<th>SAP Add-Ons</th>
<th>Namespace and Add-On registration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK Development</td>
<td>Conc.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>&lt;(ISV) Solution License&gt;</td>
<td>Named</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DI Server</td>
<td>CPU</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAP Add-Ons (free)</td>
<td>Named</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SDK Tools (free)</td>
<td>Named</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Historical Licenses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDK Implementation (free)</td>
<td>Conc.</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Compatibility License (free)</td>
<td>Conc.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The Implementation Mode is meant to be used during the implementation and development of small implementation Add-Ons
- Allows to create and run small implementation Add-Ons in a specific customer environment without applying for an Add-On License Key Name.
- Add-Ons with implementation identifier strings run only in the environment (license server) the identifier was created in.

The Development Mode is targeted to be used during the development phase of Add-On solutions (Please note: Concurrent user mode applies)
- A development license for the SAP Business One SDK must be available

The Solution Mode will be used running Add-On solutions at the customer site
- This mode was created to check valid licensing for partner Add-On solutions for SAP Business One
- The logged on user must have been assigned a (named user) license for this Add-On.
- The installation of the SDK runtime version is a prerequisite for Add-ons using DI API or Java Connector, but there’s no additional license check for the SDK in this license mode

The Compatibility Mode is available to support “old” Add-Ons that have been developed before release 2004 and do not use the Add-On Identifier string
- “Older” Add-Ons still run with SAP Business One release 2004 to ensure compatibility
- Add-Ons that do not set the “AddonIdentifier” property are assumed to be “old” Add-Ons.

several modes for different Add-ons are possible in one SAP Business One system landscape / can run with the same SAP Business One application
License Concept: Topic Summary

You should now be able to:

- Describe the license concept for the SAP Business One Software Development Kit and Add-ons
- Describe the meaning of
  - License service and license file
  - Add-on Identifier
  - License mode
  - License Key name (string)
- Order a license file
You should now be able to:
- List what you need to do to create an Add-On package
- Perform the steps that are necessary to register an Add-On
- Describe the SAP Business One license mechanism
- Explain the different Add-On Identifier types and their usage
Exercises

Unit: Add-On Packaging

Topic: Basics

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

- Write a simple VB .NET installer program.

VB .NET has capabilities to implement such an installer.

1-1 You can create your own installer or use the B1 Simple Installer or B1 Professional Installer from the SDN Development Tools
Exercises

Unit: Licensing

Topic: License mechanism for Add-Ons

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

- Use the licensing mechanism

2-1 Use Add-On Identifier generator to:

- Create a Development identifier
- Create an Implementation identifier
- Solution Identifier (need BASIS license from SAP)
- Note the differences

2-2 Use the Identifier in your code (use the property AddonIdentifier from the DI Company object or from the UI SboGuiApi object) and check out when it fails.
Solutions

Unit: Add-On Packaging

Topic: Basics

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

- Write a simple VB .NET installer program.

A solution can be found in the SDK UI samples (in the SDK Folder – see Appendix “SDK Installations” for more information),

Or from the SDN:
http://www.sdn.sap.com/irj/sdn/index?rid=/webcontent/uuid/a175fb62-0c01-0010-a8b5-fa58a13b1cf7#section21
At the conclusion of this exercise, you will be able to:

- Use the licensing mechanism

*There is no solution other than documented in the unit / the exercise.*
Appendix 1 - Tools

Contents:

Available on SDN
- Add-On test tools
- Add-On development tools
B1 SDK Tools – What?

Set of development and testing tools helping partners to develop and test their add-ons.

Given as free source code in SDN:

http://www.sdn.sap.com/irj/SDN/businessone

Tools offered:
- Development Environment
- Event Logger
- DI LogsReader
- DI Event Service
- COM License Bridge
- Test Environment
- Test Composer
B1 SDK Tools – How to download?

SAP BUSINESS ONE SDK TOOLS

DEVOTIONAL TOOLS

© SAP Business One Development Environment Tools

SAP Business One provides several APIs for building solutions. The SAP Business One Development Environment (BOE) is a tool that makes it even easier to use these APIs and speeds up the development and packaging of add-ons based on these APIs.

EL SERVICE TOOL

SAP Business One DI Event Service runs on top of the existing SAP Business One SDK interfaces and provides notification of events relating to SAP Business One DI objects through a listener-based interface.

Learn (PDF 152 KB) How to define and implement DI service. Together with this article, you can download a setup and an implementation sample. The development environment is close to .NET, for both CLI and VB NET programming languages.
Packages B1 SDK coding solutions best practices by providing wizards for code generation and helpful tools for development of add-ons.

- Based on B1 SDK
- Integrated with Microsoft Visual Studio .NET 2005 and 2008: the most used development environment for B1 solutions
- Comes with a set of documented guidelines to:
  - ensure correct usage of APIs
  - avoid the repetitive development (connection, forms and menus creation,…)
  - help partners to concentrate on the business side
  - ensure compatibility
  - add-on inter-working
  - etc
B1DE – Tools proposed

B1 Code Generator Wizard
- a set of Microsoft Visual Studio .NET wizards and add-ins
  - to generate .NET B1 solutions: VB.NET and C#

B1 Simple Installer Wizard
- a Microsoft Visual Studio .NET wizard
  - to generate the setup code to install and register an add-on with B1

B1 Professional Installer Wizard
- a Microsoft Visual Studio .NET wizard
  - to generate the .NET setup code to install and register an add-on with B1

B1 UDO Form Generator
- a Windows tool (also integrated with B1 Code Generator Wizard)
  - to generate an XML form starting from an UDO

B1 DB Browser
- a Windows tool (also integrated with B1 Code Generator Wizard)
  - to visualize the current status of a SAP Business One database in terms of the tables, columns, types, default values, database constraints and links
  - to visualize the changes in the database between two B1 versions
B1DE – B1 Code Generator Wizard

Generates your add-on code and data managing:

- UI API and DI API connections
- metadata objects creation
  - User Defined Tables
  - User Defined Fields
  - User Defined Objects
- events management
  - listener-based interface
  - events registration
  - events filtering
- menu actions
  - creation, deletion, update
  - attach a form to a menu
- form generation
B1DE - Installer Wizards

B1 add-on installing requirements:
- a unique setup executable
- an ARD file

Two wizards available
- Simple installer
  - generates a simple .NET Application Project
  - no coding required at all

- Professional installer
  - generates a .NET Setup and Deployment project
  - requires .NET Setup and Deployment projects basic knowledge
B1DE - DbBrowser

Visualizes current status of a B1 database
Offers the possibility to navigate between linked/related tables.
Shows information about changes between B1 versions
### Event Logger

#### Motivation
- Easily identify the events fired by the UI API depending on the user actions
- Observe the information given by B1 for each event.

<table>
<thead>
<tr>
<th>#</th>
<th>Time</th>
<th>Event</th>
<th>Event Type</th>
<th>Before</th>
<th>Success</th>
<th>Event Type</th>
<th>ForeTop</th>
<th>FormCount</th>
<th>FormUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>10:51:56.5264</td>
<td>Item Event</td>
<td>oe_FORM_ACTIVATE</td>
<td>False</td>
<td>True</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>10:51:57.4236</td>
<td>Item Event</td>
<td>oe_CLICK</td>
<td>True</td>
<td>False</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>10:51:57.4933</td>
<td>Item Event</td>
<td>oe_CLICK</td>
<td>False</td>
<td>True</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>10:51:57.5137</td>
<td>Item Event</td>
<td>oe_ITEM_PRESSED</td>
<td>True</td>
<td>False</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>10:51:57.5322</td>
<td>Item Event</td>
<td>oe_ITEM_PRESSED</td>
<td>False</td>
<td>True</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>10:51:57.6402</td>
<td>Item Event</td>
<td>oe_CLICK</td>
<td>True</td>
<td>False</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>10:51:57.8540</td>
<td>Menu Event</td>
<td>oe_MENU_CLICK</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>10:51:57.8862</td>
<td>Menu Event</td>
<td>oe_MENU_CLICK</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>10:51:57.9742</td>
<td>Item Event</td>
<td>oe_CLICK</td>
<td>False</td>
<td>True</td>
<td>169</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This tool provides a clear view of the XML file logs that can be produced by DI API. You can then analyze all DI API calls with detailed information, like interface and command name, elapsed time, input and output types and values.
Motivation
- UI API FormData events only alerts on user actions done on the B1 GUI, no alerts given for other add-ons action
- Avoid conflicts between different add-ons using the SBO_SP_TransactionNotification

Solution proposed
- Listener-based interface for data event notification
- Easy to use high-level interface integrated with the SDK
- Samples provided
DIEventService - Architecture

- Partner add-on
- B1DIEventService
- Connect
- AddListener
- RemoveListener
- Disconnect
- Call listeners
- .NET Remoting
- MSMQ
- Client
- Server
- B1DIEventServer
- DB
- SBO_SP_TransactionNotification
- B1DIEventSender

.NET Remoting

MSMQ
// Create an instance of the listener service
evtService = new B1DIEventService(oCompany);
            evtService.Connect(ConnectionLost_Listener)

// Add a listener method per each group: objType + transaction Type
            evtService.addListener(SAPbobsCOM.BoObjectTypes.oItems.ToString(),
                B1DIEventTransactionTypes.Add.ToString(),
                AddItems_Listener)

// Add a listener method per each group: objType + transaction Type
            evtService.addListener(SAPbobsCOM.BoObjectTypes.oOrders.ToString(),
                B1DIEventTransactionTypes.Add.ToString(),
                AddOrders_Listener)

            Listeners method declaration

            // AddItems Delegate implementation in the add-ons side
            public void AddItems_Listener(B1DIEventService.B1DIEventArgs
                eventInfo)
            {
                ...
            }

            // AddOrders Delegate implementation in the add-ons side
            public void AddOrders_Listener(B1DIEventService.B1DIEventArgs
                eventInfo)
            {
                ...
            }

            Disconnection

            // Remove a listener
            evtService.removeListener(SAPbobsCOM.BoObjectTypes.oItems.ToString(),
                B1DIEventTransactionTypes.Add.ToString())

            // Disconnect the service
            evtService.disconnect()
B1 Test Environment (B1TE)

Set of profiling tools for SAP B1 SDK add-ons

- Do not require the source code or a development environment
- Used by SAP during solution certification phase
- Scenarios:
  - Analyze add-ons compliance with SDK
  - Troubleshoot run-time issues
  - Check compatibility breakages

Tools included:

- B1 DB Browser
- B1 DB Profiler
- B1 .NET Profiler
- B1 Form Checker
- B1 Bubble Checker
- MSSQL Profiler
**B1TE - DbBrowser**

Visualizes current status of a B1 database

Offers the possibility to navigate between linked/related tables.

Shows information about changes between B1 versions
B1TE - DbProfiler

Keeps track of all changes in a B1 database carried out by a correct execution of a DI API call (based on SBO_SP_TransactionNotification stored procedure).

DOES NOT: Track incorrect accesses – as for instance accessing and modifying a B1 company DB through ODBC or direct SQL statements.
B1TE - .NETProfiler

Traces calls to SDK APIs and any other .NET objects
Marks deprecated SDK API calls
Can generate list of used objects/methods for TPP
Only available for Add-Ons using in MS .NET (uses Profiling API of MS .NET)
B1TE - Form Checker

Checks a form against the B1 programming and look-and-feel rules guidelines

Lists all the possible issues encountered in a form itself

Can check XML forms as well as forms shown in B1
B1TE - Bubble Checker

Lists all events sent by the B1 application

Marks the events that are stopped by an add-on (BubbleEvent set to false)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>MD</th>
<th>Event Type ID</th>
<th>Before</th>
<th>Bubble</th>
</tr>
</thead>
<tbody>
<tr>
<td>et_CLOCK</td>
<td>124</td>
<td>51</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>et_VALIDATE</td>
<td>134</td>
<td>5</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>et_VALIDATE</td>
<td>134</td>
<td>5</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>et_GOT_FOCUS</td>
<td>134</td>
<td>51</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>et_CLOCK</td>
<td>134</td>
<td>51</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>et_DOUBLE_CLICK</td>
<td>134</td>
<td>51</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>et_FORM_DEACTIVATE</td>
<td>134</td>
<td></td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
B1TE - Use of MSSQL Profiler

Tracks all database operations, those done with and without using the DI API.

B1TE provides some templates for Microsoft SQL Server Profiler or the MSDE’s OSQL command line tool.
B1 Test Composer (B1TC)

Motivation
- give to partners a simple way to test their add-ons

Core features
- record, replay, check values
- batching tests, selecting tests in a batch

Independence from 3rd party SW
- self consistent and free
- No dependency from any licensed tool

Automatic generation of test documentation
- want to run tests, not writing test documentation
B1TC – Record and Play tests cases

Record window

Play window

1. Press on menu "Sales - AV1"
2. Press on menu "Sales Order"
3. Open form "Sales Order"
4. Execute menu "Modify Order"
Appendix 2 – SDK Installations & Support Processes

Contents:
- SDK Installations
- Partners support process
  - Customer message
  - DRQ
- Market Place overview
  - How to open a customer message
  - How to download patches
  - How to Order License File
  - Naming Conventions
  - Searching for notes
  - RKT self Learning
- SDN Developer Area and Forum
SDK Installations: Objectives

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

- List the components of the SAP Business One SDK
- Tell some details about DI API installation
- Describe what is in the “SDK installation”
SDK Installations
1. SDK Components

- DI API
  - Available for all existing versions
  - Java connector → Part of Server installation from version 2007
  - Optionally separate installation (Part of SAP Business One client installation)
- DI Server
  - Part of Server Tools installation
- UI API
  - Part of SAP Business One client installation
- UDO – User Defined Object
  - Built in into SAP Business One itself – no additional requirements
  - Please note that there is a path for extensions that has to be specified on the Company Settings “Path” page
- “SDK package” – contains:
  - Help
  - Samples
  - Tools

- Note: UI API version must be identical to client version
SDK Installations
2. DI API and JCo – Installation

Standard installation path in version 8.8:
C:\Program Files\SAP\SAP Business One DI API

DI API: Part of the client installer

JCo included in DI API Installer:
C:\Program Files\SAP\SAP Business One Server DI API\JCO\LIB

- Version 6.5, 2004 ➔ C:\Program Files\SAP Manage\SAP Business One DI API
- DI API is a separate installation for versions 6.2 – 2004
  ➔ This is something to check in addition in case of problems
- In 2005 DI installation is part of the Autorun for the client installation.

IMPORTANT:
- The DI API installation package in the B1_SHR folder doesn’t get updated by the upgrader; you will have to copy the new DI API installation package „manually“ to that location - in case you intend to install DI API on a machine where you won’t install the SAP Business One client application…
The SDK folder contains:
- Help & Documentation
- UDO library & header files
- Samples – for several platforms
  - Visual Basic .Net
  - C#
- Samples – for most major features
  - DI API
  - UI API
  - DI API + UI API
  - UDO
  - DI Server
- Tools
  - Registration tools
  - …for other tools please visit the SAP (Developer) Network www.sdn.sap.com
SDK Installations: Summary

You are now able to:
- List components of the SAP Business One SDK
- Tell some details about DI API installation
- Describe what is in the “SDK installation”
1a. Partner support process

Use SAP Service Marketplace alias “sbosupport”
Partner opens a Customer message
The Global Support Center (GSC) team gets the message and answers the partner.
- If needed, the message is escalated to Development support team.
- There are 3 Possibilities

1. Partner needs help in using the feature
   - Support sends sample and documentation;
   - Eventually a request may be considered “consulting“ and thus might be billable

2. There is a bug in the feature
   - BUG is transferred to development for fix

3. The feature doesn’t exist in the version the partner uses
   - If the feature doesn’t already exist in the future version, Start a DRQ process
1b. DRQ – Development Request Process

Partner that needs a feature that the API doesn't supply has to open a DRQ message
- DRQ - Development request for the continuous improvement of SAP Business One
- Any request for changes or improvements in the system from its current behavior
- Development requests should be handled through the DRQ process

Process
- Open message for component SBO-DRQ

The Local PM will receive the DRQ messages and handle the versions content
2. The SAP Service Marketplace

http://service.sap.com

To access, the SAP Service Marketplace you will need a login or “S-Number” (Somebody within your organization will be able to create S-Numbers if you don't have one yet.)

An “alias” is a URL-suffix that gives you access to a particular page on the SAP Service Marketplace.

Example: “smb” alias is: http://service.sap.com/smb

Useful sites

http://service.sap.com/notes
http://service.sap.com/knowledgebase
http://service.sap.com/namespaces
http://service.sap.com/smb

To contact the community or use its resources go to:

https://www.sdn.sap.com/irj/sdn/businessone
2a. Market Place – How to open a customer message?

www.service.sap.com/smb/sbo/support

- Log in using your s-user and password
- Click "SAP Business One Messages" (from the main page) → You will get the page which is displayed above.
- Click on the 'Create message' button
- Fill in the required fields as accurately as possible
Click the following link:
- https://websmp103.sap-ag.de/sbo-swcenter
- Choose from the links tree on the left side of the screen:
  - Support Packages & Patches
    - SAP Business One Releases prior to SBO 2004
      - SAP Business One A
      - SAP Business One 6.5 (choose the version you need)
      - Binary Patches
        - SAP Business One 6.5
        - Win 32
- While upgrading from one version to another (for example if you upgrade from version 6.2 to version 6.5) first you need to install the major release and than you need to upgrade to the most recent published patch. (In the slide below it is patch number 7).
License from SAP can be ordered from the SAP Service Marketplace http://service.sap.com/licensekeys

To order a license from SAP simply an S-User and the Installation number for which the license is requested for is needed.

Partners can order licenses for customers through the respective Installation Number.

Process:
- Go to http://service.sap.com/licensekey
- Select the Installation Number a license file should be requested for. The respective systems for this Installation Number will be displayed
- To modify an existing license choose the respective System, change data and request a new license file.
- Go to “Request New System” Link to request an new license file
- Fill in data and choose “Next Step”
In this screen the licenses of the different SAP Business One components and Partner Solutions can be selected and will be included in the license file.

Certified and Uncertified Partner Solutions can be selected in the drop down boxes. They’re registered via the local SAP partner management for the solution provider.

Customer specific solutions are shown in a personalized list for the partner, only. If a customer runs a customer specific solution, the partner has to order the license file (with the same transaction). The partner can register his/her customer-specific solution via his/her license request form. His/Her solution is then shown in his/her personalized license request form and can be selected for a license file for the customer.

Also expiration dates can be set to give partners the possibility to sent out demo or test licenses for their solutions.
Different solutions using the SAP Business One APIs that may be installed at a customer site may use the same name for the solution objects (UDT, UDF, form’s unique id, item’s unique id, exe files, dll files, etc.). This may cause conflicts, and as a result one or more of the solutions will not work properly.
To prevent conflicts with other solutions using the SAP Business One APIs that may be installed at a customer site
you've to use a name prefix for your solution objects
The name prefix "ROOT" followed by the delimiter "_" ensures unique names (for example, "ROOT_myname")
Why do we need Namespaces?
- To prevent conflict with other solutions using the SDK
- A tool for setting unique names for forms, Items and menu items, User Tables and User Fields

Name prefixes define a space of possible names for objects
Therefore name prefixes are commonly called Namespaces

The Namespace must be reserved at SAP to obtain a name prefix which is unique within the "SAP world"

Your Namespace (OXYZ for example) followed by the delimiter "_" ensures unique names – XYZ_mynname

The same Namespace can be used for more than one solution by using an organizational rule to ensure unique names within the company – XYZ_S1_mynname
2d. Market Place – Namespace – How to order?

Relevant note: 647987

SAP Business One Namespace Reservation Process

- Allows an automated Namespace Reservation through the SAP Service Marketplace
- Provides fast and real time order processing
- Requires that „SDK Development License“ has been ordered

http://service.sap.com/namespaces

- Customers and partners must have a contract relating to an SDK Development Version otherwise the request will not pass the contract check and the name space will be rejected.

- see note 647987 for more information about name prefixes and how to request them.
The Namespace is entered in the syntax /XYZ/
It must contain alphanumeric characters with a letter as the first character, have
a minimum length of 3 characters, and a maximum length of 8 characters

After pressing the 'save' button
- Error Message –the prefix is already reserved
- If the name space is not reserved it will be assigned to you company

Wait for the acceptance from SAP

Reserve your accepted name prefix in the SAP Service Market Place

Do not forget to use your Namespace in all your SAP Business One solutions
(Tables Names, User Defined Objects,…)

- **Hint for SDK6.01, SDK6.2:**
  Due to technical limitations a three character prefix must be used for SAP Business One SDK releases 6.01, and 6.2.

- see note 647987 for more information about name prefixes and how to request them.
To search for a note, use SAP Service Marketplace alias “notes” (http://service.sap.com/notes)

Select “Restrict by Software Components”, then enter your selection on the restrictions options, then after pressing Select choose the software component you are looking for a note on it:

- SBO-DI-API
- SBO-UI-API
- SBO-JAVACO
- SBO-DTW
- SBO-PAINTR
- … etc.

Or use “SBO*” for all notes related to SAP Business One
An SAP Online Knowledge Product (OKP) is a set of role-specific Learning Maps that give you timely, firsthand information on the implementation and operation of the latest SAP solutions or upgrades.

Whether you are working in development, sales, consulting or support, the relevant Learning Maps will update your knowledge on basic functionality as well as on the latest product release level. SMB Learning Maps are developed within the framework of Ramp-Up Knowledge Transfer (RKT).

Use the RKT for self update in new features.
3. SAP Developer Network – Developer Area + Forum

Join the community at: http://sdn.sap.com