ORIGAM

ORIGAM Architect

Getting Started Guide

Table of Contents

Architect Overview

ORIGAM Architect is a modeling tool used to develop ORIGAM applications. Using Architect you can model the following functionality:

- Data Model
- Business Model (validations, workflows...)
- User interface
- API

Architect UI consists of docked windows and documents, like most of integrated development environments (IDE's).

Differences to Traditional Development

Modeling in Architect is completely different approach to the traditional development. ORIGAM model is a domain specific language (DSL) for creating enterprise applications that typically contain data, forms, UI validations, workflows and API's.

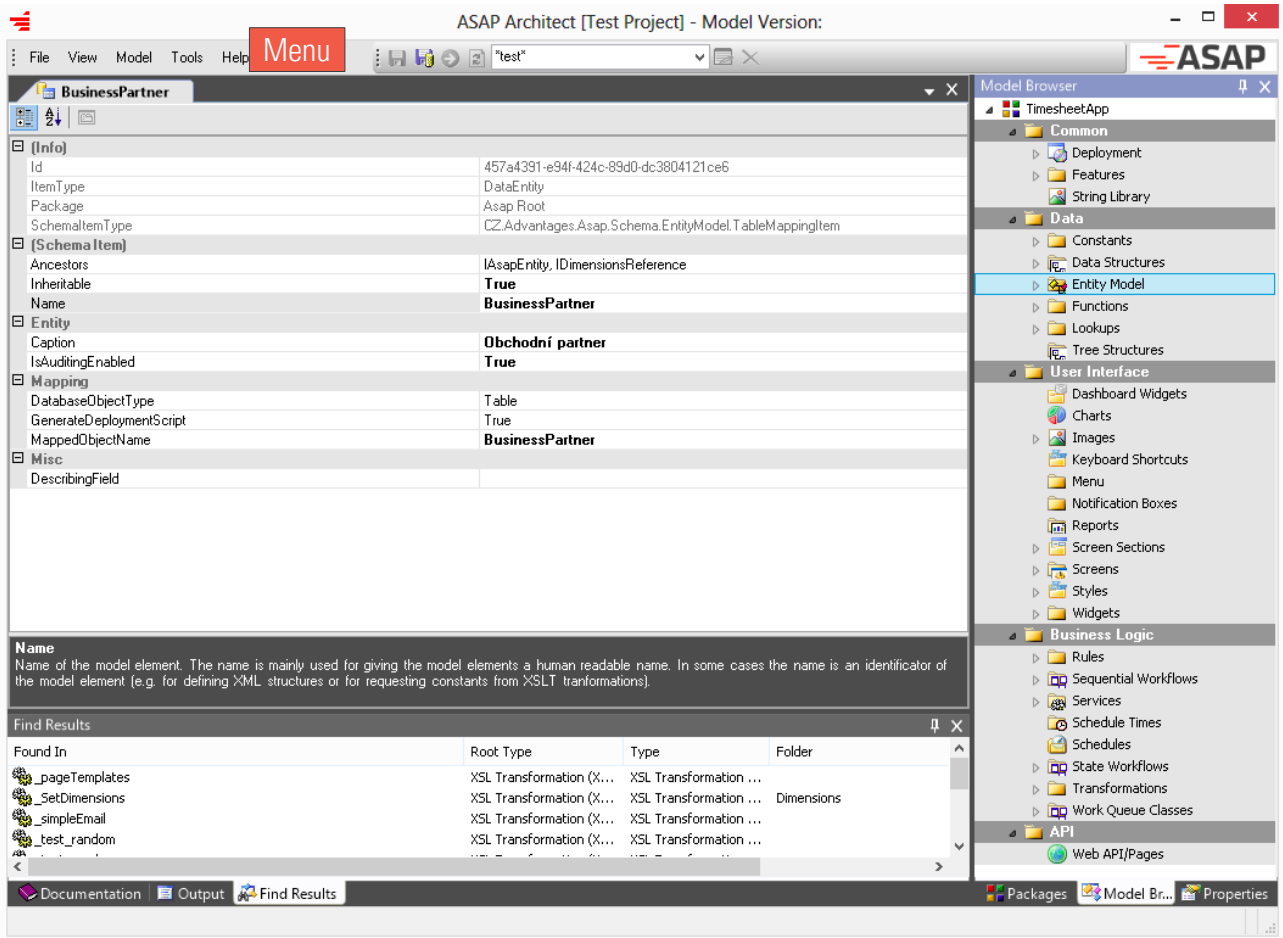
In traditional development you have to create a kind of a framework for each application you develop (function libraries, common classes etc.). With ORIGAM you have the basic building blocks encapsulated inside the model and you just express your requirements using the model. ORIGAM Runtime then takes as much care as possible to bring your requirements to life.

As for enterprise applications we tried to take all the best practices into the ORIGAM platform so you do not have to take care anymore about common stuff like writing proper SQL, application deployment, workflows etc. Now you need to think about **WHAT** your customer needs **instead of** trying to find out **HOW** to do stuff from day one.

Development Environment

ORIGAM Architect consists of the following basic elements:

- Menu
- Toolbar
- Documents
- Docked Windows



Menu

Menu contains all the commands you can use inside Architect.

Toolbar

The toolbar contains few basic commands and a search box for searching the model.

 Saves currently opened document

 Saves the model to the repository

Documents

Any data you will edit using Architect will be opened as a document. These are typically model editors or tools like Deployment Script Generator, settings editor, etc.

You can switch between the documents using tabs or you can close them using small black cross.

Docked Windows

Windows docked to the sides of Architect contain different tools for working with the model such as

- Package List
- Model Browser
- Properties
- Find Results
- Output
- Documentation

You can move docked windows as you wish inside Architect by dragging them around. If you close them, you can open them again using menu View.

By double-clicking on the docked window header you undock the window. By double-clicking again you put the window back to the dock.

Package List

Packages are the main containers of model elements. Here you can create, delete or open packages.Model Browser

In the Model Browser you see all the elements in the current package and all imported packages.

Elements in the current package are displayed in black color whereas imported packages are displayed gray.

By double-clicking on the model element you open its editor.

Properties

Some model editors use the Properties window to edit different properties. The example is a Form editor which uses this window to edit properties of different components placed on the form.

Find Results

This window displays results from using the search box or from exploring model dependencies. By double-clicking on the result you will be navigated to the selected element in the Model Browser.

Output

In this window you can see output of different tools. It can be a live application log or preview of generated SQL statements, XML files and other textual information.

Documentation

Using the Documentation window you can document the model. Some of the documentation will be used in the generated documentation and some can even be used in the final application, e.g. for displaying text in tooltips.

Model Contents

ORIGAM model consists of all the building blocks of an application. Every part of the solution is modeled and later interpreted by the ORIGAM Runtime.

Common Model

In the Common section of the model you can find general elements that you will need throughout the modeling process of your application.

Common model has the following parts:

Deployment	Deployment model allows you to define model versions and database upgrade scripts between those versions. Deployment scripts will get executed when you deploy a new version to the customer.
Features	Features allow you to define parts of the model that can be unlocked by allowing the feature by the user. This is handy e.g. for defining BETA features that will not be visible to a customer unless he turns such a feature ON.
String Library	To allow full localization of the model to different languages, error messages and other texts can be stored in the String Library. When defined here, those messages can be localized.

Data Model

Data model is at the lowest level in the modeling process. It is the basic part of the resulting application. After the data model is defined all following modeling makes use of it. For example when designing screens the editor offers list of data fields so you can just drag and drop to the form designer.

Data model has the following basic parts:

Constants	Data constants allow you to store different constant values that will be used elsewhere in the system, e.g. for defining conditions.
Data Structures	When Entity model defines the basic data entities, Data Structures define the structures of the screens, API's and workflow contexts. With Data Structures you also predefine filter sets, sort sets and rule sets to be used later with the structure. Data Structures are built on top of entities and use them to get list of fields, basic filters, etc.
Entities	Entity model is the most important part of the ORIGAM model. It defines data

	entities which in turn define database tables for storing customer data. Entities contain not only fields but also predefined filters, row-level-security rules, row-color rules and others.
Functions	Functions define a set of functions you can use for modeling data filters or simple calculated entity fields. You can add your own database functions (that you create manually in the database and map using this part of the model).
Lookups	Lookups define the way how to fill-in combo-box lists and how to translate id's to names both in the user interface and business logic.
Tree Structures	Here you can define hierarchical structures that you can later use in the user interface.

User Interface Model

You will use the user interface model to model ORIGAM GUI. That is to model a user interface that will be displayed by the ORIGAM client application.

Dashboard Widgets	Dashboards Widgets are predefined parts (e.g. charts, tables, values) that users can later add to their dashboards.
Charts	Definition of charts for the UI.
Images	List of images that can be used as menu or action button icons.
Keyboard Shortcuts	List of keyboard shortcut definitions that can be assigned to action buttons.
Menu	Defines the hierarchical user menu in the client application.
Notification Boxes	With notification boxes you can define content of notification boxes at different places in the application.
Reports	Reports model allows you to assign either internal reports (e.g. Crystal Report files) or web reports (external reporting systems URLs).
Screen Sections	Screen section is a predefined UI block on top of a single entity. It can contain all the editable fields from the entity.
Screens	Screen is a set of screen sections (e.g. in a master-detail scenario).
Styles	Lists possible styles that can be assigned to widgets inside the screen designer. The list is fixed for now.
Widgets	List of possible widgets that can be used for building the ORIGAM GUI. It also lists all screen sections since these are considered as widgets in the screen designer.

Business Logic

This part of the model brings the application "alive". Validations, state and user workflows, functionality—all is defined in this part of the model.

Rules	XPath expressions or XSLT scripts that are used in the following parts of the model: > Validations > Sequential workflow conditions > Row-level-security conditions > Conditional formatting conditions > State transition conditions > Calculated fields
Sequential Workflows	Define the functionality of the application. They can be called from: > Menu (for creating Wizard-like interfaces) > Screens (using action buttons) > Data events (record created/updated/deleted) > State workflow events (changing states) > Work Queue Actions > API > Other sequential workflows
Services	Lists available service agents that can be called in sequential workflows. You can also write your own agents to connect with other services.
State Workflows	State workflows define states and allowed transitions between them on an entity-field level. By defining a state workflow on a field the field gets checked for state rules and starts producing state events.
Transformations	XSLT Transformations that can be used in: > Sequential workflows (using Transformation Service) > API (transforming raw-data)
Work Queue Classes	Define a data structure of a work queue and possible actions on the specified work queue class.

API

This part of the model allows you to define REST API methods for your application or to directly create web pages.

Name	Description
Web API/Pages	Here you can define API methods that will face out your application. Using API's it is possible to: > Read data in different formats (XML, JSON, CSV...) > Create web sites (delivering data in HTML) > Write back data (XML, JSON) > Execute sequential workflows

TimesheetApp Tutorial

In this tutorial you will learn how to create an application in several steps.

Analysis

The application will consist of a form with the following sections:

- Project
- Time-sheets

Summary

In this tutorial you will learn how to:

- Create a package
- Create a simple data entity
- Create database scripts and execute them
- Create a screen and a menu item
- Create a screen flow

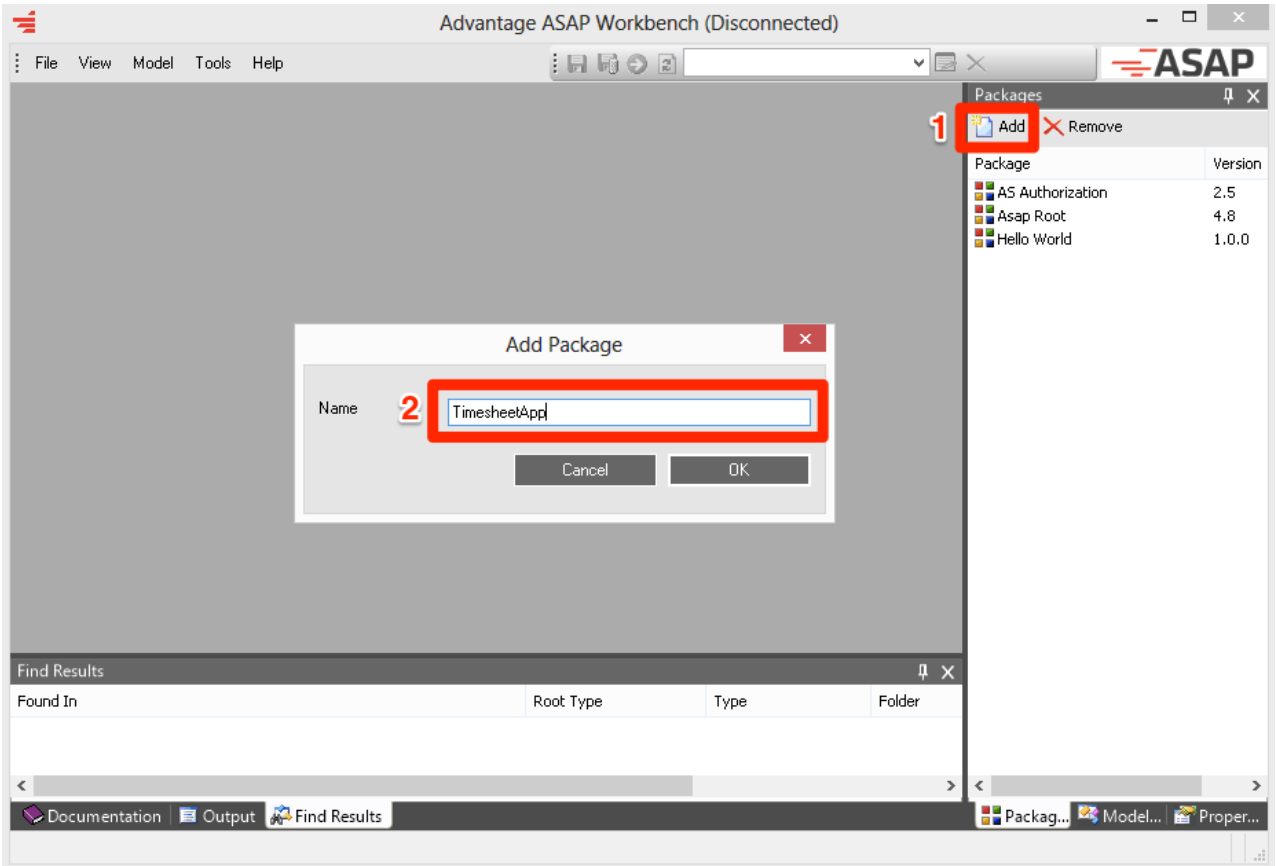
Creating a Package

Model consists of packages, so we begin with creating one.

1. Open ORIGAM Architect
2. In the Packages window click on Add button
3. Enter the name of the project "TimesheetApp"
4. Click OK

Now an empty package is created and loaded.

Note there are already elements in the package. That is because AsapRoot package was automatically referenced.

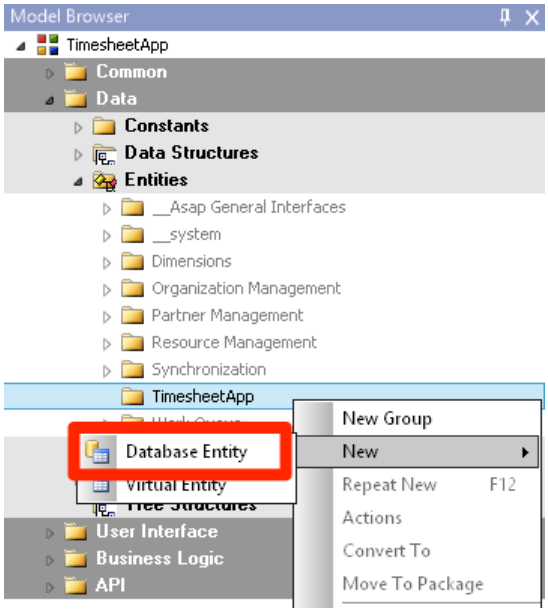


Creating a Project Entity

The first thing to be created is a data model. It is the most basic part of the application.

First we will create the Project entity for storig project data. We will also allow data auditing.

1. Open **Data > Entities > TimesheetApp** folder
2. Right click and select **New > Database Entity**
3. Enter "Project" into the **Name** field and also to the **Caption** field
4. Set **IsAuditingEnabled** to **True**
5. Click the **Save** button in the toolbar



Please note that the entity has already some fields thanks to the **IAsapEntity2** entity from which we automatically inherited.

Now add the following fields by repeatedly right-clicking on the Project entity and selecting **New > Database Field:**

Name	Data Type	AllowNulls	Length	Caption
Name	String	False	200	Name
EstimatePrice	Currency	True		Est. Price
EstimateTime	Float	True		Est. Time (h)
Description	Memo	True		Description

Save the model.

Generating a Database Script

After the entity is modeled you have to create the corresponding table in the database. You do this using the **Deployment Script Wizard.**

Deployment scripts will get executed when you deploy a new version to the customer. The scripts can contain database commands or files to be restored to the target server.

1. Choose **Tools > Deployment Script Wizard** from the menu

2. **Mark** the checkbox at the **Project** entity
3. Click on the **Add to Deployment** button
4. Answer **Yes** to the question if you want to see the generated scripts in the search results.
5. Double-click on the item in the search results and you will see the script highlighted in the Model Browser.
6. Right-click on the script item and choose **Execute.**
7. Now the Project table has been created in the database.

Save the model.

Creating a Menu

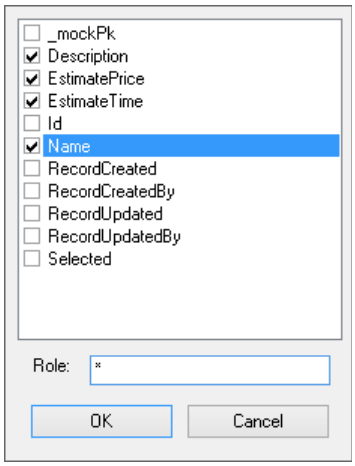
Before you can start creating a user interface you have to create a single menu in the Menu model. Not all packages have to contain a menu (consider them libraries) but finally you will have to create one if you want to use ORIGAM GUI.

1. Right-click on the **User Interface > Menu** and select **New > Menu**
2. Call it simply “Menu” and set DisplayName property to “Menu” as well.
3. Click the **Save** button in the toolbar

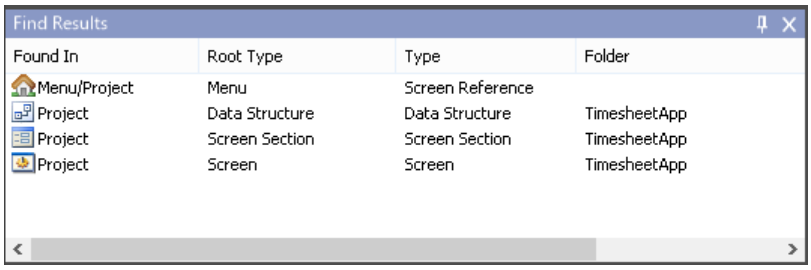
Creating User Interface

The last step in developing a simple application is to create the user interface, which will consist of a single menu item which will open a screen with an editable table.

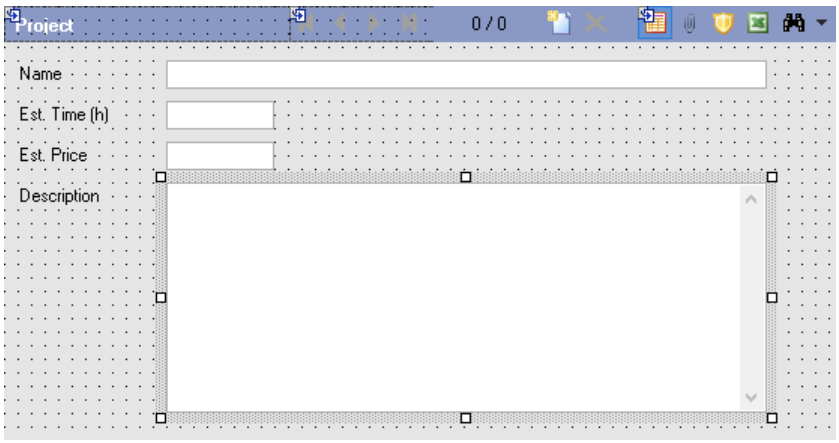
1. Right-click on the "Project" entity and choose **Actions > Create Menu Item...**
2. Now choose the fields you want to display in the UI. Set "Role" to * since for now we do not want to create security roles.



3. Press OK.
4. In the Find Results docked window you will see the list of model elements that were now created for you automatically.



5. Double-click on the screen section so you can fix the field order, sizes etc.

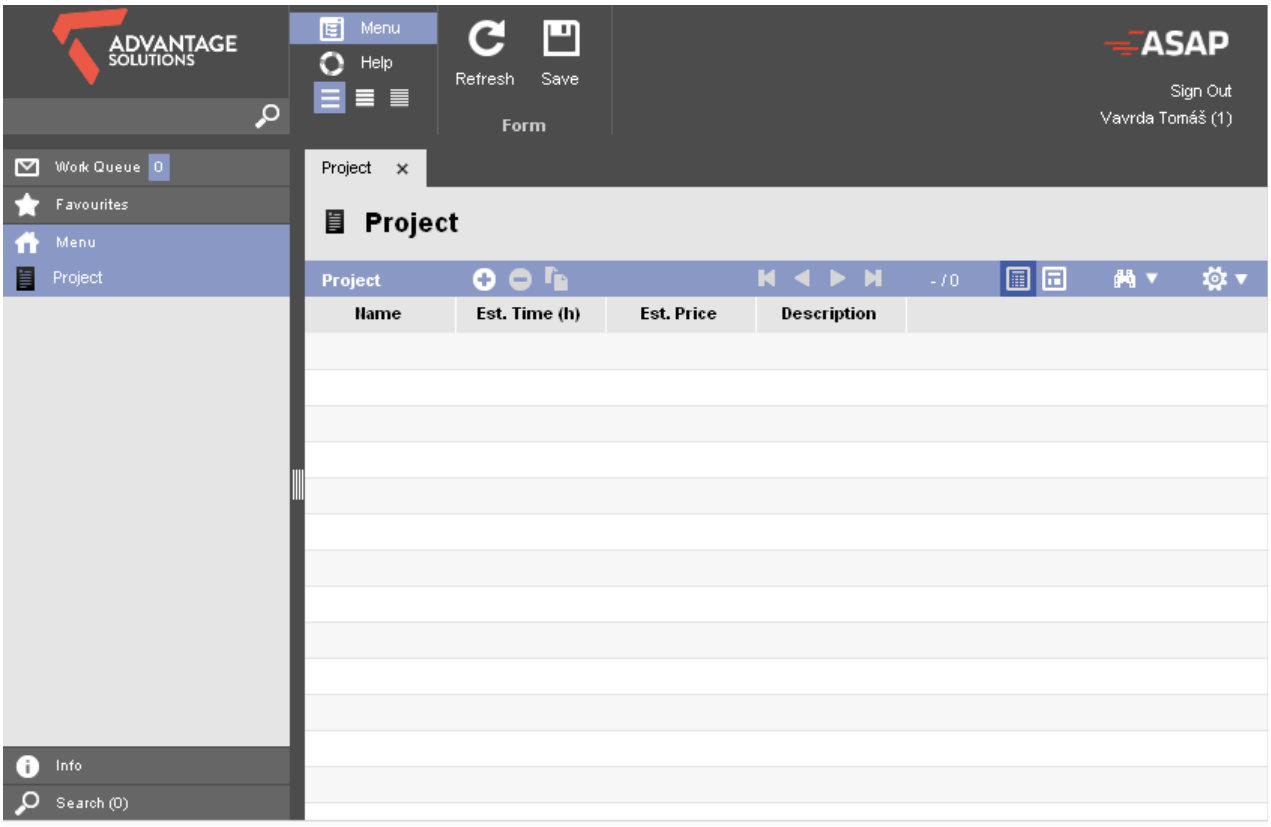


6. Use **Ctrl+Shift+T** to fix a tab order (by clicking one-by-one on the widgets).
7. Click on the Description field and use the **Properties (F4)** docked window to set the **Multiline** property to **True**.
8. Change the menu item's name to “Projects”.

See the Results

As the final step you can review the results by running your application.

1. Make sure to have **DefaultSchemaExtensionId** set in your **AsapSettings.config** file. The value must be an Id of your package. To see your package id right-click on the top-level node **TimesheetApp** in the Schema Browser docked window and select **Edit ….** Copy the value from the Id field and paste it into AsapSettings.config.
2. Restart the IIS application pool
3. Open the application.
4. Open the menu Projects



Using BusinessPartner Entity

When you look into the data database you can see there many existing tables. You can see these as entities that you inherited from the Root model.

You will not directly use most of these entities because they are used by the infrastructure.

There is an exception from this–a BusinessPartner entity.

This entity has both infrastructure and custom use. It is used as

- List of users with mapping to login user names
- A base for an address book

The idea is that you will extend this entity with your custom fields (phone number, e-mail or any other attributes you will see fit for your project's address book) and at the same time it will be used as your user database (users are your business partners too).

Adding BusinessPartner Screen

Let's start with looking at what the BusinessPartner table contains and being able to add some data. You can design your own screen by following the same steps as when you were creating a screen for a **Project** entity.

Creating a Foreign Key To BusinessPartner

Now it would be nice to be able to pick a business partner in our project. We can do this by:

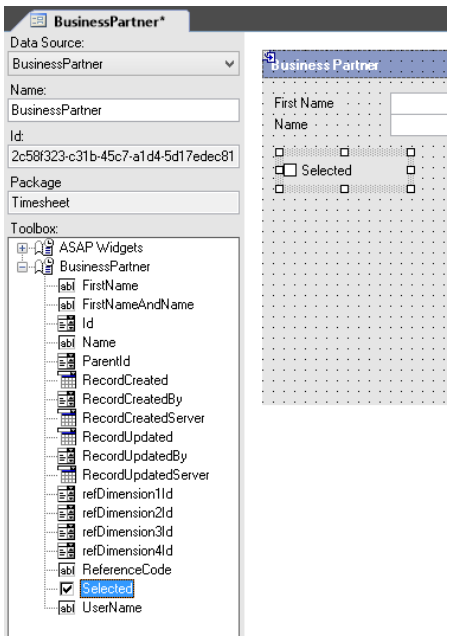
- Adding a **foreign key field** to the Project entity that will link to the BusinessPartner
- Adding a **drop-down** widget to your Project screen

Creating a Simple Screen Flow

A screen flow is a sequence of screens that you can present to a user, much like a Wizard kind of interface. In order to create it you will create:

- Sequential Workflow
- Screens
- XSLT Transformation
- Menu Item

> Deployment Script



In the following chapters we will create a screen flow that will allow the user to easily add a new project.

Creating Screens For a Screen Flow

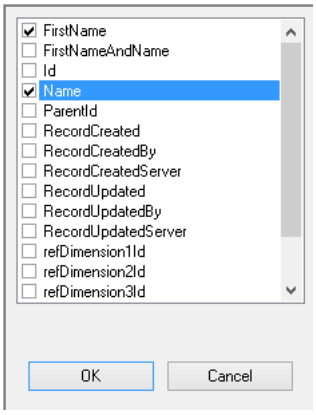
You can use any screen that you learned to create in the previous chapters but we will make two screens that will be optimized for the screen flow:

> Customer Selection Screen
> Single Project Entry Screen

Customer Selection Screen

We will base this screen on a BusinessPartner entity which became our system's address book and we will let the user select one of the business partners as the project's partner.

1. Right-click on the "BusinessPartner" entity and choose **Actions > Create Screen...**
2. Then we will choose the fields that we want to display.



3. Now we will add a checkbox that will allow the user to select the business partner for the new project. Open the newly created BusinessPartner screen section and drag the Selected field to the section.
4. Click on all entry fields (Name, First Name, Selected) on the section and set the **ReadOnly** attribute to **True** (Press F4 to display the Property Grid). We do not want the user to edit any of these fields while in the wizard.
5. Save the screen section.
6. We will also need to hide the Add and Delete buttons on the screen sections so open the Business Partner **Screen**, select the screen section on it and se the **ShowNewButton** and **ShowDeleteButton** to **False**.
7. Rename the screen to **BusinessPartner_Selection** and save the screen.

Project Entry Screen

In the project entry screen we will reuse the screen section for the project entry we already created. We will just make it displaying in the Detail View by default and we will hide the Navigation Panel because when entering a single project we do not need one.

1. Right-click on the existing "Project" screen section and choose **Actions > Create Screen...**
2. This will immediately open a new screen based on the selected screen section.
3. First rename the screen to **ProjectEntry.**
4. Then select the section on the screen and set **HideNavigationPanel** to **True**. Also set the **ShowNewButton** and **ShowDeleteButton** to **False**.
5. Save the screen.

Now we are done with preparing the user interface for the screen flow.

Creating a Sequential Workflow

In order to see the screens displaying one by one we need to create a Sequential Workflow.

Sequential workflows can be use to implement business logic's functional flows but also to create screen flows. By assigning a sequential workflow to a menu item we will get a screen flow.

Step 0: The Sequential Workflow

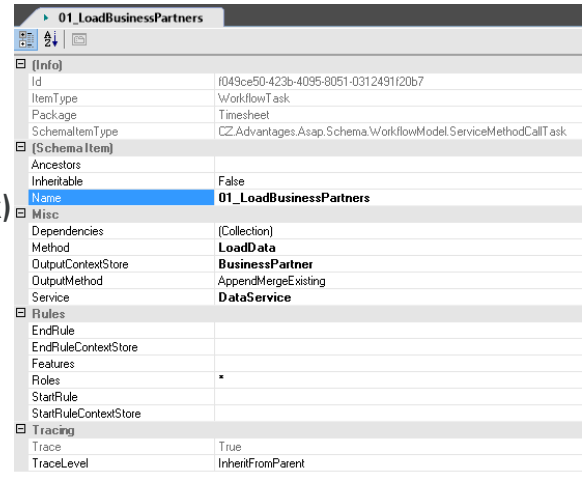
Now we will create a new sequential workflow and define some **Context Stores**.

A Context Store is much like a Variable in a traditional programming language. We will use a context store to store the data which the sequential workflow will manipulate.

1. Right-click on the "Business Logic > Sequential Workflows" model part and select **New > Sequential Workflow**
2. Name it "ProjectEntry"
3. Set the **TraceLevel** to **TraceClientAndArchitect** so we can later debug this workflow.
4. Save the workflow.
5. Right click on the newly created workflow and select **New > Context Store**.
6. We will use an existing data structure **BusinessPartner**, so select one from the DataStructure list.
7. Save the context store.
8. We will ad one more context store that will be based on the **Project** data structure (repeat steps 5 to 7).

Step 1: Loading Business Partner List

As we said the first step would be displaying a list of business partners and letting the user select one. So the first thing to do is to actually load the list from the database.



1. Right-click on our workflow and select **New > (Task) Service Method Call.**
2. Select a Service **DataService**, Method **LoadData** and OutputContextStore **BusinessPartner**. That means we will use a DataService.LoadData to fill in the BusinessPartner context store.

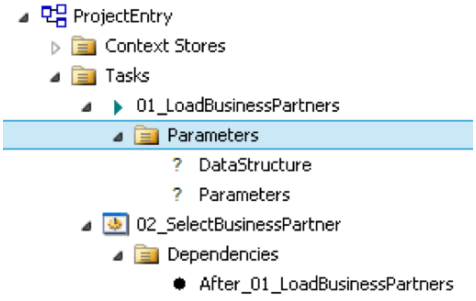
   In case you do not want all the data from the database you need to specify a filter and eventually a

sort order. In that case right click on the DataStructure under the task and select **New > Data Structure Reference** and define your own parameters.

Step 2: Business Partner Selection Screen

After we have a list of partners loaded we can present them to the user so he can select one.

1. Right-click on our workflow and select **New > (Task) User Interface.**
2. Select a Screen property to **BusinessPartner_Selection** and OutputContextStore **BusinessPartner**. This way the wizard will show our selection screen and will use data we loaded in the previous step.
3. Since this is a second step, we need to tell that to the engine. Right click on this step and select **New > Dependency** and select the first step in the Task property.



Step 3: Preparing a New Project Record

Now the user selected a record using a checkbox we want to pass it as a new value to an XSLT transformation that will prepare a new Project record for us. For that we will add a third step.

First we will prepare an XSLT transformation:

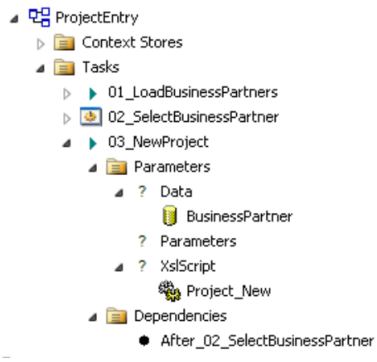
1. Right-click on the "Business Logic > Transformation" model part and select **New > XSL Transformation (XSLT).**
2. Name it **Project_New** and add the following as the transformation body:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
	xmlns:AS="http://schema.advantages.cz/AsapFunctions"
	xmlns:date="http://exslt.org/dates-and-times" exclude-result-prefixes="AS date">

	<xsl:template match="ROOT">
		<ROOT>
			<xsl:apply-templates select="BusinessPartner[@Selected='true']"/>
		</ROOT>
	</xsl:template>

	<xsl:template match="BusinessPartner">
		<Project
			Id="{AS:GenerateId()}"
			Name="{@Name}"
			refBusinessPartnerId="{@Id}"
			DueDate="{date:date()}"
		/>
	</xsl:template>
</xsl:stylesheet>
```

The transformation will go through each selected business partner and create a new project with a preselected business partner, name and due date (which is a mandatory field so it has to be filled otherwise we would get an error after the transformation would be processed).

Then we will add the transformation step to our screen flow:

1. Right-click on our workflow and select **New > (Task) Service Method Call.**
2. Select a Service **Data Transformation Service**, Method **Transform** and OutputContextStore **Project**. That means we will be generating data into the Project context (which we will display to the user in a screen in the next step). Name it **03_NewProject.**
3. Save the element.
4. Right-click on the **03_NewProject** and create a new dependency on task 02.
5. Now we will set the **Data** and **XslScript** parameters of the transformation step:
   a) Right-click on the **Data** parameter and select **New > Context Store Reference**.
   b) Choose the **BusinessPartner** context and save the element.
   c) Right-click on the **XslScript** parameter and select **New > Tranformation Reference.**
   d) Choose the **Project_New** transformation you created earlier and save the element.



Step 4: Displaying the Project

1. Repeat Step 2 but now with the **ProjectEntry** screen you created earlier.
2. Don't forget to create a task dependency on the previous step.

Step 5: Saving the Project

1. Repeat Step 1 but now with **DataService** method **StoreData**.
2. Don't forget to create a task dependency on the previous step.

Testing the Screen Flow

You can always test what you did by right-clicking on the sequential workflow and selecting **Execute (Ctrl+X)**.

Always check the Output pad for information about what is going on. If error occurs you can also see exactly what went on by selecting **Tools > Show Trace (Ctrl+T).** There you can look up the instance you executed and see the data that were intput and output for each step.

Important: You should turn off the tracing for production deployments because it creates excessive data.

Creating a Menu Item

Now when the sequential workflow is ready you need to assign it to a menu so the user can execute the screen flow. You can do this easily by right-clicking on the sequential workflow and selecting **Actions > Create Menu Item**. It works the same way as when you were creating your first screen–it creates a **Workflow Reference Menu Item** (which you can then re-order as needed in the menu structure) and an application role (with a deployment script for inserting the role into the database in case you selected a role different than *).

ADVANTAGE
SOLUTIONS

Advantage Solutions, s. r. o.

5. května 876
250 82 Úvaly
Czech Republic

+420 273 160 699
info@advantages.cz
www.advantages.cz