

UX402

Advanced SAPUI5 Development

PARTICIPANT HANDBOOK INSTRUCTOR-LED TRAINING

Course Version: 22

Course Duration: 5 Day(s)

Material Number: 50159449

SAP Copyrights, Trademarks and Disclaimers

© 2022 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see <https://www.sap.com/corporate/en/legal/copyright.html> for additional trademark information and notices.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials may have been machine translated and may contain grammatical errors or inaccuracies.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation



Demonstration



Procedure



Warning or Caution



Hint



Related or Additional Information



Facilitated Discussion



User interface control

Example text

Window title

Example text

Contents

vii	Course Overview
1	Unit 1: SAP User Experience and SAPUI5 Strategy
3	Lesson: Describing SAP User Experience Strategy
17	Lesson: Explaining SAP User Experience Tools and Technologies
21	Lesson: Describing SAP User Experience Use Case for Building Fiori-like Apps
37	Unit 2: MVC Review and Advanced UI Controls
39	Lesson: Performing an MVC Architecture Review
45	Lesson: Binding Data to a UI5 Control
55	Lesson: Describing Best Practices for SAPUI5 Applications
65	Lesson: Implementing App Navigation
73	Lesson: Implementing a Full-screen Application
93	Lesson: Implementing a Master-Detail Application
117	Lesson: Working with Messages
125	Lesson: Describing Key Responsive Design Controls
137	Lesson: Extending Standard Controls
145	Lesson: Describing Custom Controls
151	Lesson: Creating Control and Component Libraries
159	Lesson: Implementing Unit Tests with Qunit
167	Lesson: Implementing One-Page Acceptance (OPA) Tests
197	Unit 3: Advanced Data Handling
199	Lesson: Describing Remote vs. Local OData Services
207	Lesson: Working with the MockServer
211	Lesson: Working with the ODataModel
221	Lesson: Describing OData Deep Inserts
225	Lesson: Introducing SAPUI5 Smart Controls
231	Lesson: Working with SAPUI5 Smart Controls
237	Lesson: Introducing SAP Fiori Elements
267	Unit 4: Application Extensibility, Introduction
269	Lesson: Introducing SAPUI5 Flexibility
273	Lesson: Explaining Extension Points
283	Lesson: Describing Other Types of Extensibility in SAPUI5
295	Unit 5: Version Control - Working in Teams
297	Lesson: Working with GIT
305	Lesson: Working with GIT Repositories
313	Lesson: Working with Branches

Course Overview

TARGET AUDIENCE

This course is intended for the following audiences:

- Application Consultant
- Developer
- Enterprise Architect
- Solution Architect

UNIT 1

SAP User Experience and SAPUI5 Strategy

Lesson 1

Describing SAP User Experience Strategy

3

Lesson 2

Explaining SAP User Experience Tools and Technologies

17

Lesson 3

Describing SAP User Experience Use Case for Building Fiori-like Apps

21

UNIT OBJECTIVES

- Describe SAP User Experience design methods for SAP Fiori-like application
- Explain how SAPUI5 aligns with the overall SAP User Experience strategy for online and mobile apps
- Describe Use cases for Fiori-Like Apps

Unit 1

Lesson 1

Describing SAP User Experience Strategy



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe SAP User Experience design methods for SAP Fiori-like application

User Experience Strategy

Scenario

As a developer, you are requested by the business process owners to permanently improve the user experience.

Creating user interfaces is a critical aspect in the user experience. In this course, you will extend your knowledge in developing great SAPUI5 user interfaces.

The World is Changing



Our opportunity, our vision

Deliver a **seamless** digital experience for **you** & your business

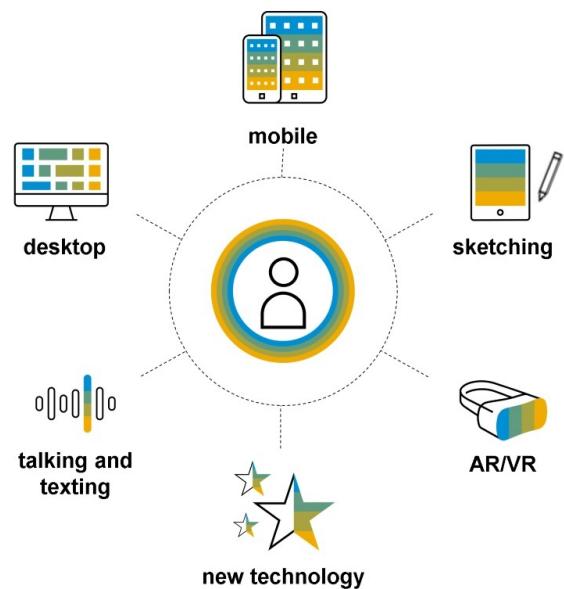


Figure 1: The World is Changing

Take for example, purchasing something personal online like a television. That same user would expect if they needed to order something from work, say a new cell phone that the process would be the same in terms of ease and simplicity. This idea has set the bar for expectations when using business applications.

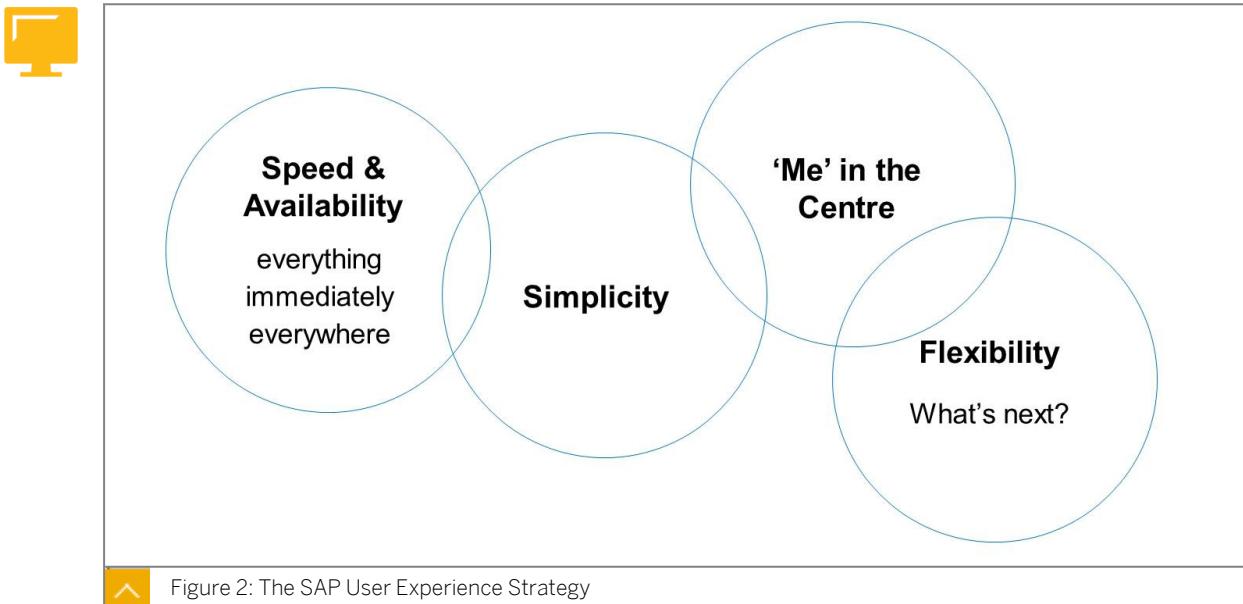
Consumerization is the growing tendency for new information technology to emerge first in the consumer market and then spread into business and government organizations.

This is evidenced today more so than ever before. People are used to the application user experience they interact with their daily applications such as Google and social media apps.

This experience has been embraced and accepted by people to the point where it is now the new standard. People want to have the same easy, feature rich experience they have with their personal sites in the business place.

SAP has recognized this change in society and made it our goal to meet this new standard.

Digital Transformation — The Influence on User Expectations



SAP developed a user experience strategy consisting of 3 main components: New, Renew, and Empower.

During our research phase we realized that most users still use the SAP GUI to access applications.

The GUI contains approximately 300,000 screens and consist of a vast number of functionalities.

We looked at all the functionality offered in our GUI and developed a list of the most frequently used applications, namely manager and employee functions such as leave request or travel expenses. We decided it was time to renew these top scenarios to make good on our mission.

It was from this idea that SAP Fiori was born.

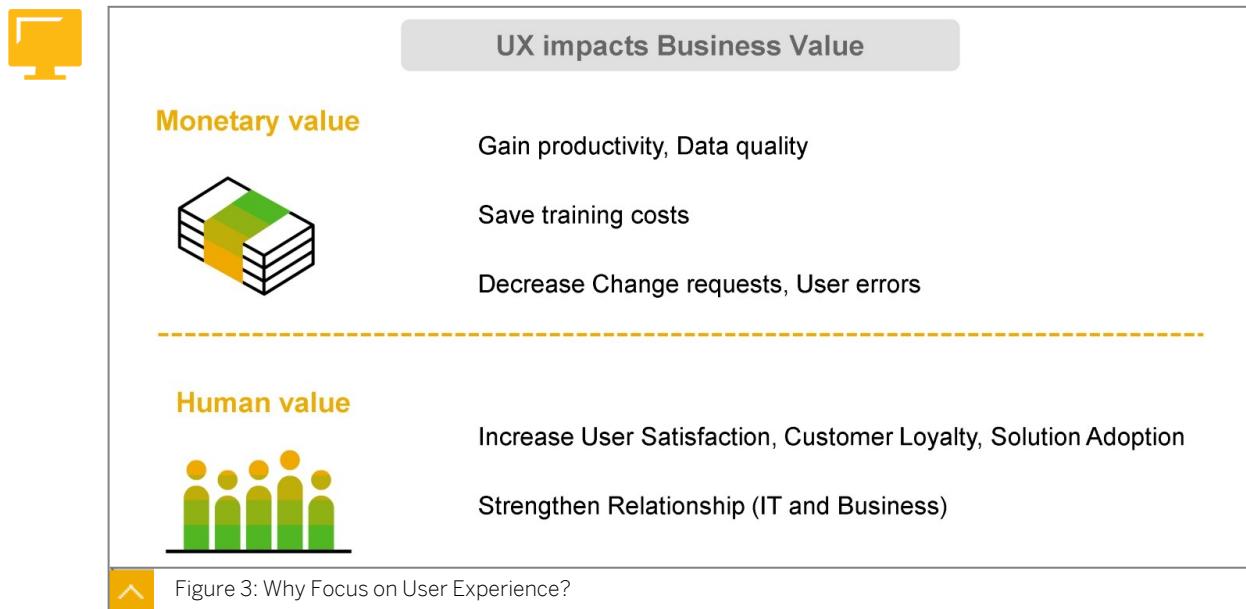
We also decided to renovate business suite programs and enable customers to improve their user experience on their own.

For example, SAP developed SAP Screen Personas which allows customers to optimize and simplify any screen in the GUI. While developing SAP Fiori we also decided to renovate our business suite programs and provide enablement tools to enable customers to improve their user experience on their own - take for example, SAP screen personas that allows customers to optimize and simplify any screen in the GUI.

While we were working on renewing our existing solutions and enabling our customers we also continue to develop new applications. All the while SAP is continuing to create new innovative applications to fulfill customer demands and needs.

Not looking at creating a new business process, looking at creating a new user experience without changing the backend ("lipstick on a pig").

Why Focus on User Experience?

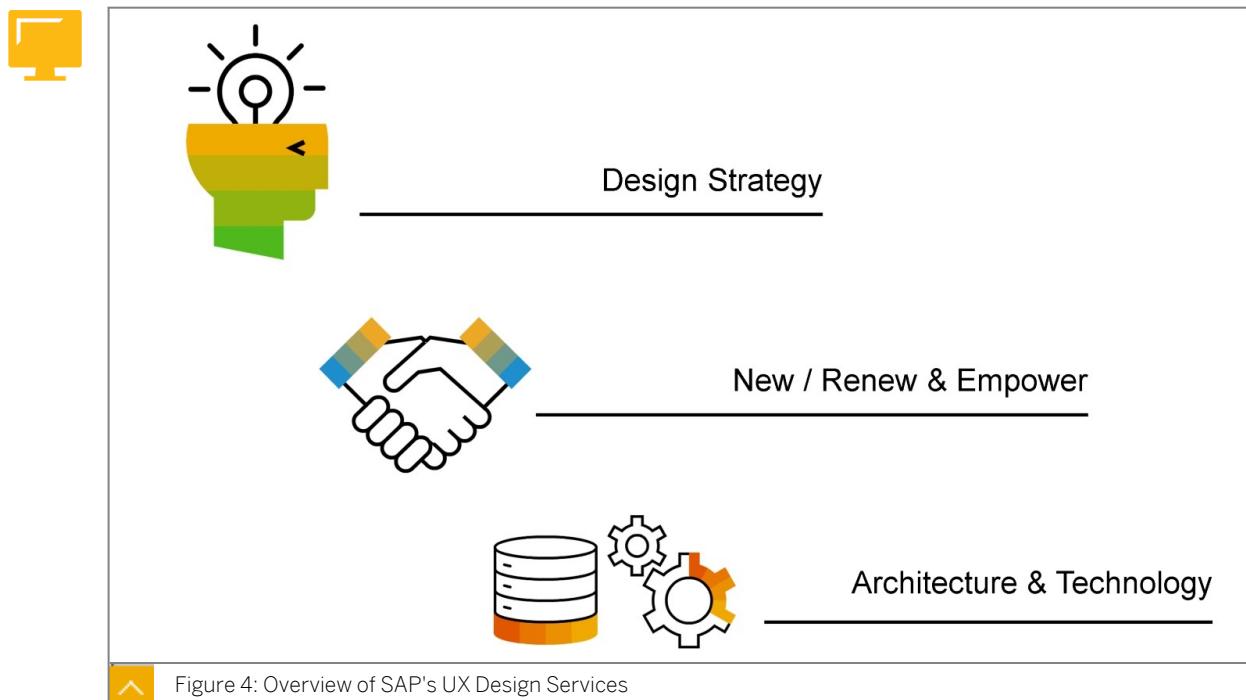


The figure shows the influences on monetary and human value.

SAP User Experience Design Services

Finally, we will look deeper into the User Experience design services component.

Overview of SAP's UX Design Services



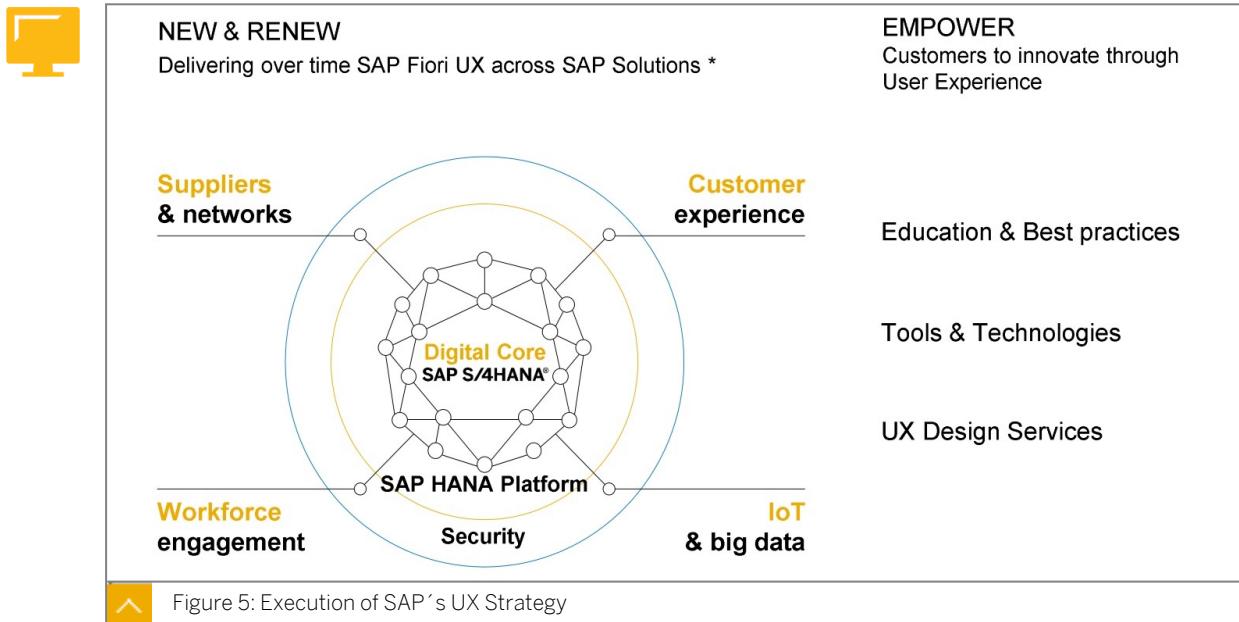
SAP Identified several leads from our customers and they asked us for advice to understand the SAP strategy and translate it to their reality: they asked for services to realize: implement, adapt and optimize the user experience of existing software.

If you achieved results such as with screen personas or Fiori the next level is to have customers build up skills on their own or empower their organization for a user experience strategy.

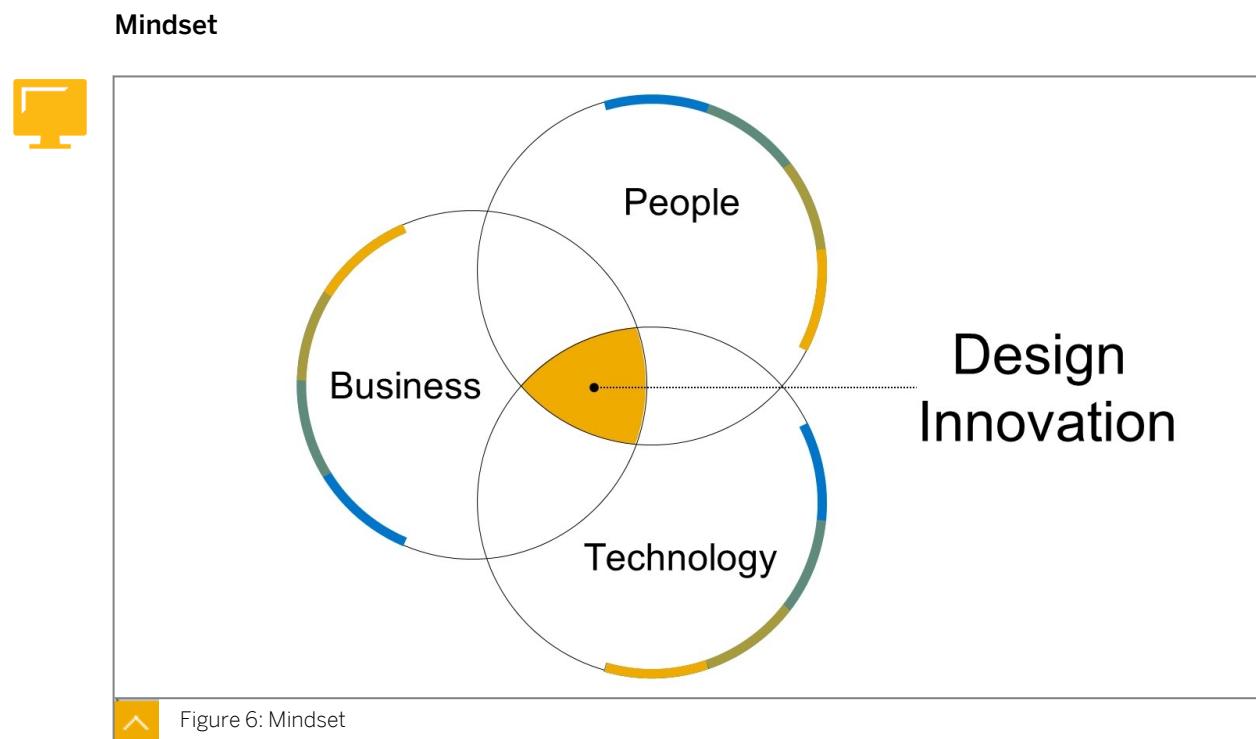
The last level is for the customer to become really innovative. This is the final goal of all customers, to be more innovative and they can achieve that by designing new products, looking for new services etc.

All of this is powered by design thinking. In summary: the design services that SAP offers are to advise about the strategy and the value of a solid user experience.

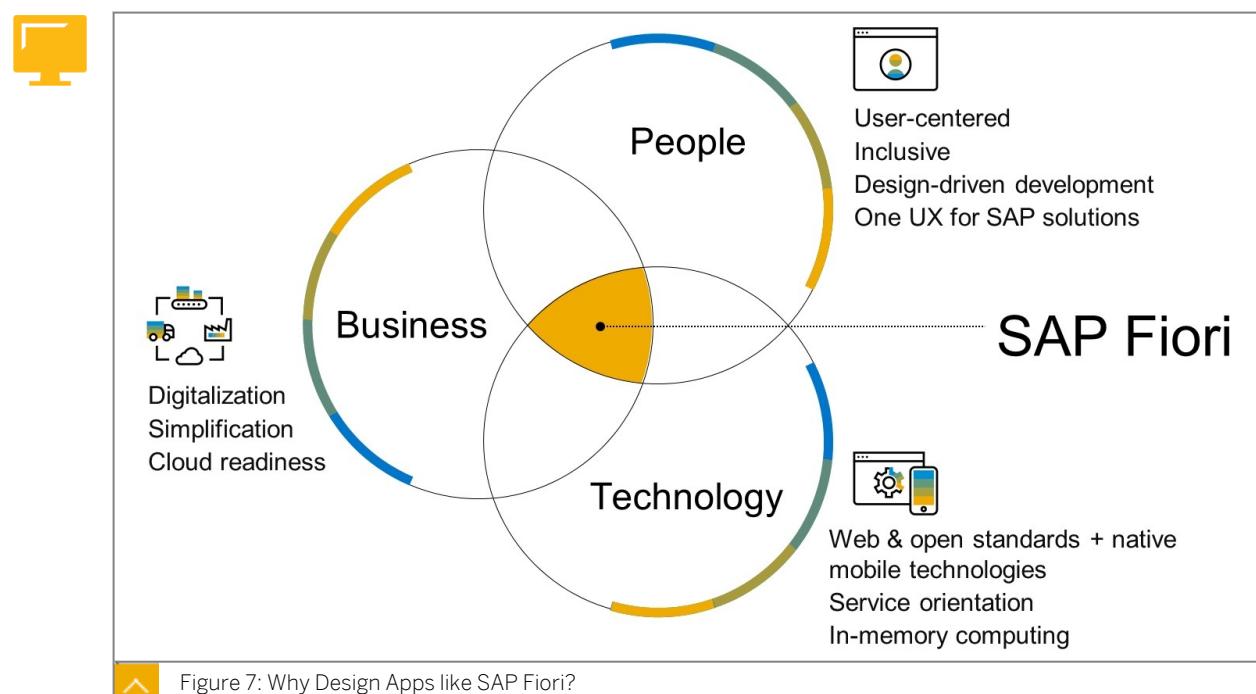
Execution of SAP's UX Strategy



The figure shows the execution of SAP's UX strategy.



SAP expanded their research to include the most commonly used functionality across all lines of business. With the first set of Fiori apps we focused on the most commonly used business functions and those focused on the HR, workflow, and SRM lines of business. Here you can gain an overview of the other business lines we expanded to such as research and development and finance.

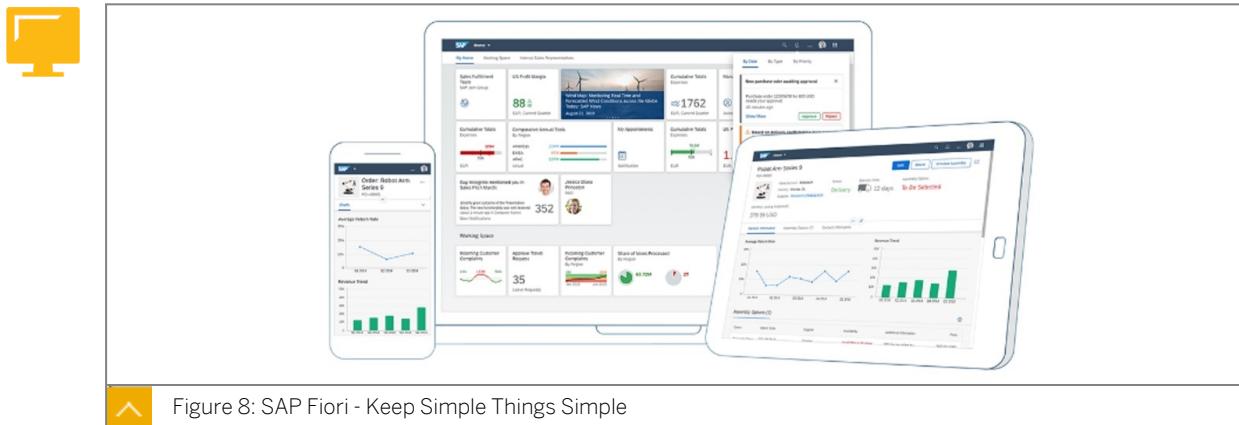


SAP Fiori is the design language that brings great user experiences to enterprise applications. Based on user roles and business processes, SAP Fiori simplifies doing business. SAP Fiori is a paradigm shift away from monolithic UI design principles.

The SAP Fiori Apps Library (https://fioriappslibrary.hana.ondemand.com/sap/fi/x_externalViewer) provides detailed information about all of the SAP Fiori Apps that are currently available.

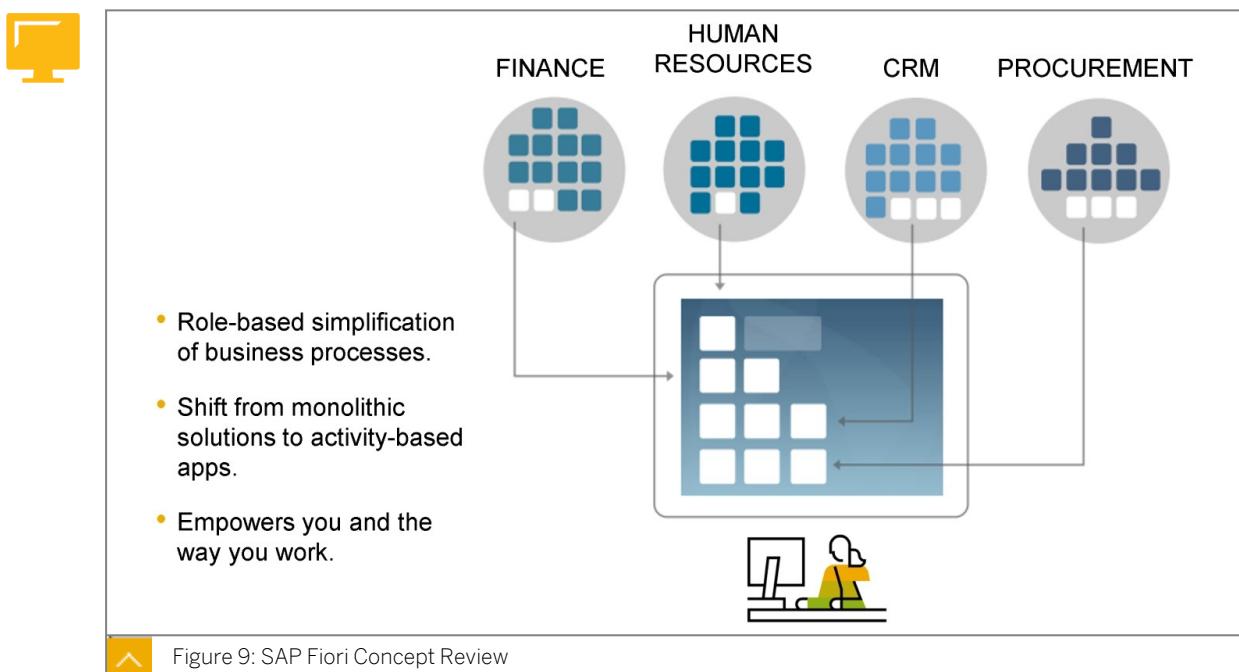
SAP Fiori is 100% geared toward business users. We wanted to provide a consistent user experience. SAP Fiori assures that people, both employees and managers have consistent, coherent, simple and intuitive user experience across multiple devices which allows them to work smarter, more efficiently and deliver on business objectives.

SAP Fiori - Keep Simple Things Simple



If you look at the tablet in this image, you'll see the detail list on the left side and the main pane. Now look to the mobile phone and notice only the main pane is visible. The detail list can be accessed by swiping the phone but both panes will not fit at once. Responsive design is credited for automatically completing the look on our UI framework.

SAP Fiori Concept Review



The figure shows the entry to the SAP Fiori Concept.

Idea of SAP Fiori



- SAP Fiori is more than just a collection of apps, it represents the new SAP User Experience paradigm based on SAPUI5 framework
- Like SAP Fiori, SAPUI5 offers the developer opportunities to develop various business roles into simple, easy-to-use experiences for SAP software functions, and works seamlessly on desktop, tablet, or smartphone

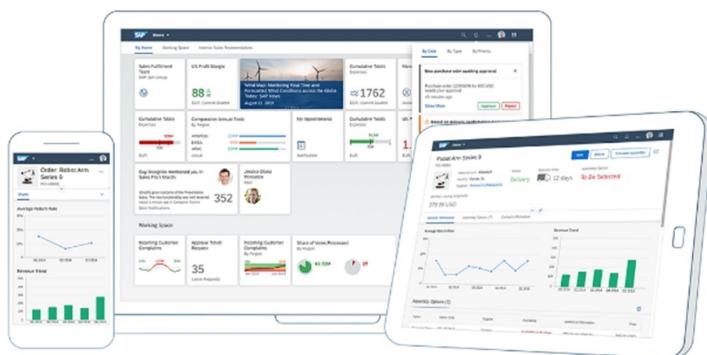


Figure 10: Idea of SAP Fiori

The figure shows examples of SAP Fiori on various devices.

SAP Fiori, Concept Review

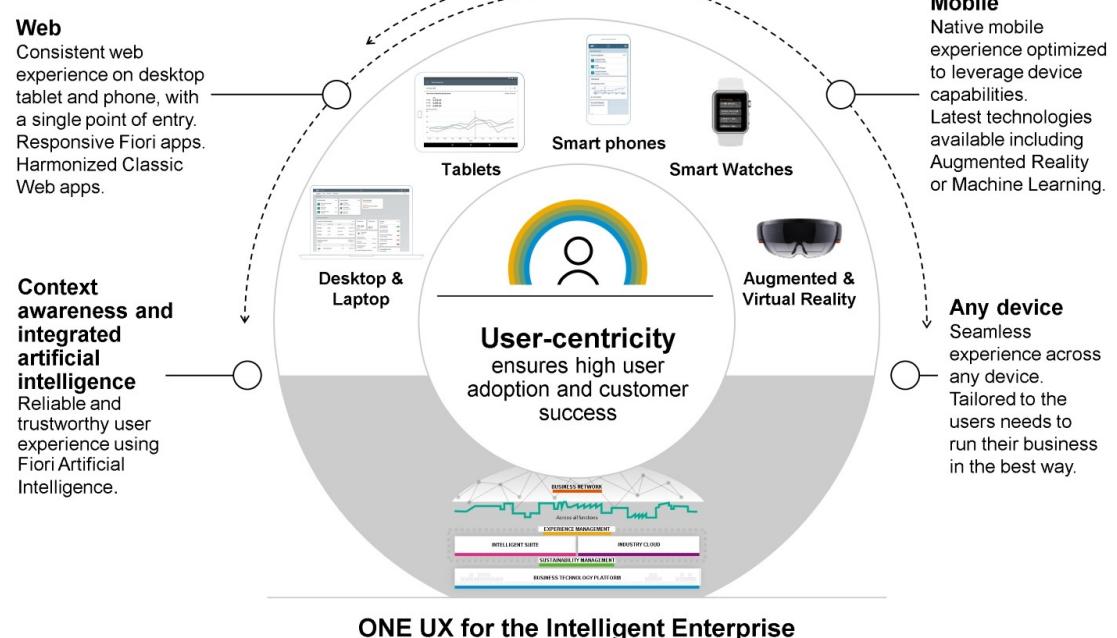
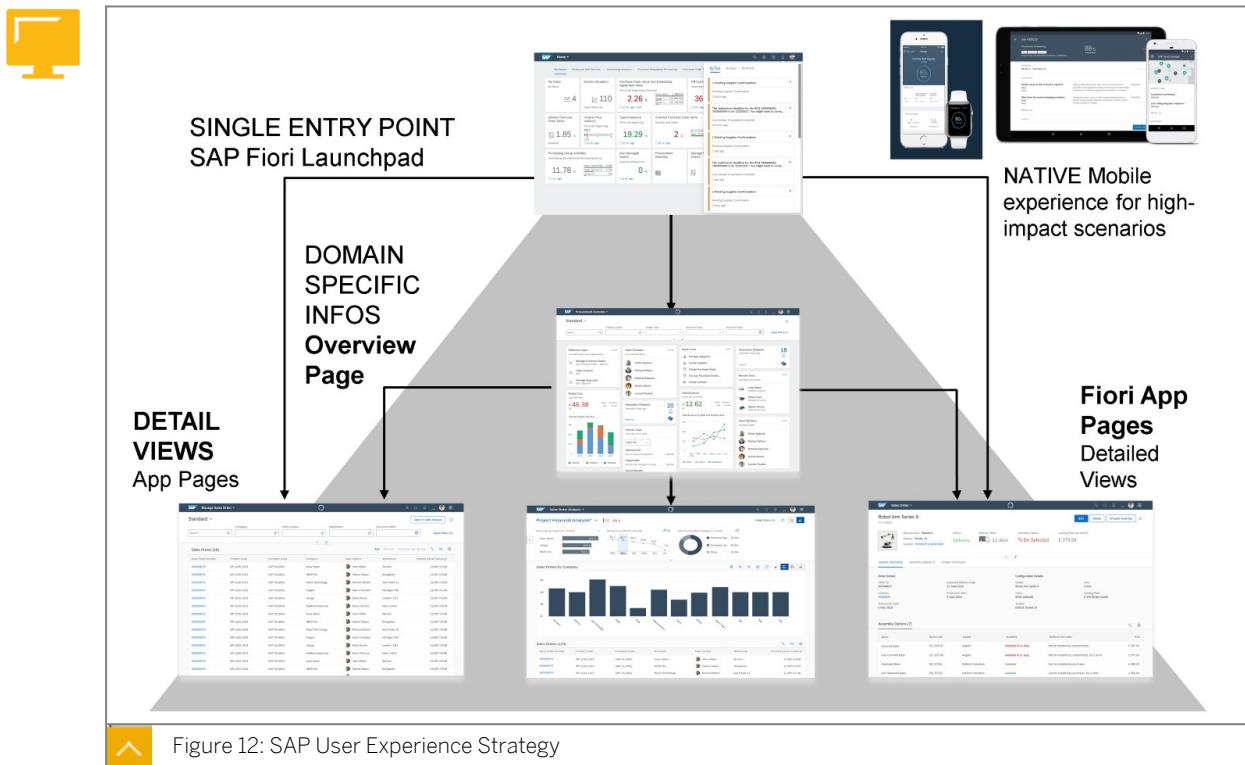


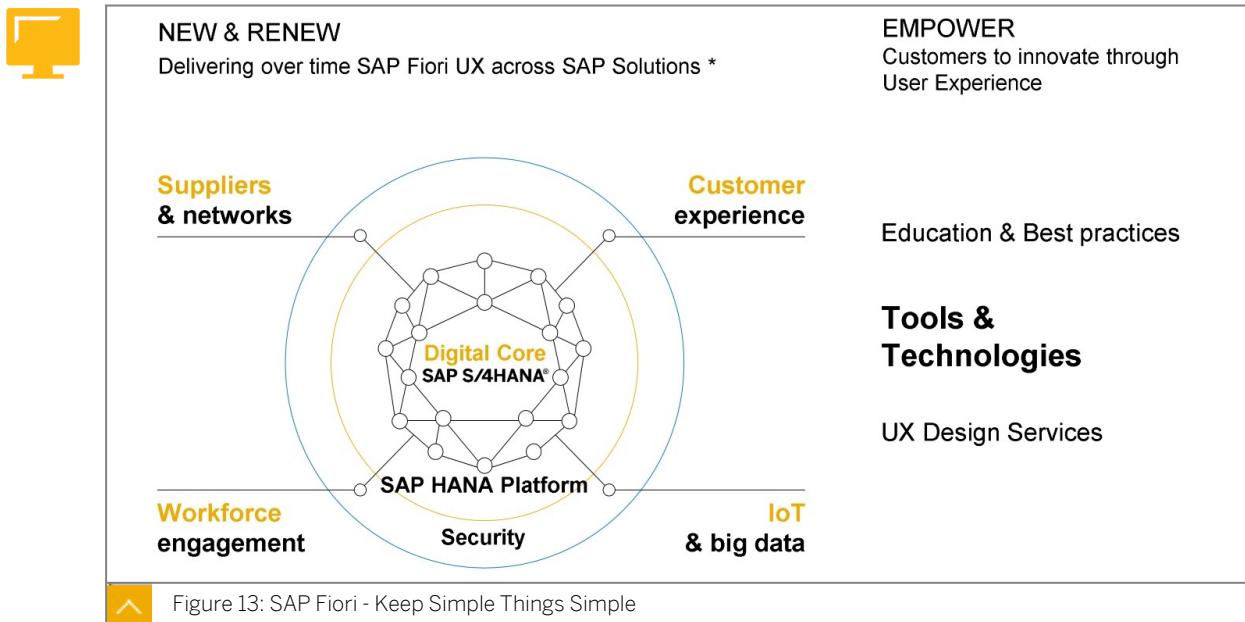
Figure 11: SAP Fiori, Concept Review

SAP applications can be provided on-premise or as cloud applications. The user gets access to these applications using a single point of entry. This single point of entry is the SAP Fiori Launchpad (FLP).

SAP Fiori for SAP S/4HANA



SAP Fiori - Keep Simple Things Simple



SAP offers a new and broad portfolio of UX design services that guide organizations into a user-centered design perspective. We help you define and execute the best UX strategy for your business using proven design methodologies like design thinking and user-centered design.

SAP Fiori is the way we renewed the most widely used scenarios.

SAP Fiori uses HTML5 and SAPUI5 technology and it can run on all devices.

Depending on the device, SAP Fiori adapts visualization to the device specifically and uses responsive design.

If you look at the tablet in this image you'll see the detail list on the left side and the main pane. Now look to the mobile phone and notice only the main pane is visible. The detail list can be accessed by swiping the phone but both panes will not fit at once. Responsive design is credited for automatically completing the look on our UI framework.

An important thing to note with SAP Fiori is it can be deployed in the customers existing landscape. SAP customers running ECC 6.0, Business Suite on HANA or S4/HANA should have a look on the SAP Fiori Deployment Options and System Landscape Recommendations at: <https://www.sap.com/documents/2018/02/f0148939-f27c-0010-82c7-eda71af511fa.html>.

The document provides the up-to-date details on what landscape should be implemented in which customer scenario.

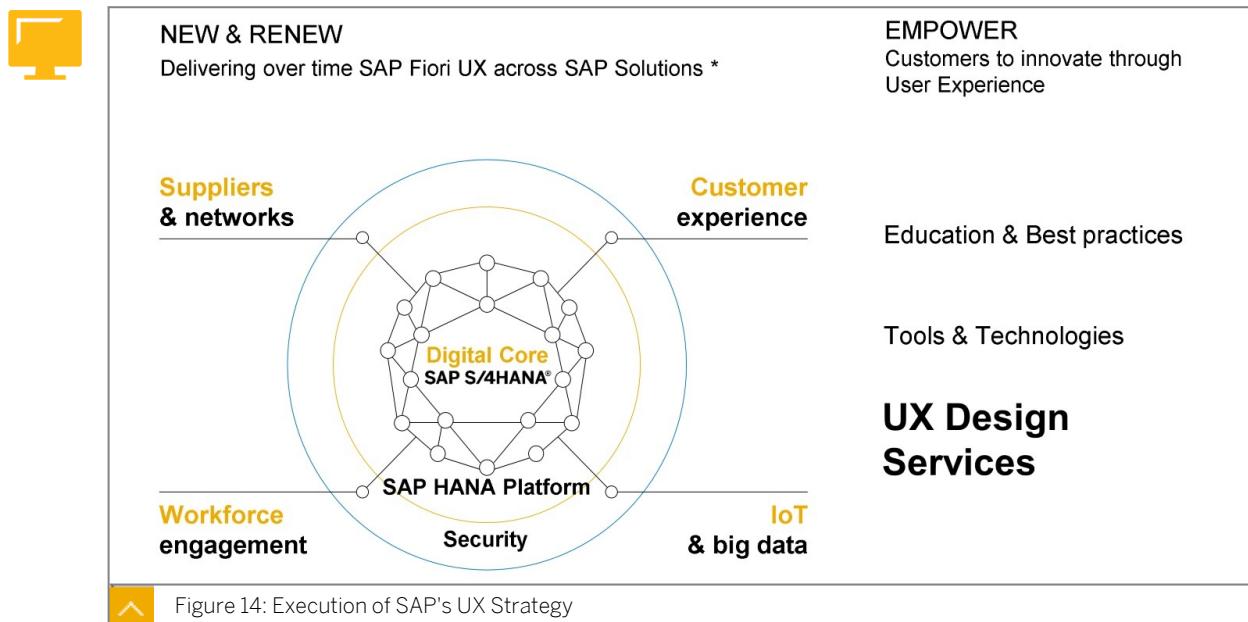
UX as a Service

SAP is committed to designing role-based applications that address the needs of our end users across all lines of business, tasks, and devices. We believe this is the key to a great user experience. But how do we guarantee a solid and consistent design for our customers and end users? The answer is **SAP's design-led development process**.

Design-led development takes advantage of proven design thinking methods to achieve an optimal user experience. The process spans the entire development lifecycle, is simple and easy to follow, and provides a solid basis for scaling design as a whole. It fosters unity between designers and developers, while ensuring that the needs of the end user are addressed at every step along the way.

You can get more details under: <https://experience.sap.com/fiori-design-web/design-led-development-process-external/>

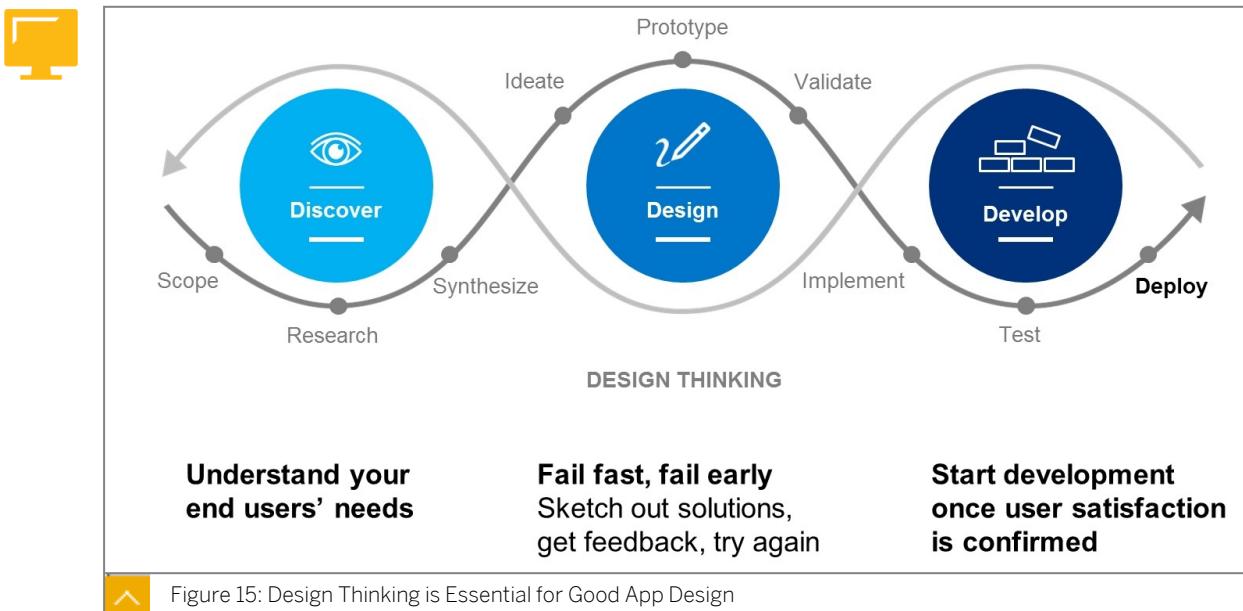
Execution of SAP's UX Strategy



SAP offers a new and broad portfolio of UX design services that guide organizations into a user centered design perspective. We help you define and execute the best UX strategy for

your business using proven design methodologies like design thinking and user-centered design.

Overview of SAP's UX Design Services



So, how to go about actually designing a good app? Well, the approach that is essential is design thinking.

And you might have heard of design thinking but design thinking is basically very simple from the idea, of course it can be quite demanding to do it right, but basically the main principle is that you use it to understand your end users' needs.

So often enough, people start looking at end users and come up immediately with a solution proposal, that's something that engineers often do, but you should start off by really listening and observing and understanding what the end users want to do, what they need to do, maybe they're even doing things without talking about it.

So understand your end users' needs. Then of course, once you have an idea of what they need, you can start thinking about how to solve it. And here, of course, you say fail fast, fail early, so basically, rather than actually programming out a really sleek or good-looking prototype, just sketch it out on paper, show it to the users, get the feedback very, very quickly, is that something that could potentially fit or not, probably get feedback, some aspects might be good, some not so good.

So sketch it out again, try again, keep iterating. And then, as you converge gradually onto the right solutions, then of course you can start investing more in actual real, clickable prototypes, for example. And then, of course, actually go to the third part, start the development. But of course, only start the development once you really are sure that user satisfaction has been confirmed. So once you really know that the user is behind what you're doing, then you actually start developing. And of course, that's a great way of making sure the users are happy and, of course, you use your time and investment most effectively. So, that's how to design a good app. But what if you have to do hundreds of apps?

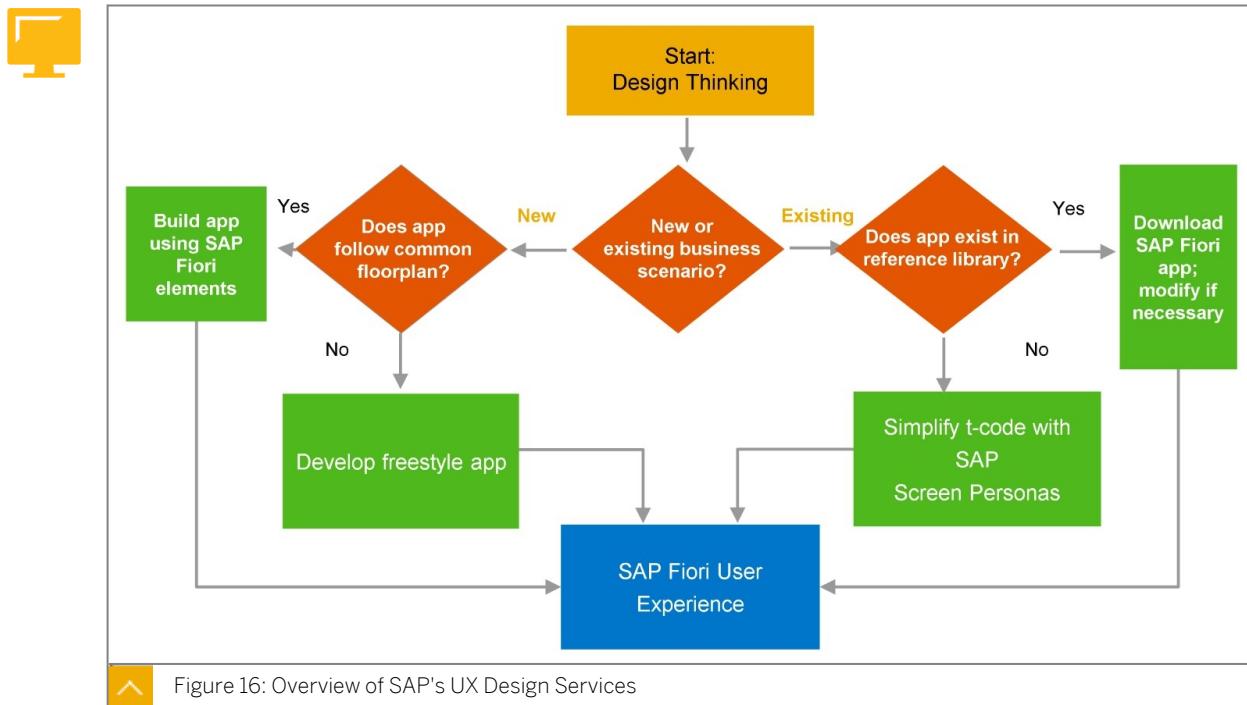
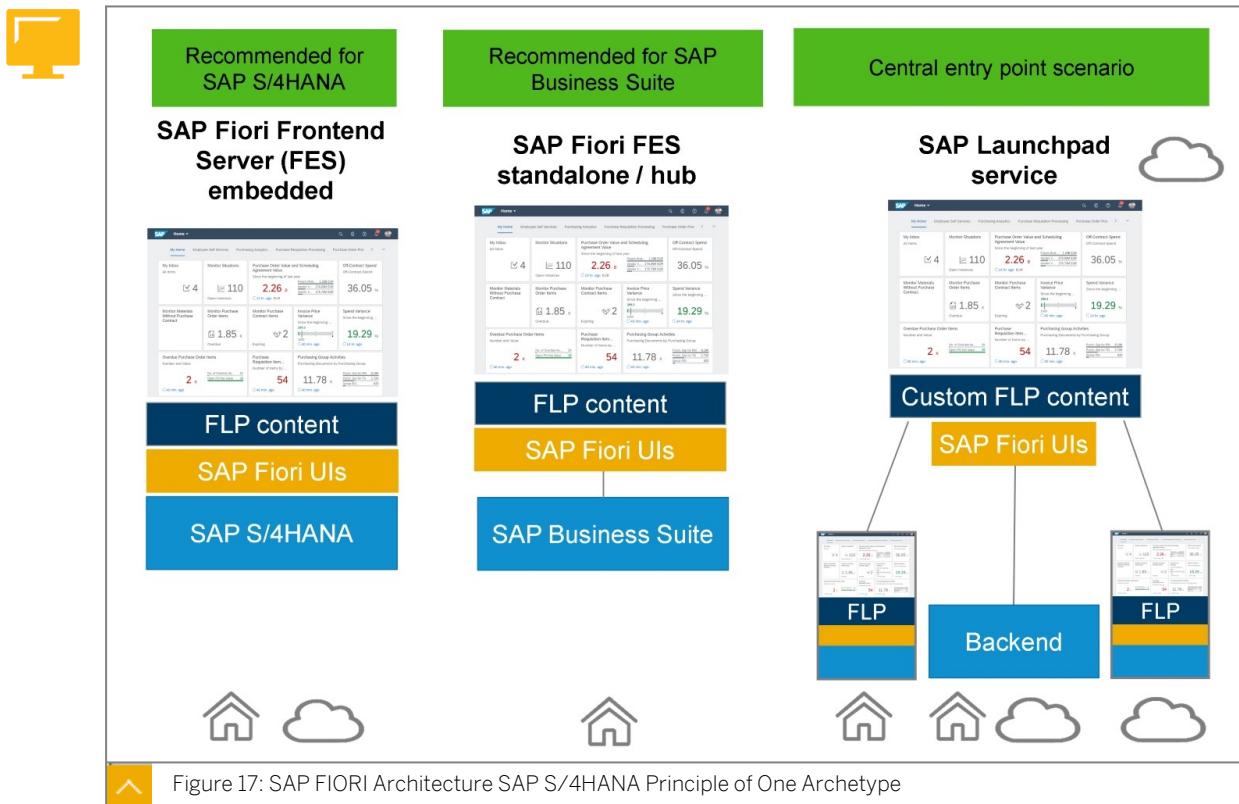


Figure 16: Overview of SAP's UX Design Services

SAP has identified several leads from our customers and they asked us for advice to understand the SAP strategy and translate it to their reality:

- Asked for services to realize: implement, adapt, and optimize the user experience of existing software.
- If you achieved results such as with screen personas or Fiori the next level is to have customers build up skills on their own or empower their organization for a user experience strategy.
- The last level is for the customer to become really innovative. This is the final goal of all customers, to be more innovative and they can achieve that by designing new products, looking for new services.
- All of this is powered by design thinking.
- In summary: the design services that SAP offers are to advise about the strategy and the value of a solid user experience.

SAP FIORI Architecture SAP S/4HANA Principle of One Archetype



SAP Fiori Architecture for SAP S/4HANA consists of only one archetype for transactional, analytical and search:



Fiori technology components

- SAP Fiori launchpad.
- Metadata driven UIs - Smart Controls & Smart Templates.

ABAP infrastructure components

- Draft Infrastructure for transactional Logic.
- SADL for CDS read access.
- Analytical Engine (embedded BW) for analytical CDS access.
- SAP Gateway for OData exposure.

CDS Views (ABAP managed)

- Uniform Business Object Modelling.
- Central repository for Metadata.

Key Enablement Tools

We just reviewed SAP Screen Personas.

Floorplan Manager which is based on ABAP Web Dynpro and allows customers to build new screens and adapt floorplan manager screens.

SAP UI5 application development tools are available so customers can build, or adapt SAP Fiori apps on their own or build their own UI5 applications.

NW business client, side panel and we offer a theme designer which is a tool that allows to adapt the branding of customer specific branding (colors, fonts, logos, change stylesheets etc.).



LESSON SUMMARY

You should now be able to:

- Describe SAP User Experience design methods for SAP Fiori-like application

Unit 1

Lesson 2

Explaining SAP User Experience Tools and Technologies

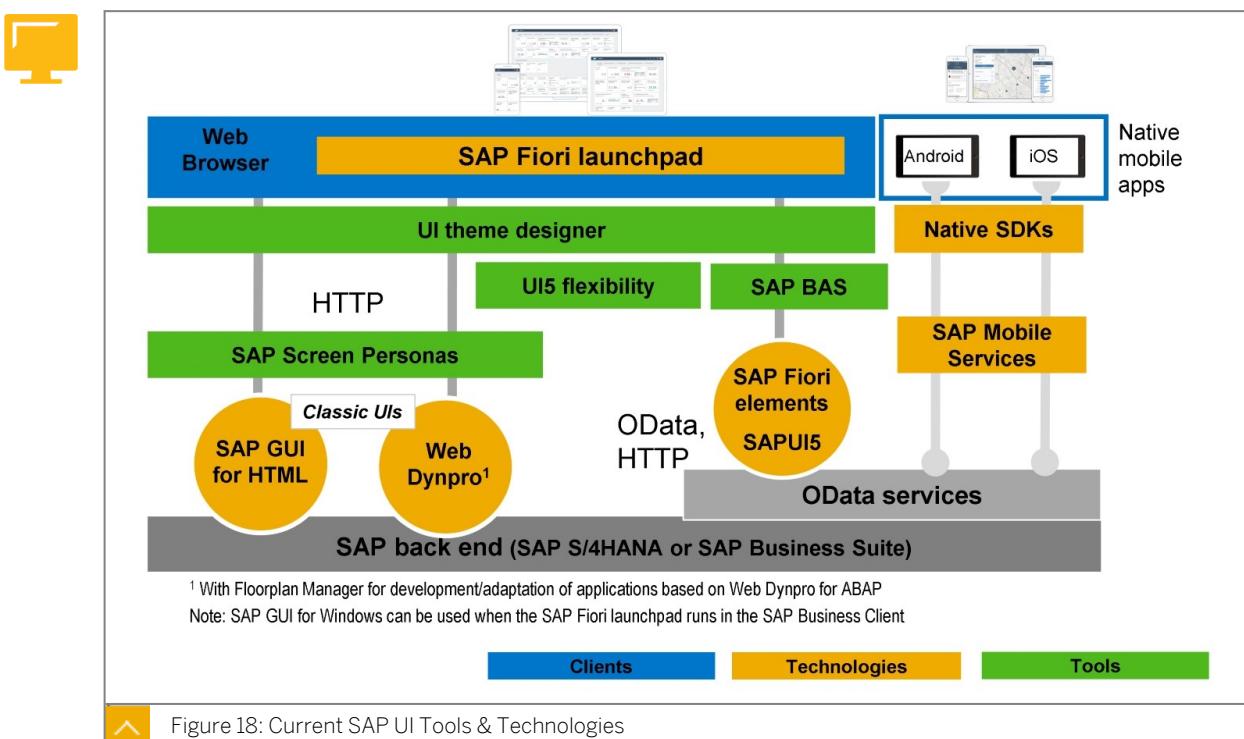


LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain how SAPUI5 aligns with the overall SAP User Experience strategy for online and mobile apps

User Experience Tools and Technologies



Overview of key UI technologies currently (as of the release of this course) offered by SAP:

SAP SAPUI5 and UI5 application development tools to change adapt or develop new applications.

Web dynpro ABAP and Floorplan manager - tool that can be used for adoption or creating new apps.

SAP dynpro - which includes SAP screen personas to optimize SAP GUI screens. Also notice the connections to the backend - dynpro screens have a close relationship to the backend, they are very interwoven. SAP Screen personas for example, is only a layer on the dynpro where you can reduce fields, merge tops, combine fields but you cannot add business functionality - this only works on the UI level.

With SAP UI5 with gateway, there is a clear separation between UI and business logic. This setup enables you to be much more flexible in the future. Gateway does not need to run on the same machine as the backend.

Theme designer running up the right side of the screen that allows for the branding of UIs.

Basic fundamental architectural change is that SAP decoupled UI and business logic which makes it easier for developers to react to change in UI technology. SAP's UI technologies will continue to evolve and improve into new UI technologies as they make it to the marketplace in the future.

Goal of the SAP SAPUI5 Framework

The following list shows the goals of the SAP SAPUI5 Framework:



- The UI development toolkit for HTML5 (SAP SAPUI5) is a user interface technology that is used to build and adapt client applications.
- The SAPUI5 runtime is a client-side HTML5 rendering library with a rich set of standard and extension controls.
- It provides a lightweight programming model for desktop and mobile applications.
- Based on JavaScript, it supports RIA like client-side features:
 - SAPUI5 complies with OpenAjax and can be used together with standard JavaScript libraries.
- It supports CSS3, which allows you to adapt themes to your company's branding in an effective manner.
- It is based on an extensibility concept regarding custom controls.

SAPUI5 is a UI technology that provides everything you need to build enterprise-ready web apps. It comes with all main SAP platforms but can also be used outside the SAP ecosystem because a large part of SAPUI5 had been open sourced with OpenUI5.



- SAPUI5 is a client UI technology based on JavaScript, CSS and HTML5
- SAPUI5 applications run in a browser
- Depending on the device the application is to run on (mobile, tablet or desktop PC), you use different UI libraries

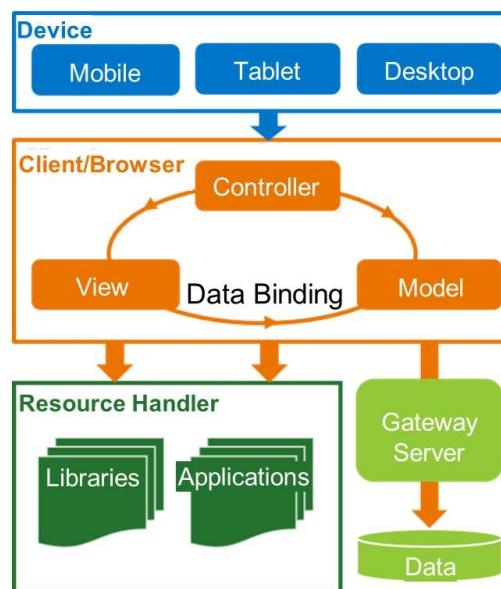


Figure 19: SAP SAPUI5 Architecture

Servers come into play for deploying your applications, storing the SAPUI5 libraries and connecting to a database. Depending on the environment in which SAPUI5 is used, the libraries or your applications are stored on an SAP NetWeaver Application Server or an SAP Business Technology Platform (SAP BTP), for instance. The preferred way to access business data for your application is using the OData model through a SAP NetWeaver Gateway.

When users access an SAPUI5 application from their device, a request is sent to the respective server to load the application into the browser. The view accesses the relevant libraries. Usually the model is also instantiated and business data is fetched from the database.

SAPUI5 Provides Predefined Models

SAPUI5 provides predefined models. These are the benefits:



The JSON model can be used to bind controls to JavaScript object data, which is usually serialized in the JSON format:

- The JSON model is a client-side model and, therefore, intended for small datasets, which are completely available on the client.
- The JSON model supports two-way binding.

The XML model is a client-side model intended for small datasets, which are completely available on the client:

- The XMLModel does not contain mechanisms for server-based paging or loading of deltas.

The Resource model is designed to handle data in resource bundles, mainly to provide texts in different languages.

The OData model enables binding of controls to data from OData services:

- The OData model is a server-side model: the dataset is only available on the server and the client only knows the currently visible rows and fields.
- This also means that sorting and filtering on the client is not possible.
- For this, the client has to send a request to the server.
- The OData model currently supports OData version 2.0.

SAPUI5 supports different types of models. A model in the Model View Controller concept holds the data and provides methods to retrieve the data from the database and to set and update data.



LESSON SUMMARY

You should now be able to:

- Explain how SAPUI5 aligns with the overall SAP User Experience strategy for online and mobile apps

Unit 1

Lesson 3

Describing SAP User Experience Use Case for Building Fiori-like Apps



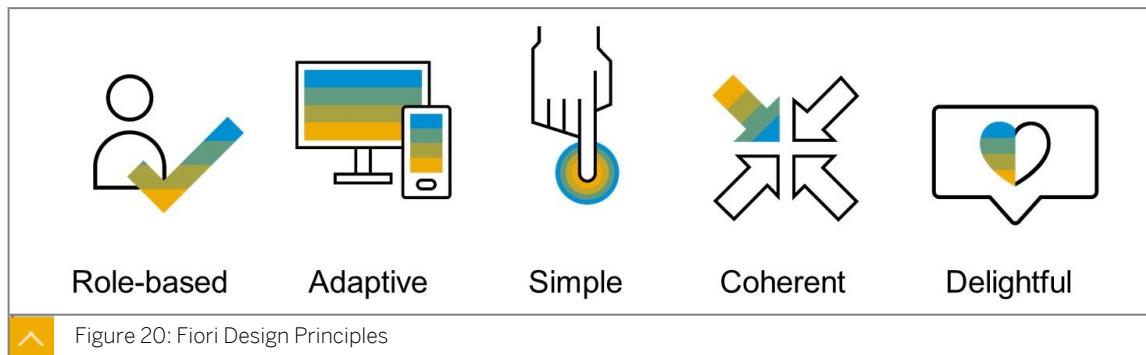
LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe Use cases for Fiori-Like Apps

SAP User Experience Use Case for Building Fiori-like Apps

Fiori Design Principles



SAP Fiori Design Principles

Role based:

Designed for you, your needs, and how you work.

Adaptive:

Adapts to multiple use cases and devices.

Simple:

Only what is necessary.

Coherent:

Provides one fluid user experience.

Delightful:

Makes an emotional connection.

Use Case - Design Process for Fiori-Like Apps Using SAPUI5

The Design Process for Fiori-Like Apps using SAPUI5 is:



1. Observe how users are currently using the application to complete a business process.
2. Facilitate a Design Thinking workshop with users and designers.

3. Create wireframes to represent the agreed upon use case.
4. Validate the wireframes with the users to ensure that their use case was understood correctly.
5. Iterate through the wireframe scenarios & user validation until agreement is met.
6. Make proposal of finalized wireframes to all stakeholders.
7. Begin the development & test cycles of the SAPUI5 application.



Note:

When implementing SAP Fiori apps you have to go new ways.

Use Case - Financial Dashboard Outcome

Business Problem: standard implementation of a financial dashboard was cumbersome and difficult to use by users.

Solution: Via the user experience driven design process, a user-centric financial dashboard cockpit was developed that focused on the core 4 KPIs across all users.

Online SAP SAPUI5 Application Demonstrations

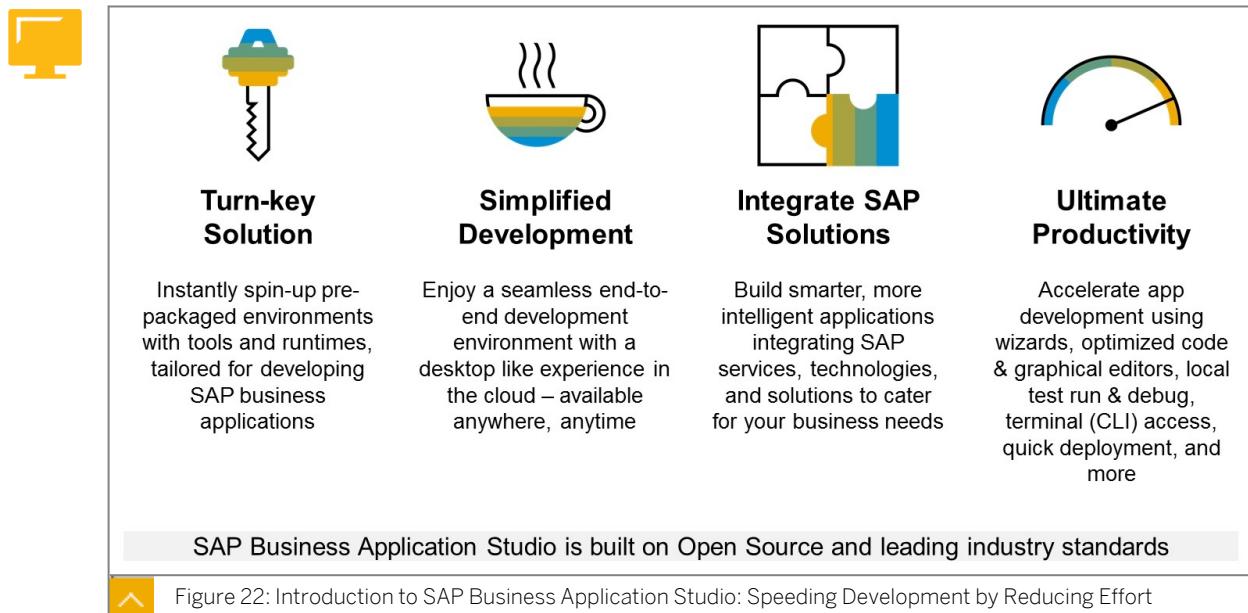


The figure shows the Demo entry page.

Take a moment to view the sample application that SAP has made available online at: <https://ui5.sap.com/#/demoapps>

SAP Business Application Studio Introduction

Introduction to SAP Business Application Studio — Speeding Development by Reducing Effort



Delivering a web-based development tool designed to support the End-to-End application lifecycle for SAPUI5:

Prototyping

Developing

Packaging

Deploying

Extending

SAP Business Application Studio is a browser-based tool that empowers developers, business experts and designers to build new user interfaces that work with SAP applications. SAP Business Application Studio intends to simplify the end-to-end application lifecycle: prototyping, development, packaging, deployment, and customer extensions for SAPUI5 applications.

Introduction to SAP Business Application Studio: Product Benefits

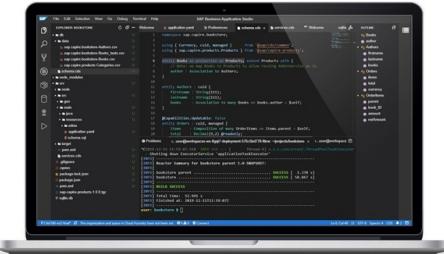


SAP Business Application Studio is a **modern** development environment, tailored for **efficient** development of business applications for the **Intelligent Enterprise**



Key capabilities

- Turn key solutions suited for various business scenarios providing isolated browser based-development environment with pre-packaged tools and run times
- Desktop-like experience in the cloud with terminal (CLI) access and local test run, providing more control over the environment
- Consistent experience with leading IDEs using optimized code editors, debuggers, Git, and more



Benefits

- Easily develop and extend SAP solutions, seamlessly integrating SAP services, technologies, and solutions
- End-to-end development environment supporting key scenarios such as: SAP Fiori, SAP S/4HANA extensions with CAP, Workflow and Mobile
- Accelerate time-to-market with high-productivity development tools e.g. wizards and templates, graphical editors, quick deployment, and more
- Available anytime, anywhere on SAP's multi-cloud environment
- Simplifies and saves time in setting up the development environment



Figure 23: SAP Business Application Studio

Highly efficient development environment:

- Source code editor with SAPUI5-specific code completion (configurable for any UI5 version).
- Allows new project creation based on SAPUI5 or Fiori templates.
- Extending and adapting existing Fiori/UI5 application easily.
- WYSIWYG tooling which is a graphical environment to build the UI and to modify it later.
- Simple Project persistency (GIT).
- Mock data support for:
 - Testing Decoupling frontend development from backend.
- Instant preview in browser.
- Comprehensive search capabilities.

Introduction to SAP Business Application Studio: Key Features II - Templates

SAP Fiori Dev Space

- IDE with language services for SAPUI5, JS, XML
- Layout editor and SAP Fiori tools
- Rich set of templates: SAP Fiori elements & freestyle
- Local test: multiple SAPUI5 versions, unit and integration tests, mock
- Env: CLI tools, SAPUI5 build, deploy to ABAP Repository, CF, ...
- Service catalog

SAP Cloud Business Application Dev Space

- IDE with language services for Java, NodeJS, XML
- Layout editor and SAP Fiori tools *
- Rich set of templates: CAP, SAP Fiori elements* & freestyle
- Tools: CDS tools, MTA tools, Spring-Boot tools, SQL tools, REST client and SAP HANA tools*
- Local test: Run service, in-memory db, local SQLite db, REST client, mock, ...
- Env: CLI tools, builders,...
- Service and Events* catalog

Mobile Dev Space

- Basic IDE with auto-completion, validation, and more
- Templates for: Mobile Dev Kit, Mobile Cards, Mobile Backend Tool
- Local test: Mobile runtime
- Env: CLI tools, deploy to Mobile Services
- Service catalog

This is the current state of planning and may be changed by SAP at any time.

+ Create custom Dev Space templates for developers*

* Future Innovation

Figure 24: Introduction to SAP Business Application Studio: Key Features II - Templates

The dev-spaces are at the heart of SAP Business Application Studio. Let's take a closer look at this concept:

- There are various dev-space types. Each dev-space type fits an Intelligent Enterprise development use case.
- This slide presents additional examples of dev-space types. Each of these dev-space types comes prepackaged with the tools and runtimes relevant for its scenario. If we take for example the SAP Fiori dev space.
 - SAP Fiori:
 - Open source IDE with editors and language services
 - SAP Fiori templates
 - Local test tools
 - SAPUI5 build and the ability to deploy to the actual target runtime directly from SAP AppStudio
 - CAP
 - Mobile

In the future there will be the option to customize the dev-space as well as share, so all developers in the team will use the same set of development tools.

SAP Business Application Studio Plugins

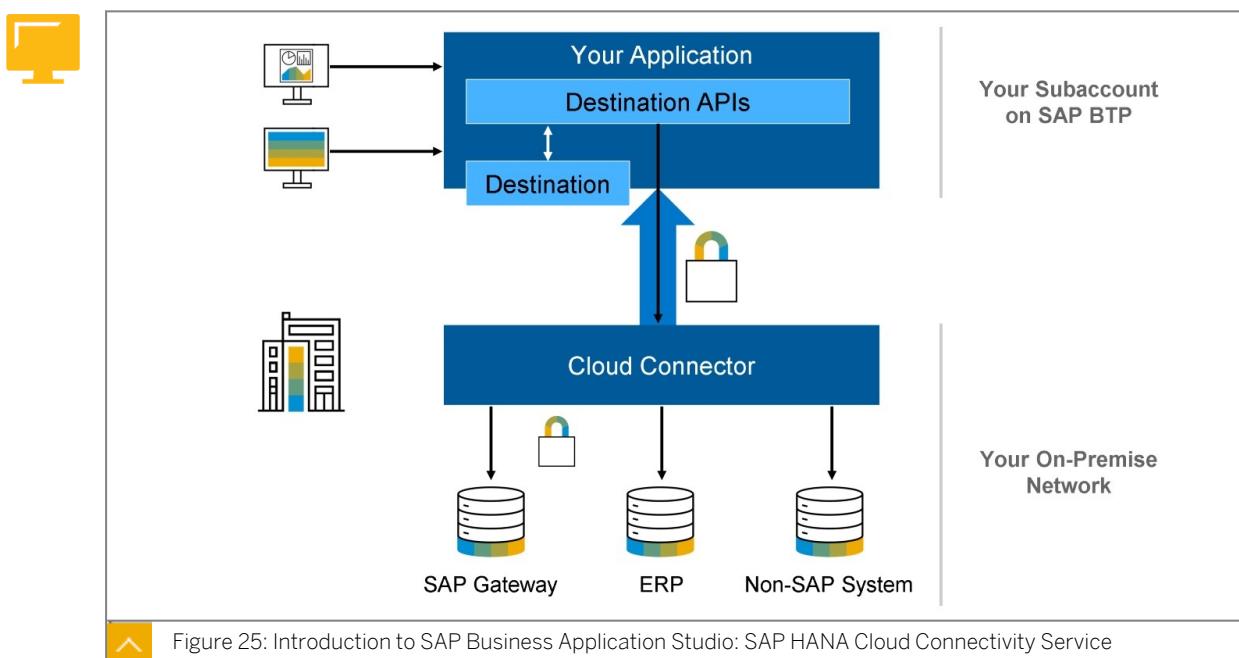
SAP offers out-of-the-box plug-ins and templates that leverage the SAP Business Application Studio modular and extensible framework:

- Hybrid Application Toolkit
- Create and deploy Apache Cordova hybrid apps
- Templates, code completion, testing, build and deploy
- OData Model Editor
- Define OData models
- Create OData models from scratch or import from file
- Edit OData models quickly using code assist, auto complete and beautify
- Validation of the code on the fly

UI Annotations modeler:

- Create new UI annotations
- Edit existing annotations

Introduction to SAP Business Application Studio: SAP HANA Cloud Connectivity Service



If you want to connect to an ABAP system, you need to specify this in the destination details.

Let's suppose you want to connect to the GM6 ABAP system (<https://wdflbmt2291:55080>) in order to consume some OData for creating a new Fiori app or for extending some existing ones.

Furthermore you want also to execute some other SAPUI5 applications. We need to pass this information to the SAP Business Application Studio and the way to do it is through the destination's additional property WebIDEUsage; we need to specify in this property the three usages we want to have with this destination:

- odata_abap (for consuming OData),
- dev_abap (for extending existing Fiori apps) and

- ui5_execute_abap (for executing SAPUI5 apps).

Furthermore we have to set the property HTML5.DynamicDestination and WebIDEEnabled to true. Please refer to the official SAP Business Application Studio documentation to get more information on this.



LESSON SUMMARY

You should now be able to:

- Describe Use cases for Fiori-Like Apps

Learning Assessment

1. What impact does UX have on monetary values?

Choose the correct answers.

- A Increases user satisfaction.
- B Provides productivity gains and increases data quality.
- C Strengthens relationships with customers.
- D Provides training cost savings.
- E Reduces the number of change requests and user errors.

2. What is the principle of SAP UX strategy?

Choose the correct answer.

- A Design Strategy
- B New, Renew, Enablement
- C New, Renew, Empower
- D Architecture and Technology
- E SAP Screen Personas

3. What impact does SAP Fiori have on business?

Choose the correct answers.

- A Digitalization
- B Simplification
- C Support the web and open standards
- D Provides a user-centered approach
- E Leads to re-imagination of processes

4. Which of the following are the current SAP UI Tools?

Choose the correct answers.

- A SAPUI5 Application Development Tools
- B SAP Screen Personas
- C SAP NetWeaver Portal
- D Flexible UI Designer
- E SAP NetWeaver Gateway

5. What are the current UI Technologies of SAP?

Choose the correct answers.

- A Business Server Pages
- B SAPUI5
- C Java Server Pages
- D Web Dynpro ABAP / Floorplan Manager
- E Dynpro

6. What are the goals of the SAPUI5 framework?

Choose the correct answers.

- A Provide a user interface technology for building and adapting client applications.
- B Provide a user interface technology for building and adapting server-based applications.
- C Provide a lightweight programming model for desktop only applications.
- D Provide an extensible framework for building desktop and mobile applications.

7. What are the SAP Fiori principles?

Choose the correct answers.

- A Role-based
- B Adaptive
- C Creative
- D Coherent
- E Complex

8. Which of the following steps are part of the discover phase in the DLD?

Choose the correct answers.

- A** Scope
- B** Test
- C** Implement
- D** Research
- E** Synthesize

9. Which of the following steps are part of the design phase in the DLD?

Choose the correct answers.

- A** Test
- B** Validate
- C** Prototype
- D** Scope
- E** Ideate

Learning Assessment - Answers

1. What impact does UX have on monetary values?

Choose the correct answers.

- A Increases user satisfaction.
- B Provides productivity gains and increases data quality.
- C Strengthens relationships with customers.
- D Provides training cost savings.
- E Reduces the number of change requests and user errors.

Correct. UX provides productivity gains and increased data quality, it saves training costs, and reduces the number of change requests and user errors.

2. What is the principle of SAP UX strategy?

Choose the correct answer.

- A Design Strategy
- B New, Renew, Enablement
- C New, Renew, Empower
- D Architecture and Technology
- E SAP Screen Personas

Correct. The principle of SAP UX strategy is: New, Renew, Empower.

3. What impact does SAP Fiori have on business?

Choose the correct answers.

- A Digitalization
- B Simplification
- C Support the web and open standards
- D Provides a user-centered approach
- E Leads to re-imagination of processes

Correct. SAP Fiori leads us to re-imagine processes, provides application simplification, and supports digitalization in the company.

4. Which of the following are the current SAP UI Tools?

Choose the correct answers.

- A SAPUI5 Application Development Tools
- B SAP Screen Personas
- C SAP NetWeaver Portal
- D Flexible UI Designer
- E SAP NetWeaver Gateway

Correct. Currently, the three SAP UI tools are: 1) SAPUI5 to implement SAP Fiori applications, 2) SAP Screen Personas to reduce the complexity of existing Dynpro and Web Dynpro ABAP applications, and 3) Flexible UI Designer.

5. What are the current UI Technologies of SAP?

Choose the correct answers.

- A Business Server Pages
- B SAPUI5
- C Java Server Pages
- D Web Dynpro ABAP / Floorplan Manager
- E Dynpro

Correct. SAPUI5, Web Dynpro ABAP, and Dynpros are the current UI technologies of SAP.

6. What are the goals of the SAPUI5 framework?

Choose the correct answers.

- A Provide a user interface technology for building and adapting client applications.
- B Provide a user interface technology for building and adapting server-based applications.
- C Provide a lightweight programming model for desktop only applications.
- D Provide an extensible framework for building desktop and mobile applications.

Correct. SAPUI5 is a user interface technology for building and adapting client applications. It provides an extensible framework for building responsive applications for desktop and mobile devices.

7. What are the SAP Fiori principles?

Choose the correct answers.

- A Role-based
- B Adaptive
- C Creative
- D Coherent
- E Complex

Correct. The SAP Fiori principles are: role-based, coherent, and adaptive.

8. Which of the following steps are part of the discover phase in the DLD?

Choose the correct answers.

- A Scope
- B Test
- C Implement
- D Research
- E Synthesize

Correct. The discover phase consists of scoping to understand the problem spaces, research to understand the current state, and synthesize to describe the results of the research phase.

9. Which of the following steps are part of the design phase in the DLD?

Choose the correct answers.

- A Test
- B Validate
- C Prototype
- D Scope
- E Ideate

Correct. Ideate, prototype, and validate are part of the design phase in the DLD.

UNIT 2

MVC Review and Advanced UI Controls

Lesson 1

Performing an MVC Architecture Review

39

Lesson 2

Binding Data to a UI5 Control

45

Lesson 3

Describing Best Practices for SAPUI5 Applications

55

Lesson 4

Implementing App Navigation

65

Lesson 5

Implementing a Full-screen Application

73

Lesson 6

Implementing a Master-Detail Application

93

Lesson 7

Working with Messages

117

Lesson 8

Describing Key Responsive Design Controls

125

Lesson 9

Extending Standard Controls

137

Lesson 10

Describing Custom Controls

145

Lesson 11

Creating Control and Component Libraries

151

Lesson 12

Lesson 13

UNIT OBJECTIVES

- Describe MVC and application architecture best practices
- Bind Data to a UI5 Control
- Describe SAPUI5 best practices
- Implement in App Navigation and deep Linking
- Implement a full-screen application
- Implement a Master-Detail-Application
- Work with Messages
- Describe Key Responsive Design Controls
- Describe how to leverage the features of the standard controls in the SAPUI5 framework
- Describe how to implement custom controls using the SAPUI5 framework
- Create a control or component library
- Implement unit tests
- Implement OPA tests

Performing an MVC Architecture Review



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe MVC and application architecture best practices

MVC Architecture Review

Scenario

As a Business Process owner you are requested to permanently improve the user experience in your area of responsibility.

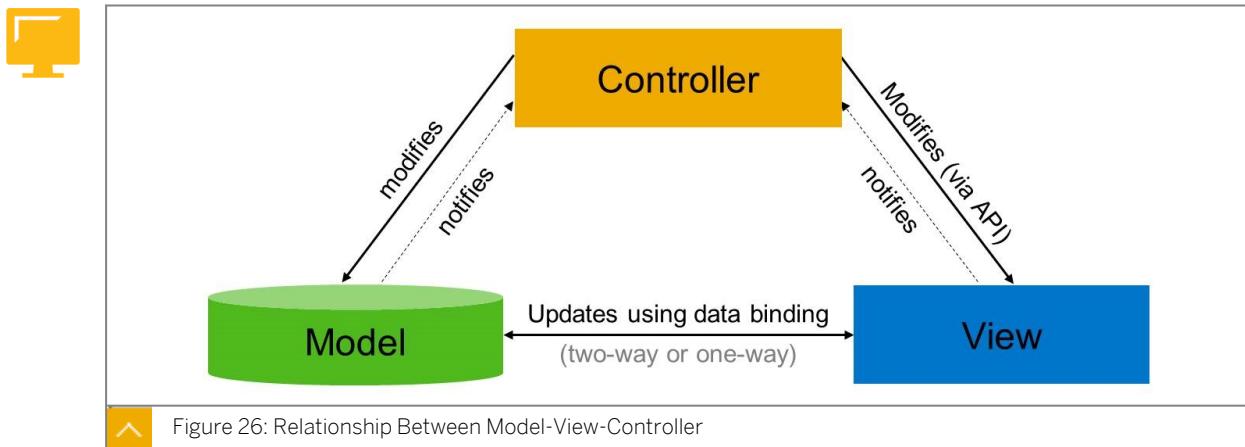
Creating user interfaces is a critical aspect in the user experience. In this training, you will learn how to develop SAPUI5 user interfaces to induce a great user experience.

What is Model-View-Controller (MVC)?

The Model-View-Controller (MVC) is:



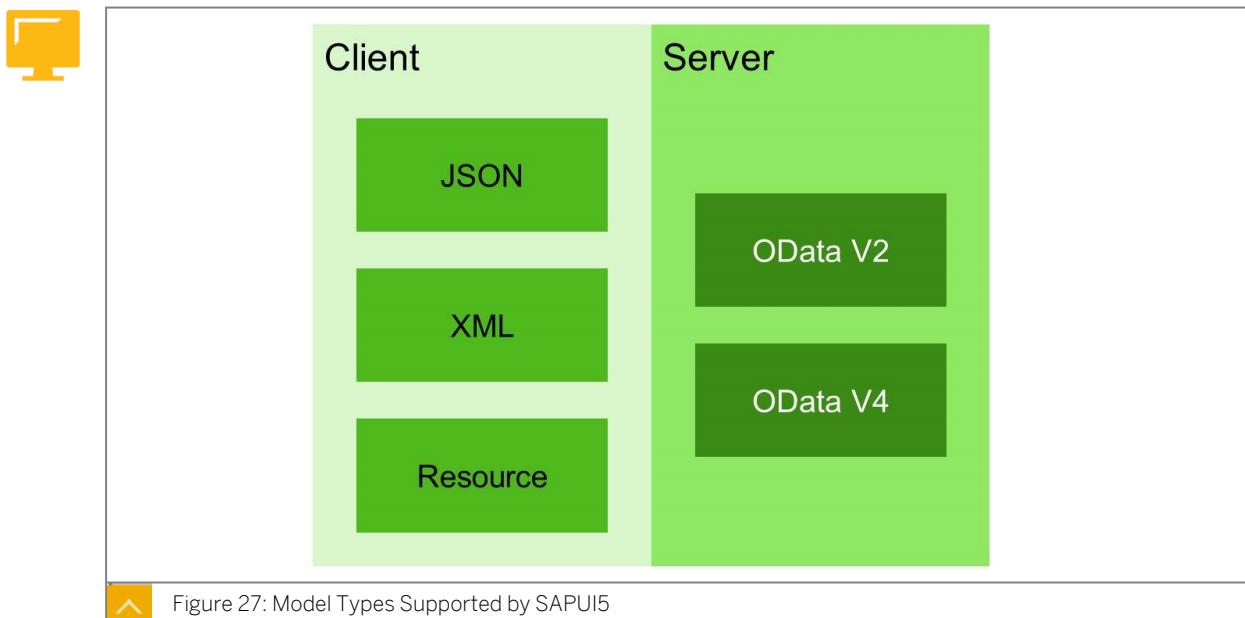
- Model-View-Controller (MVC) is a software architecture design pattern.
- The Model represents the data of the application:
 - Represents your enterprise data.
 - Includes business rules.
 - Where most of the processing takes place.
- The View is updated by the model and presents the data to the user:
 - No real processing happens in the View.
 - Allows the data to be displayed and/or manipulated by the user.
- The Controller is used by the user to manipulate the data in the model:
 - Does not perform processing.
 - Determines which model components to invoke depending on the user's action.
 - Can also perform data formatting for the view.



The Model View Controller (MVC) concept is used in SAPUI5 to separate the representation of information from the user interaction. This separation facilitates development and the changing of parts independently.

Model, view, and controller are assigned the following roles:

- The View is responsible for defining and rendering the UI.
- The Model manages the application data.
- The Controller reacts to view events and user interaction by modifying the view and model.



The following model types are supported by SAPUI5:

- **JSONModel**
 - Client-side model intended for small datasets which are completely available on the client.
- **XMLModel**

- Client-side model intended for small datasets which are completely available on the client.
- No mechanism for server-based paging or loading of deltas.
- ResourceModel
 - For handling data in resource models.
 - Mainly for localization.
- ODataModel
 - Server-side model where the dataset is only available on the server and the client only knows of the currently visible rows/fields.
 - Sorting and filtering are server-side activities.
 - The OData model currently supports the following OData versions:
 - OData V2.
 - OData V4 (limited feature scope).

MVC and Reusability

The following list shows the basics of MVC and Reusability:



- By separating the data and the display you are limiting the duplication of code.
- Since the model returns the raw data (without formatting) it could be reused by several different types of user interfaces:
 - One application might create a HTML interface;
 - One application might create an interface using Java or some other language;
- Separating the model from the display also allows you to change the data layer without impacting the other components.
 - For example, if you changed the underlying database from Oracle to Microsoft SQL Server, you need only change the model code.
 - The view does not care from which database the data is coming.

MVC and Delegation of Work

The following list shows the properties of MVC and delegation of work:



- Using the MVC design pattern allows companies to delegate work as they are building applications:
 - UI designers can build the HTML and CSS (View).
 - Programmers can create the JavaScript (Controller).
 - People with skills in back-end systems, like SAP systems or databases can define the model.
- Allows people with certain skills to focus only on the pieces they are responsible for:

- For example, people who are creating the UI do not have to wade through JavaScript code.
- Speeds application development.

Downside of MVC

The following list shows the downsides of MVC:



- Using MVC is not always easy.
- Requires much thought and planning:
 - Need to spend a lot of time thinking about how the different components will interact.
 - May require coordination between different developers or different teams.
- All of the layers makes debugging more difficult:
 - However, once tested, you have a reusable component.
- More components to manage.

Sample Project Structure - MVC and UI5



- The project structure of a standard SAPUI5 project is shown here
- The view folder contains the views and fragments
 - Views and fragments are written in XML
- The controller folder contains the controller objects
 - Controllers are written using JavaScript
- Component.js has code that encapsulates the entire app (has startup code)
- The test folder contains everything for Unit and OPA tests
- manifest.json is the Application descriptor that contains the metadata description of the application
- index.html has only minimal, bootstrap code
 - Loads SAPUI5 libraries
 - Instantiates the component

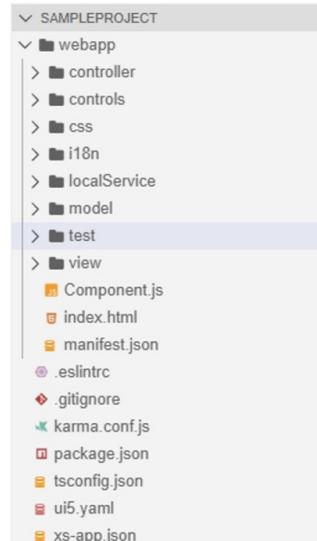


Figure 28: Sample Project Structure - MVC and UI5

An SAPUI5 project is structured by using different folders. Each folder defines by its name what kind of UI5 artefacts are part of the folder.

The following predefined view types are available:

- XML view (file or string in XML format); the XMLView type supports a mix of XML and plain HTML.
- JSON view (file or string in JSON format).
- JS view, constructed in a traditional manner.
- HTML view (file or string in HTML format).

- SAP recommends to use the XML view type when ever possible.

Views in SAPUI5 can be XML, JavaScript, JSON, or HTML

- XML views are used for the exercises and most SAP delivered Fiori apps.
- SAP BTP creates views and controllers at the same time and gives them a basic feature.



```

Flights.view.xml ×
1  <mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Flights"
2    xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:l="sap.ui.layout"
3    xmlns:cust="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.control"
4    <Page navButtonPress="OnNavBack" showNavButton="true" title="{AirLineName}">
5      <content>
6        <l:VerticalLayout>
7          <ObjectHeader title="{AirLineName}" number="" numberUnit="{LocalCurrencyCode}" titleHref="{URL}">
8            <attributes>
9              <ObjectAttribute text="{AirlineID}"></ObjectAttribute>
10             <ObjectAttribute text="{URL}"></ObjectAttribute>
11           </attributes>
12         </ObjectHeader>
13         <Table id="idFlights" items="{ path: 'CarrierFlights', sorter: { path: 'AirlineID' } }">
14           mode="SingleSelectMaster" growing="true" growingThreshold="10">
15           <headerToolbar>
16             <Toolbar>
17               <Title text="Flights" level="H2"/>
18             </Toolbar>
19           </headerToolbar>
20           <columns>
21             <Column width="12em">
22               <Text text="{i18n>carriername}"></Text>
23             </Column>
24             <Column minScreenWidth="Tablet" demandPopin="true">
25               <Text text="{i18n>flightno}"></Text>
26             </Column>
27             <Column minScreenWidth="Tablet" demandPopin="true">
28               <Text text="{i18n>flightdate}"></Text>
29             </Column>
30             <Column minScreenWidth="Tablet" demandPopin="true">
31               <Text text="{i18n>seatmax}"></Text>
32             </Column>
33             <Column hAlign="Right">
34               <Text text="{i18n>seatoc}"></Text>
35             </Column>
36             <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
37               <Text text="{i18n>planeInfo}"></Text>
38             </Column>
39             <Column minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
40               <Text text="{i18n>bookaction}"></Text>
41             </Column>
42           </columns>
43           <items>
44             <ColumnListItem>
45               <cells>

```

Figure 29: Excerpt from a SAPUI5 View

Important things to notice:

- Written using XML.
- XML namespaces are defined at the top.
- SAPUI5 control ObjectHeader is used.
- OData data binding is used to fill properties such as the title of the page.
- {i18n>carriername} is reading from the resource model called i18n and reading the property carriername. In this case this is used for internationalization.



```

Flights.controller.js x
1 sap.ui.define([
2   "sap/ui/core/mvc/Controller",
3   "student00/com/sap/training/ux402/fullscreen/ux402_fullscreen/control/HoverButton",
4   "sap/m/MessageToast",
5   "student00/com/sap/training/ux402/fullscreen/ux402_fullscreen/control/PlaneInfo"
6 ],
7   function (Controller, HoverButton, MessageToast, PlaneInfo) {
8     "use strict";
9
10    return Controller.extend("student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Flights", {
11      onInit: function () { ... }
12    },
13
14    getRouter: function() { ... }
15  ),
16
17    _onObjectMatched: function(oEvent) { ... }
18  ),
19
20
21    _onBindingChange: function() { ... }
22  ),
23
24
25    onNavBack: function() { ... }
26  ),
27
28
29    onHover: function(evt) { ... }
30  );
31
32 });
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68 });

```

Figure 30: Excerpt from a SAPUI5 Controller

Important things to notice:

- Written using JavaScript
- The controller is implemented using the Asynchronous Module Definition (AMD)
- The controller declaration at line 9 extends the base SAPUI5 Controller
- As a rule of thumb private functions are starting with an underscore
- Functions that will be called from the View start at line 48 and 52



LESSON SUMMARY

You should now be able to:

- Describe MVC and application architecture best practices

Unit 2

Lesson 2

Binding Data to a UI5 Control



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Bind Data to a UI5 Control

Data Binding to a SAP UI5 Control

Data Binding is:



- Used to bind two data sources together, and keeping them in sync.
- If implemented properly, changes in one data source are automatically reflected in the other data source.

To implement, you need:



- The Model:
 - Holds the data and has the methods to retrieve and display the data.
 - Also has a method for creating bindings to the data.
- The Data Binding:
 - Created when the method of the model is called.
 - Contains the binding info and provides an event that is fired when data is changed.
 - An element can listen for this event and update its visualization when new data exists.

The purpose of data binding in the UI is to separate the definition of the user interface (View), the data visualized by the application (Model), and the code for the business logic for processing the data (Controller). The separation has the following advantages. It provides better readability, maintainability, and extensibility and it allows you to change the view without touching the underlying business logic and to define several views of the same data.

Binding Modes

Basics about Binding Modes:



- The UI uses data binding to bind controls to the model so the controls are updated automatically when the model data changes.
 - One-way binding – from the model to the view.
- Data binding also used when changes in the control need to update data in the model, for example, the user enters some new data.

- Two-way binding – from the model to the view and the view to the model.
- There is also a one-time binding mode:
 - Binding from the model to the view but only once.
- Code for changing the binding mode of an OData model type:
`oModel.setDefaultBindingMode(sap.ui.model.BindingMode.TwoWay);`

The binding mode defines how the data sources are bound. The different model implementations require specific binding modes. The resource model, for example, only supports one-time binding, that is, a binding from the model to the view.

SAPUI5 provides the following binding modes:

- One Way: One-way binding means a binding from the model to the view; value changes in the model update all corresponding bindings and the view.
- Two Way: Two-way binding means a binding from the model to the view and from the view to the model; value changes in the model and in the view update all corresponding bindings and the view and model, respectively.

One Time: One-time binding means from model to view once.

Binding Mode Support by Model Type

Supported binding modes, support by model type are:



Table 1: Supported binding modes by model type

Model	One-way	Two-way	One-time
Resource model	—	—	X
JSON model	X	X	X
XML model	X	X	X
OData model	X	X	X

Default binding modes, support by model type are:



Table 2: Default binding mode by model type

Model	Default binding mode
Resource model	One-time
JSON model	Two-way
XML model	Two-way
OData model	One-way

The resource model only supports the one-time binding mode because it deals with static texts only.



- Resource bundle file located in project:

```
i18n.properties x
1 title=Full-screen Application
2 appTitle=Full-screen Application
3 appDescription=Full-screen Application.
4
5 # Overview View
6 overviewPageTitle=Carriers
7 columnId=Id
8 columnName=Name
9
```

- Code instantiating a ResourceModel and assigning it an alias of i18n:

```
manifest.json x
65 "models": {
66   "i18n": {
67     "type": "sap.ui.model.resource.ResourceModel",
68     "settings": {
69       "bundleName": "student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.i18n.i18n"
70     }
71   },
72 }
```

- Code binding resource to control in an XML view:

```
Carrier.view.xml x
1 <mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Carrier"
2   xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
3   <Page title="{i18n>overviewPageTitle}">
4     <content>
5       <Table items="{/CarrierCollection}">
6         <columns>
7           <Column minScreenWidth="Tablet" demandPopin="true">
8             <Text text="{i18n>columnName}" />
9           </Column>
10      </Table>
11    </content>
12  </Page>
13</mvc:View>
```

Figure 31: Binding Syntax for Resource Models

i18n is short for Internationalization.

The translation is provided by using files with ending .properties. The ResourceModel instance is instantiated during the instantiation of the component controller using a so called Application Descriptor.

The file name is *manifest.json*.

Inside the XML-view, the binding to the i18n is done using a model id.



- Code instantiating an XMLModel and assigning it raw XML:

```
var oModel = new sap.ui.model.xml.XMLModel();
oModel.setXML("<?xml version='1.0' encoding='UTF-8'?>"+
              "<carriers>"+
                "<carrier>"+
                  "<id>AA</id>"+
                  "<name>American Airlines</name>"+
                  "<currency>USD</currency>"+
                  "<URL>http://www.aa.com</URL>"+
                "</carrier>"+
              "</carriers>");

oTable.addColumn(
  new sap.ui.table.Column({
    label: new sap.ui.commons.Label({text: "URL"}),
    template: new sap.ui.commons.TextField().bindProperty("value", "URL/text()")
  })
);

oTable.setModel(oModel);
oTable.bindRows({path: "/carrier/"});
```

Figure 32: Binding Syntax for XML Models

The XML model allows to bind controls to XML data. It is a client-side model intended for small data sets, which are completely available on the client. The XML model does not contain mechanisms for server-based paging or loading of deltas. It supports two-way binding.



```
<companies>
  <company name="Treefish Inc">
    <info>
      <employees>3</employees>
    </info>
    <contact phone="873">Barbara</contact>
    <contact phone="734">Gerry</contact>
    <contact phone="275">Susan</contact>
  </company>
</companies>
```

- Absolute binding paths:

```
/company/@name
/company/info/employees
```

- Relative binding paths within the /company context:

```
@name
info/employees/text()
```

Figure 33: Additional Syntax Examples for XML Models

The above slide shows you how to access attributes and text nodes of an XML model.



- Code instantiating an ODataModel and assigning it the URL of a service:

```
var url = "/sap/opu/odata/iwbep/RMTSAMPLEFLIGHT_2/";
var oModel = new sap.ui.model.odata.v2.ODataModel(url);
```

- Code binding the ODataModel to controls in an XML view

- Binding the service's /CarrierCollection entity to the items of a Table control
- Binding the AirLineID and AirLineName properties of the CarrierCollection to certain attributes of the ColumnListItem control:

```
Carrier.view.xml ×
1  <mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Carrier"
2    xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
3    <Page title="{i18n>overviewPageTitle}">
4      <content>
5        <Table items="{$CarrierCollection}">
6          <columns>
7            <Column minScreenWidth="Tablet" demandPopin="true">
8              <Text text="{i18n>columnId}" />
9            </Column>
10           <Column minScreenWidth="Tablet" demandPopin="true">
11             <Text text="{i18n>columnName}" />
12           </Column>
13         </columns>
14         <items>
15           <ColumnListItem press="onPress" type="Navigation">
16             <cells>
17               <ObjectIdentifier title="{AirLineID}" />
18               <Text text="{AirLineName}" />
19             </cells>
20           </ColumnListItem>
21         </items>
22       </Table>
23     </content>
24   </Page>
25 </mvc:View>
```

Figure 34: Binding Syntax for OData Models

The OData model is a server-side model, meaning that the data set is only available on the server and the client only knows the currently visible (requested) data. Operations, such as sorting and filtering, are done on the server. The client sends a request to the server and shows the returned data.



- Use the manifest.json file to define DataSources

```

    "dataSources": {
      "mainService": {
        "uri": "/sap/opu/odata/iwbep/RMTSAMPLEFLIGHT_2/",
        "type": "OData",
        "settings": {
          "odataVersion": "2.0",
          "localUri": "localService/metadata.xml"
        }
      }
    }
  
```

Figure 35: Datasource Configuration in the App Descriptor

Using the *manifest.json* to declare a datasource is the recommended approach.



```

  "models": {
    "i18n": { ... },
    "": {
      "dataSource": "mainService",
      "preload": true,
      "settings": {
        "defaultBindingMode": "TwoWay",
        "defaultCountMode": "Inline",
        "refreshAfterChange": false
      }
    }
  }

```

Figure 36: Model Configuration in the App Descriptor

To instantiate a new model during application start, the developer has to configure a new configuration object at the *sap.ui5-configuration* object of the *manifest.json* file.

As you can see in the screenshot a model is introduced by adding a new configuration to the models-configuration object. You can define a model id followed by a configuration object.

The configuration object references the dataSource shown in the slide before. The so-declared model will be created during startup and added to the application component object.



- Raw JSON data acting as the model:

```
*mock.json  Component.js  Master.view.xml
1 {
2     "SalesOrderCollection": [
3         {
4             "SoId": "300000097",
5             "GrossAmount": "13224.47",
6             "BuyerName": "11113.00",
7             "TaxAmount": "2111.47",
8             "CurrencyCode": "EUR",
9             "ApprovalStatus": "",

var oModel = new sap.ui.model.json.JSONModel("model/mock.json");
oView.setModel(oModel);

• Control binding code is similar to
that of the ODataModel:
<List
    id="list"
    mode="{device>/listMode}"
    select="handleListSelect"
    items="{/SalesOrderCollection}" >
<ObjectListItem
    type="{device>/listItemType}"
    press="handleListItemPress"
    title="{SoId}"
    number="{GrossAmount}"
    numberUnit="{CurrencyCode}" >
```

Figure 37: Binding Syntax for JSON Models

To work with data provided in the JavaScript object notation (JSON), SAPUI5 provides the class `sap.ui.model.json.JSONModel`.

A JSON-model is a so-called clientside model. Clientside means all operations executed on the models data are executed locally on the client side. It is necessary to load the data before operate on the data.



- Binding to the value property of an Input control using JavaScript:

```
App.controller.js x
1 sap.ui.define([
2     "sap/ui/core/mvc/Controller",
3     "sap/m/Input"
4 ],
5 function (Controller,Input) {
6     "use strict";
7
8     return Controller.extend("com.sap.training.sampleproject.controller.App", {
9         onInit: function () {
10             var oInput = new Input({
11                 value : "{/company/name}"
12             });
13         });
14     });
15 });

onInit: function () {
    var oInput = new Input({
        value : { path: "/company/name" },
        mode: sap.ui.model.BindingMode.OneWay
    });
}
```

- Adding additional binding information such as binding mode
 - “path” must be used in the settings object if the literal string is not used:

- Binding can also be performed after control instantiation:

```
oInput.bindProperty("value","/company/name");
```

Figure 38: Other Binding Examples - Property Binding

To define property binding on a control, you have the following options:

- As part of the control's declaration in an XML view.
- Using JavaScript, in the settings object in the constructor of a control, or in special cases, using the `bindProperty` method of a control.

Once you have defined the property binding, the property is updated automatically every time the property value of the bound model is changed, and vice versa.



- Items are an aggregation of a sap.m.Select control
- Binding to the items property of a Select-control using a template:

```
<Select selectedKey="{Category}" items="/Categories" id="idSelect">
|   <core:Item text="{Name}" key="{Name}"/>
</Select>
```

- You can also use the bindAggregation method:

```
App.controller.js x
1  sap.ui.define([
2    "sap/ui/core/mvc/Controller",
3    "sap/ui/core/Item"
4  ],
5  function (Controller, Item) {
6    "use strict";
7
8    return Controller.extend("com.sap.training.sampleproject.controller.App", {
9      onInit: function () {
10        let oSelect = this.byId("idSelect");
11        oSelect.bindAggregation("items", "/Categories",
12          new Item({text:"Name",key:"Name"})
13        );
14      }
15    });
16  });
});
```

Figure 39: Other Binding Examples - Aggregation Binding

Aggregation binding is used to automatically create child controls according to model data.

You can define aggregation binding either in the *settings* object in the constructor or by calling the *bindAggregation* method. Aggregation binding requires the definition of a template, which is cloned for each bound entry of the list. For each clone that is created, the binding context is set to the respective list entry, so that all bindings of the template are resolved relative to the entry. The aggregated elements are destroyed and recreated whenever the bound list in the data model is changed.

Element Binding



- Element binding allows you to bind elements to a specific object in the model data, which will create a binding context and allow relative binding within the control and all of its children. This is especially helpful in master-detail scenarios.

The screenshot illustrates a SAPUI5 application using element binding. On the left, a JSON model object contains an array of customer data. On the right, a combobox is populated with customer names, and a detail view shows the selected customer's name, address, and postal code/city.

Figure 40: Element Binding

Element binding allows you to bind elements to a specific object in the model data, which will create a binding context and allow relative binding within the control and all of its children.



```
<ComboBox items="{/Customers}" selectionChange="onItemSelect">
  <items>
    |   <core:ListItem text="{Id}" />
  </items>
  <!-- sap.ui.core.Item -->
</ComboBox>
<f:SimpleForm id="simpleForm" minWidth="1024" maxContainerCols="2" editable="true" layout="ResponsiveGridLayout" . . .
labelSpanL="4" labelSpanM="4" emptySpanL="0" emptySpanM="0" columnsL="2" columnsM="2" class="editableForm">
  <f:content>
    <Label text="Name"/>
    <Input value="{CompanyName}" />
    <Label text="Address"/>
    <Input value="{Address}" />
    <Label text="ZIP Code/City"/>
    <Input value="{PostalCode} {City}" . . .
      <layoutData>
        <l:GridData span="L3 M3 S4" />
      </layoutData>
    </Input>
  </f:content>
</f:SimpleForm>
```

```
onItemSelect: function(oEvent) {
  var sPath = oEvent.getParameter("selectedItem").getBindingContext().getPath();
  this.getView().byId("simpleForm").bindElement(sPath);
}
```

Figure 41: Element Binding (Cont.)

The above shown code snippet shows the implementation of the combobox example from the previous slide. The event handler `onItemSelect` for the `selectionChange` event of the combobox receives the attribute `selectedItem`. It contains the selected item chosen by the user.

To get the path information of the selected item of the model we asked the item by choosing the `getBindingContext().getPath()` function and store the path to the selected item from the bound model to the variable `sPath`. Now we are able to change the binding of the form, to show the details of the selected `Item`. This is done by the `bindElement-function`.

Expression Binding



- **Expression Binding** is an enhancement of the SAPUI5 binding syntax, which allows for providing expressions instead of *Custom Formatter Functions*.
- An **Expression Binding** is specified via `{= expression}` in an XML view, where `expression` complies with a subset of the JavaScript expression syntax.
- Some characters such as the left angle bracket (<) and ampersand (&) that are used by operators need to be escaped in XML.
- Bindings embedded in the `expression` are specified in the following way `{= ${embedded}}`.

```
<!-- set the icon to visible, when status equals critical-->
<Icon src="sap-icon://message-warning" visible="={ ${status} === 'critical' }"/>
```



Figure 42: Expression Binding

The following are basics about Expression Binding:

- Expression binding is an enhancement of the SAPUI5 binding syntax, which allows for providing expressions instead of custom formatter functions.
- Saves the overhead of defining a function and is recommended in cases where the formatter function has a trivial implementation like comparison of values. Expression binding is especially useful in the context of SAPUI5XML templating where XML views with templating are preprocessed so that the SAPUI5 controller is a natural place to put custom formatter functions that are not available.
- An expression binding is specified via `{=expression}` in an XML view.
- Expression complies to a subset of JavaScript expression syntax except for bindings embedded in the expression which are specified like this `{=${embedded}}`.
- It is recommended to use Formatter Functions instead of very complex and hard-to-read expressions.
- To use Expression Binding, you need to enable Extended Binding Syntax via the configuration setting `xx-bindingSyntax` set to `complex`.
- To avoid escaping of operators in XML, you may do one of the following:
 - Rephrase the expression to make it more readable, for example, using `a > b` instead of `< a`.
 - Use a Custom Formatter Function.
- The following syntax elements can be used:
 - Literalsnumber, object, string, null, true, false

- Grouping{=(expression)}
- Unary operators!, +, -, typeof
- Multiplicative operators*, /, %
- Additive operators+, -
- Relational operators<, >, <=, >=
- Strict equality====, !==
- Binary logical operators&&, ||
- Conditional operator?
- Member access with the . operator and function calls. You can use members and member methods on standard types like string as follows {= \${/firstName}.length}.
- Math, RegExp and encodeURIComponent



```
<!-- set the icon to visible, when status equals critical-->
<Icon src="sap-icon://message-warning" visible="={ ${status} === 'critical' }"/>
<!--Set to visible if the status is critical and the amount is above the threshold (note escaping of &&)-->
<Icon src="sap-icon://message-warning" visible="={ ${status} === 'critical' &amp; ${amount} > 10000 }"/>
<!--Text for amount level using language-dependent texts from the resource model.-->
<Text text="={ ${/amount} > 10000 ? ${i18n>high} : ${i18n>normal} }"/>
<!--Set to visible if the rating is VIP, ignoring case or if the order amount is greater than 10,000.-->
<Icon src="sap-icon://message-warning" visible="={ ${/rating}.toUpperCase() === 'VIP' || ${/orderAmount} > 10000 }"/>
<!--Set to visible if the rating contains VIP, ignoring the case. -->
<Icon src="sap-icon://message-warning" visible="={ RegExp('vip' , 'i').test(${/rating}) }"/>
<!--Text is maximum of three values.-->
<Text text="={ Math.max(${/value1}, ${/value2}, ${/value3}) }"/>
<!--Control is enabled only if the order status is set.-->
<Input enabled="={ ${/orderStatus} !== null }"/>
```



Figure 43: Advanced Expression Binding Samples

The above slide shows some examples of the usage of expression binding.



LESSON SUMMARY

You should now be able to:

- Bind Data to a UI5 Control

Unit 2

Lesson 3

Describing Best Practices for SAPUI5 Applications



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe SAPUI5 best practices

Best Practices for SAPUI5 Applications

Best Practice 1: Determine View Creation Approach Early



- SAPUI5 allows you to define views using HTML, XML, JSON and JavaScript.
- SAP recommends to create views with XML:
 - Detection of errors at design time.
 - HTML can also be embedded into an XML view.
 - Tool support by SAP Business Application Studio is very good.

It is possible to implement views with HTML, JavaScript Object Notation (JSON), JavaScript and XML.

SAP recommends to use XML for view creation.

Best Practice 2: Ensure Application Independence



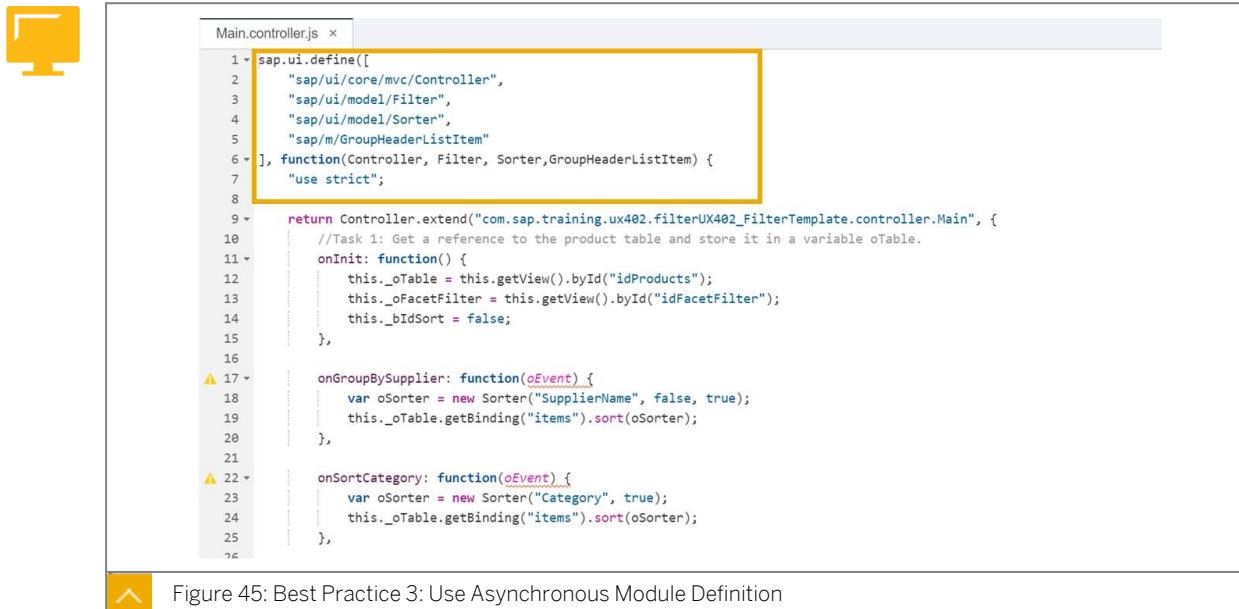
The screenshot shows the SAP Fiori Launchpad interface. At the top, there is a navigation bar with links for Home, SAP Education (UT 4.4), Technical Monitoring, Enterprise Search, Custom Code Migration, Extensibility, Fiori Launchpad, and Reference Apps. Below the navigation bar, there is a search bar and a user icon. The main area displays three application cards: "Shop EPM" (with a shopping cart icon and "0" items), "Purchase Orders EPM" (with a document icon and "20" items), and "Manage Products EPM" (with a gear icon and "123" items). The background is light gray, and the overall design is clean and modern.

Figure 44: Best Practice 2: Ensure Application Independence

To ensure Application Independence:

- The application should be accessible via an icon or a bookmark.
- The index.html should simply be the delivery mechanism for the app:
 - It should not have a lot of code in it other than the bootstrap code.
- Build the app so that you can take its functionality and embed it into any other structure:
 - Such as the Fiori Launchpad, the HTML-based user facing component.

Best Practice 3: Use Asynchronous Module Definition



```
Main.controller.js x
1 sap.ui.define([
2   "sap/ui/core/mvc/Controller",
3   "sap/ui/model/Filter",
4   "sap/ui/model/Sorter",
5   "sap/m/GroupHeaderListItem"
6 ], function(Controller, Filter, Sorter, GroupHeaderListItem) {
7   "use strict";
8
9   return Controller.extend("com.sap.training.ux402.filterUX402_FilterTemplate.controller.Main", {
10     //Task 1: Get a reference to the product table and store it in a variable oTable.
11     /**
12      * @private
13      */
14     /**
15      * @private
16      */
17     onGroupBySupplier: function(oEvent) {
18       var oSorter = new Sorter("SupplierName", false, true);
19       this._oTable.getBinding("items").sort(oSorter);
20     },
21
22     /**
23      * @private
24      */
25     onSortCategory: function(oEvent) {
26       var oSorter = new Sorter("Category", true);
27       this._oTable.getBinding("items").sort(oSorter);
28     }
29   });
30 }
```

Figure 45: Best Practice 3: Use Asynchronous Module Definition

With the introduction of the function `sap.ui.define` in the version 1.28, SAPUI5 has introduced the support for Asynchronous Module Definition (AMD).

- A module is a JavaScript file that can be loaded and executed in a browser.
- Asynchronous Module Definition (AMD) is a JavaScript API which specifies a way to define a module and its dependencies in such a way that they can be loaded asynchronously without worrying about the loading order.

Best Practice 4: Use of a Component to Encapsulate the App



- An app developed that is independent can typically be built by separate teams working on pieces of the overall functionality
- Requires building the app using the Component concept
- The component approach allows you to run the app within a standalone HTML page OR within a larger context such as the Fiori Launchpad

```
Component.js x
1 sap.ui.define([
2   "sap/ui/core/UIComponent",
3   "sap/ui/Device",
4   "com/sap/training/sampleproject/model/models"
5 ], function (UIComponent, Device, models) {
6   "use strict";
7
8   return UIComponent.extend("com.sap.training.sampleproject.Component", {
9     metadata: {
10       manifest: "json"
11     },
12
13   /**
14    * The component is initialized by UIS automatically during the startup of
15    * the app and calls the init method once.
16    *
17    * @public
18    * @override
19    */
20   init: function () {
21     // call the base component's init function
22     UIComponent.prototype.init.apply(this, arguments);
23
24     // enable routing
25     this.getRouter().initialize();
26
27     // set the device model
28     this.setModel(models.createDeviceModel(), "device");
29   }
30 });
31 });
32 });
```


Figure 46: Best Practice 4: Use of a Component to Encapsulate the App

Component.js is in the same folder as *index.html* in this example. Note that the root view is instantiating a view named *App*.

Best Practice 5: Describe Your App by a Set of Metadata



- The application runtime requirements are described by a set of metadata using an application descriptor
- The application descriptor is implemented by a file called *manifest.json*

```
manifest.json x
1 {
2   "_version": "1.1.0",
3   "sap.app": {},
4   "sap.ui": {},
5   "sap.uis": {
6     "_version": "1.1.0",
7     "rootView": {
8       "viewName": "sap.training.flight.view.Main",
9       "type": "XML"
10     },
11     "dependencies": {
12       "minUISVersion": "1.30.0",
13       "libs": {
14         "sap.m": {},
15         "sap.ui.core": {},
16         "sap.ui.layout": {}
17       }
18     },
19     "contentDensities": {
20       "compact": true,
21       "cozy": true
22     },
23     "models": {
24       "i18n": {
25         "type": "sap.ui.model.resource.ResourceModel",
26         "settings": {
27           "bundleName": "sap.training.flight.i18n.i18n"
28         }
29       },
30       "": {
31         "dataSource": "ZBC_TRAVEL_SRV",
32         "settings": {
33           "disableHeadRequestForToken": true
34         }
35       }
36     },
37     "routing": {
38       "config": {
39         " viewType": "HTML"
40       }
41     }
42   }
43 }
```


Figure 47: Best Practice 5: Describe Your App by a Set of Metadata

From SAPUI5 version 1.28 an application descriptor was introduced.

The application descriptor is implemented by a file with the name *manifest.json*. The *manifest.json* file is referenced by the Component implementation.

Best Practice 6: Minimal Code in index.html



- Only these bare essentials should be in the index.html:
 - Meta tags in the document header giving the browser rendering information
 - The UI5 bootstrap code which loads the UI5 runtime core
 - HTML body containing the declaration of one or more UI-areas with configuration to instantiate the component

```
index.html x
sampleproject > webapp > index.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <title>App Title</title>
8      <style>
9          html, body, body > div, #container, #container-uiarea {
10              height: 100%;
11          }
12      </style>
13      <script id="sap-ui-bootstrap"
14          src="resources/sap-ui-core.js"
15          data-sap-ui-theme="sap_fiori_3"
16          data-sap-ui-resourceroots='{
17              "com.sap.training.sampleproject": "./"
18          }'
19          data-sap-ui-oninit="module:sap/ui/core/ComponentSupport"
20          data-sap-ui-compatVersion="edge"
21          data-sap-ui-async="true"
22          data-sap-ui-frameOptions="trusted"
23      ></script>
24  </head>
25  <body class="sapUiBody sapUiSizeCompact" id="content">
26      <div data-sap-ui-component
27          data-name="com.sap.training.sampleproject"
28          data-id="container"
29          data-settings='{"id": "com.sap.training.sampleproject"}'
30          data-handle-validation="true"
31      ></div>
32  </body>
33  </html>
```

Figure 48: Best Practice 6: Minimal Code in index.html

Based on the namespace declaration the resources like views, fragments, util-classes and controller implementation are resolved.

Best Practice 7: Make the App Addressable, UI-Focused and Responsive



Allow for “deep linking”:

- Where the user can bookmark places in an app and be able to come back and pick up where they left off.
- Do not require that the user has to start at the top page of a website and descend through multiple links just to get information from a specific page.

Focus on the UI by following the MVC approach in keeping data model handling, the UI, and logic separate.

Build the app so that it makes the best use of available screen real-estate and navigation features regardless of device (phone, tablet, desktop):

- Make use of the sap.m library which was designed from the ground up to support this requirement.
- Create a “Device Model” that will allow you to determine things such as if the device is touch-capable.

Best Practice 8: Device Model



- The sap.ui.Device library allows you to determine various runtime platform characteristics

```
models.js x
1  sap.ui.define([
2    "sap/ui/model/json/JSONModel",
3    "sap/ui/Device"
4  ], function (JSONModel, Device) {
5    "use strict";
6
7    return {
8
9      createDeviceModel: function () {
10        var oModel = new JSONModel(Device);
11        oModel.setDefaultBindingMode("OneWay");
12        return oModel;
13      }
14    };
15  });
16});
```

Figure 49: Device Model, Overview

Using the sap.ui.Device API the developer is able to determine information about the runtime environment of the application.



```
// Set device model
var oDeviceModel = new sap.ui.model.json.JSONModel({
  isTouch: sap.ui.Device.support.touch,
  isNoTouch: !sap.ui.Device.support.touch,
  isPhone: sap.ui.Device.system.phone,
  isNoPhone: !sap.ui.Device.system.phone,
  listMode: sap.ui.Device.system.phone ? "None" : "SingleSelectMaster",
  listItemType: sap.ui.Device.system.phone ? "Active" : "Inactive"
});
oDeviceModel.setDefaultBindingMode("OneWay");
this.setModel(oDeviceModel, "device");

<List
  id="list"
  mode="{device>/listMode}"
  select="handleListSelect"
  items="{/SalesOrderCollection}" >
  <ObjectListItem
    type="{device>/listItemType}"
    press="handleListItemPress"
    title="{SoId}"
    number="{GrossAmount}"
```

Figure 50: Device Model, Code

Properties of the device model:

- The sap.ui.Device library allows you to determine various runtime platform characteristics.
- These characteristics are stored in a client-side JSON model.
- This model is defined in the Component, as seen below, and then is used throughout the application.

- An example of using the Device model is also shown in the List control's mode attribute.

A very common practice is to implement a device model to access runtime specific aspects for the used platform.

The developer can access these aspects using the sap.ui.Device implementation and create a JSON-model with application specific attributes for the runtime aspects. Beside this approach SAP Business Application Studio generates (depended on the used template) code for the device model creation.



- Driven explicitly or via the user's locale settings
- sap.m controls support right-to-left languages
- Manage the static text of the app separately, in a way that they can be easily swapped for a different locale
- Define the resource bundle in the config sap.ui5 object of the manifest.json file

```

{
  "models": {
    "i18n": {
      "type": "sap.ui.model.resource.ResourceModel",
      "settings": {
        "bundleName": "com.sap.training.sampleproject.i18n.i18n"
      }
    }
  }
}
  
```

The screenshot shows the SAP Business Application Studio interface. On the left, there is a tree view of a project structure named 'sampleproject'. Under 'sampleproject', there are 'node_modules', 'webapp', 'controller', 'css', and an 'i18n' folder. The 'i18n' folder is selected and highlighted in blue. Inside 'i18n', there are three files: 'i18n_de.properties', 'i18n_en.properties', and 'i18n.properties'. On the right, the 'manifest.json' file is open, showing the configuration for the 'i18n' model.

Figure 51: Best Practice 8: Be Prepared to Support Localization

During the initialization process of the application SAPUI5 determines the runtime language and load the correct localization file.

If no file for given language available then the file for the default language is loaded. SAPUI5 will instantiate a ResourceModel-object and provide the created model with a model id.



- Files can specify a language or a locale to separate regions such as U.S. English vs. British English
- Use data binding to the model

```

<Page
  id="masterPage"
  title="{i18n>masterTitle}">

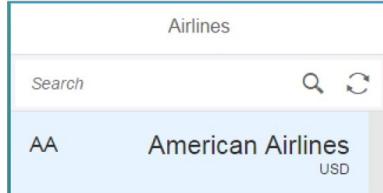
```

The screenshot shows the SAP Business Application Studio interface. On the left, there is a tree view of a project structure named 'sampleproject'. Under 'sampleproject', there are 'node_modules', 'webapp', 'controller', 'css', and an 'i18n' folder. The 'i18n' folder is expanded, showing three files: 'i18n_de.properties', 'i18n_en_GB.properties', and 'i18n.properties'.

```

masterTitle=Airlines
detailTitle=Flights
notFoundTitle=Not Found

```



```

masterTitle=Luftverkehrsgesellschafts
detailTitle=Flüge
notFoundTitle=Nicht gefunden

```

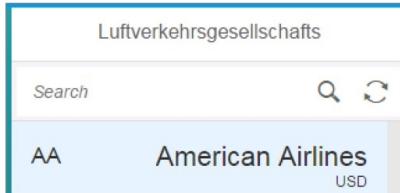


Figure 52: Best Practice 8: Be Prepared to Support Localization

As the current resource model is provided in the context of the component, you can easily bind the language depended aspects of your application to the ResourceModel-instance. This model instance can be accessed by the i18n model id defined in the manifest.json.

Best Practice 9: Make Use of Patterns Like Master-Detail

Figure 53: Best Practice 9: Make Use of Patterns Like Master-Detail

It is important to implement a common look and feel among UI5 applications. Therefore it is useful to implement UIs based on patterns.

One pattern is the Master-Detail-Pattern.

Figure 54: SplitApp and Master-Detail, 1

Recall that the App view is the root view and instantiated at app start by the Component. The App view defines the overall layout of the application.



- SplitApp has two sections in the form of aggregations – Calles masterPages and detailPages.
- What pages is shown when is defined in the routes and target configuration in the manifest.json

```

"routing": {
  "config": {
    "routerClass": "sap.m.routing.Router",
    "viewType": "XML",
    "async": true,
    "viewPath": "sap.training.masterdetail.view",
    "controlAggregation": "detailPages",
    "controlId": "idAppControl",
    "clearControlAggregation": false,
    "bypassed": {
      "target": ["master", "notFound"]
    }
  },
  "routes": [
    {
      "name": "master",
      "pattern": "",
      "target": ["object", "master"]
    },
    {
      "name": "object",
      "pattern": "Carriers/{objectId}",
      "target": ["object", "master"]
    }
  ],
  "targets": {
    "master": {
      "viewLevel": 1,
      "viewId": "master",
      "viewName": "Master",
      "controlAggregation": "masterPages"
    },
    "object": {
      "viewLevel": 1,
      "viewId": "detail",
      "viewName": "Detail"
    }
  }
}

```



Figure 55: SplitApp and Master-Detail, 2

The routing information from the *manifest.json* file defines the navigation routes for the application.

Best Practice 10: Create Custom Utilities to Share Code



Figure 56: Best Practice 10: Create Custom Utilities to Share Code

The figure shows a file tree for a SAPUI5 application named "sampleproject". The "util" folder contains a file named "Formatter.js", which is highlighted with a blue background. Other files in the "util" folder include "view", "Component.js", "index.html", "manifest.json", ".eslintrc", ".gitignore", "package-lock.json", "package.json", "tsconfig.json", "ui5.yaml", and "xs-app.json". The "sampleproject" folder also contains "node_modules" and "webapp" folders.

Utility classes are stored in a folder called *util*. The utility implementation uses also the ADM-syntax.



Inherits functionality

Figure 57: Best Practice 10: Reuse Controller Aspects

The figure displays two code snippets: `BaseController.js` and `App.controller.js`.

```

sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/core/routing/History"
], function (Controller, History) {
    "use strict";

    return Controller.extend("sap.training.masterdetail.controller.BaseController", {
        getRouter: function () {
            return this.getOwnerComponent().getRouter();
        },
        getListSelector: function () {
            return this.getOwnerComponent().oListSelector;
        },
        getResourceBundle: function () {
            return this.getOwnerComponent().getModel("i18n").getResourceBundle();
        },
        onNavBack: function() {
            var sPreviousHash = History.getInstance().getPreviousHash();
            if (sPreviousHash != undefined) {
                // The history contains a previous entry
                history.go(-1);
            } else {
                // Otherwise we go backwards with a forward history
                var bReplace = true;
                this.getRouter().navTo("master", {}, bReplace);
            }
        }
    });
});
```

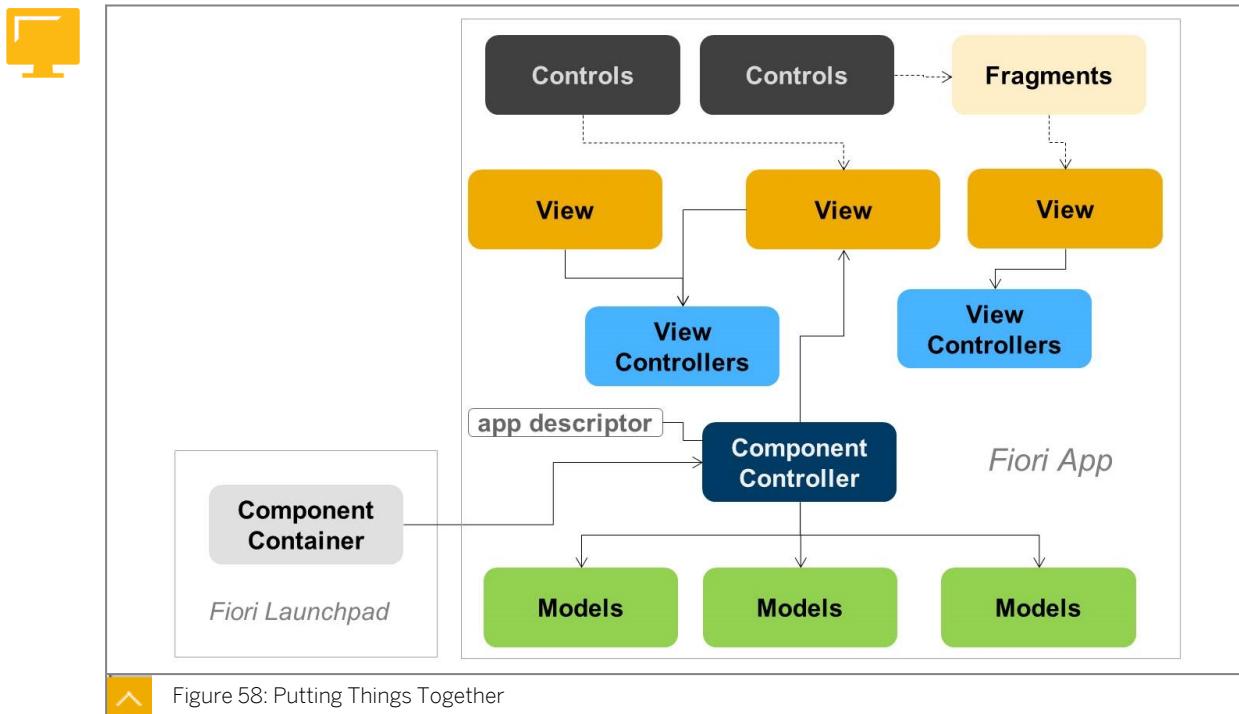
```

sap.ui.define([
    "sap/training/masterdetail/controller/BaseController",
    ],
    function (Controller) {
        "use strict";

        return Controller.extend("sap.training.masterdetail.controller.App", {
    });
});
```

Introduce a base controller that contains all common controller aspects. Derive the controller implementations from your application from the base controller to reuse and share functionality and data.

Putting Things Together



Summary:

- SAPUI5/Fiori apps are component-based and have a component controller.
- App descriptor contains application-specific configuration settings.
- SAPUI5 apps can have more than one model and can use different model types.
- Model instantiation via app descriptor.
- SAPUI5 views are built with controls and can have a view controller to handle events.
- Reusable view content can be grouped in fragments and can be reused in views.
- Different view types can be used in one application.



LESSON SUMMARY

You should now be able to:

- Describe SAPUI5 best practices

Unit 2

Lesson 4

Implementing App Navigation



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement in App Navigation and deep Linking

In-App Navigation

APP Navigation, Characteristics:



- SAPUI5 offers hash-based navigation, so called routing.
- Application can consist of a single page with navigation done by changing the hash:
 - No need to reload the entire page.
 - Better and faster experience for the user.
 - Repeatable and predictable URL pattern.
 - URLs can be bookmarked.
 - Parameters can be passed by the URL or between view.

SAPUI5 offers hash-based navigation, which allows you to build single-page apps where the navigation is done by changing the hash. In this way the browser does not have to reload the page; instead there is a callback to which the app and especially the affected view can react. A hash string is parsed and matched against patterns which will then inform the handlers.

You use routing in the following cases:
Enable users to navigate back using the browser history, for example, the Back button of the browser or a physical back button on mobile devices.

Enable bookmarks and deep links to pages inside an app; this means that you can start the app and resume the bookmarked state.

Pass on data via the hash to application logic.

You use routing in the following cases:

- Enable users to navigate back using the browser history, for example, the Back button of the browser or a physical back button on mobile devices.
- Enable bookmarks and deep links to pages inside an app; this means that you can start the app and resume the bookmarked state.
- Pass on data via the hash to application logic.

The following are characteristics of hash-based navigation:



One of the most important features of hash-based navigation is the ability to bookmark a view:

Users can leave a page and return to it without navigating through the app's main startup view.

Each change of the hash is inserted into the browser history chain:

- Allows the user to use the familiar browser back button.
- Dialogs and Popups cannot be bookmarked.

Navigation can also be done without a hash and without changing the URL or the browser history chain when required.

Routing and Navigation in Detail

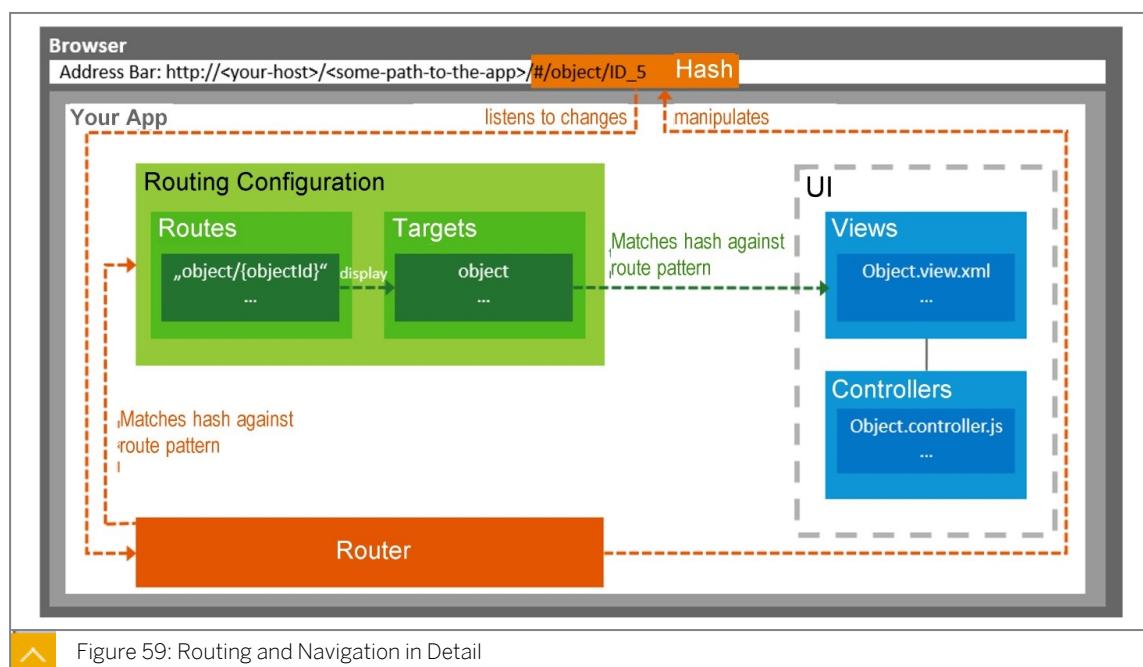


Figure 59: Routing and Navigation in Detail

In SAPUI5, navigation and routing is implemented using a "router" to forward the hash change and the data in the hash to one or more views of the app.

You use routes to notify your application that the hash has changed to a certain value. For each route, you define the pattern that can be used in the app implementation.

With targets, you define where a view is loaded and where the view is shown on the UI. By referring to one or multiple targets in a route's definition, you can load and show the views once the route's pattern matches the current hash.

Routing Patterns are:



- Router checks whether there is a route with a matching pattern.
- First matching route is taken, corresponding target view is called.
- Data provided with the hash are passed to the target.
- Examples:

- <https://appurl.com/#/ProductList> - Link to a view in the app.
- <https://appurl.com/#/ProductDetails/12345> - Link to a specific product for example.
- <https://appurl.com/#/ProductDetails/12345/tab2> - Link to a specific tab on the details view.

Patterns

The following kinds of patterns are available:



- Hard-coded pattern
- Route with mandatory parameter
- Route with optional parameter

Hard-coded pattern:

The pattern matches the hash exactly. For example, when a pattern is defined as product/settings, this pattern matches only if the hash is product/settings and no data is passed on to the events of the route.

Route with mandatory parameter:

You can define mandatory parameters for the pattern by placing the parameter in curly brackets ({parameter ID}).

For example, if you define the pattern product/{id}, the hashes product/5 and product/3 (where 3 and 5 are product IDs) match the pattern. The matched event handler gets 5 or 3 passed on with the key id in its arguments. But hash product/ does not match the pattern because the mandatory parameter is missing.

Route with optional parameter:

You can define optional parameters for the pattern by placing the parameter between colons (:parameter ID:).

For example, if you define a pattern product/{id}/detail/:detailId:, the detailId parameter is optional, whereas id is mandatory. Both hashes product/5/detail and product/3/detail/2 match the pattern.

Routing



- Navigation is configured in the *manifest.json* or *Component.js* using a Routing section that defines one or more routes.
- A route is connected to a target configuration.
- Routes form a pattern that the URL is checked against to decide where to navigate.
- Routes are checked from top to bottom and completes when the first route is matched.
- Catch all patterns can be used as the last route for any URL that does not match any other route pattern.

The routing configuration is done inside the *manifest.json*. Because the *manifest.json* as an Application Descriptor was added to SAPUI5 starting with SAPUI5 version 1.28 it is possible that older SAPUI5 versions do have their routing configuration inside the *Component.js* metadata-section.

Routing, Code

■ A simple routing section with target configuration is shown below

```

"routing": {
  "config": {
    "routerClass": "sap.f.routing.Router",
    "viewType": "XML",
    "viewPath": "student00.com.sap.training.ux402.templateusage.ux402_usingtemplate.view",
    "controlId": "layout",
    "controlAggregation": "beginColumnPages",
    "bypassed": {
      "target": "notFound"
    },
    "async": true
  },
  "routes": [
    {
      "pattern": "",
      "name": "master",
      "target": "master"
    },
    {
      "pattern": "CarrierCollection/{objectId}",
      "name": "object",
      "target": [
        "master",
        "object"
      ]
    }
  ],
  "targets": {
    "master": {
      "viewName": "Master",
      "viewLevel": 1,
      "viewId": "master"
    },
    "object": {
      ...
    }
  }
}

```

References target



Figure 60: Routing, Code

Routing configuration consists of routes, targets, config, and owner.

Routes

Each route defines a name, a pattern, and optionally one or more targets to which to navigate when the route has been matched. In the routes section, you define which patterns are available for navigation.

- The name of the route (unique within one router instance).
- The pattern as hash part of the URL that matches the route.
- The navigation target as defined in the targets section.
- If you want to load multiple views at the same time, you can assign multiple targets.
- The parent to specify a route reference in a parent component.
- The titleTarget to specify from which target the title is taken when multiple targets are displayed. If no titleTarget is defined, the first target that has a title is chosen.

Targets

A target defines the view that is displayed. It is associated with one or more routes or it can be displayed manually from within the app. Whenever a target is displayed, the corresponding view is loaded and added to the aggregation configured with the controlAggregation option of the control. The target definition can contain the following parameters:

- The target key.
- The viewName.
- Additional optional parameters.

If you do not specify a parameter, the default value is taken from the config section.viewType (e.g. XML).

- **viewId** of the view instance.
- **viewLevel**: You can use different levels to define the navigation direction, for example, the navigation from a lower view level to a higher view level leads to forward navigation. This is important for flip and slide transitions, where the slide animation should go from left to right or vice versa.
- **controlId** of the control that is used to display the view (for example, app).
- **controlAggregation** target aggregation to which the view is added.
The `NavContainer` control, for example, has an aggregation called `Pages` and the shell container has `Content`.
- **parent**: a view is created and added before the target view is added.
- **viewPath** where the view is located in the app.
- **targetParent** where the control is located.
- **clearAggregation** specifies whether the aggregation should be cleared before adding the new view instance. When you use the `sap.m.routing.Router` the default is `false`, for `sap.ui.core.routing.Router` it is `true`.

When using `sap.ui.ux3.Shell` this value should be set to true, for `sap.m.NavContainer` to `false` to ensure that the correct content is shown:

- Transition defines how the transition happens; you can choose between slide (default), flip, fade, and show.
- Title contains either a static text or a valid binding syntax. For example, to an i18n model, which is resolved under the binding context of the view.

Routers



- The router configuration is part of the `manifest.json` file
- Standard Router are provided
- Usually the `sap.m.routing.Router` implementation is used
- If needed customer router are possible by extending standard router
- The router is configured using the `routerClass`-property in the `config-customizing` object in the `routing-object`.

```
"routing": {
  "config": {
    "routerClass": "sap.f.routing.Router",
    "viewType": "XML",
    "viewPath": "student00/com.sap.training.ux402/templateusage.ux402_usingtemplate.view",
    "controlId": "layout",
    "controlAggregation": "beginColumnPages",
    "bypassed": {
      "target": "notFound"
    },
    "async": true
  }
},
```

Figure 61: Routers 1/2

The figure shows an example code of a standard router.



- Routers are initialized in Component.js init function

```
init: function () {
    // call the base component's init function
    UIComponent.prototype.init.apply(this, arguments);

    // enable routing
    this.getRouter().initialize();

    // set the device model
    this.setModel(models.createDeviceModel(), "device");
}
```

- To access the router and to use its functions, use this.getRouter() function or the static getRouterFor function of the UI component. You can pass either a controller, or a view.

```
var oRouter = sap.ui.core.UIComponent.getRouterFor(this);
// or
var oRouter = this.getOwnerComponent().getRouter();
```



Figure 62: Routers 2/2

The figure gives further information about the used code for routers.

Trigger Navigation



The navTo method can be used in code for navigation to a defined route name.

```
class sap.ui.core.routing.Router
```

Overview Constructor Events ▾ Methods ▾

▼

attachRouteMatched

Attaches event handler fnFunction to the routeMatched event of this sap.ui.core.routing.Router.
When called, the context of the event handler (its this) will be bound to oListener if specified, otherwise it will be bound to this sap.ui.core.routing.Router itself.

Visibility: public

`attachRouteMatched(oData?, fnFunction, oListener?) : sap.ui.core.routing.Router`

Param	Type	Default Value	Description
oData?	object		An application-specific payload object that will be passed to the event handler along with the event object when firing the event
fnFunction	function		The function to be called when the event occurs
oListener?	object		Context object to call the event handler with, defaults to this sap.ui.core.routing.Router itself



Figure 63: Trigger Navigation

The *navTo-method* of the *sap.ui.core.routing.Router* implementation is used to trigger navigation.

The method takes the name of the route as a first argument, the optional arguments.

Parameter Passing with Navigation



- Parameters can be configured with the route configuration (1,2), passed with the navTo method (3), and handled in the attachRouteMatched handler to exchange data between views (4).

```

1  {
  "pattern": "CarrierCollection/{objectId}",
  "name": "object",
  "target": [
    "master",
    "object"
  ]
}

2  "targets": {
  "object": {
    "viewName": "Detail",
    "viewId": "detail",
    "viewLevel": 2
  },
}

3  this.getOwnerComponent().getRouter().navTo("object", {objectId : sObjectId}, true);

4  var sObjectId = oEvent.getParameter("arguments").objectId;

```

Figure 64: Parameter Passing with Navigation

The above slides shows how to configure a navigation route with corresponding targets.

Navigation Handling in the Navigation Target



- The attachRouteMatched event is fired every time the Router matches a route. This is used by views to handle the navigation, such as reading a parameter to update what is displayed on the view.
- Views often only react to a specific route so the name of the route matched is checked in the function that handles the attachRouteMatched event.

```

this.getOwnerComponent().getRouter().getRoute("object").attachPatternMatched(this._onObjectMatched, this);

_onObjectMatched : function (oEvent) {
  var sObjectId = oEvent.getParameter("arguments").objectId;
  this.getModel().metadataLoaded().then( function() {
    var sObjectPath = this.getModel().createKey("CarrierCollection", {
      AirLineID : sObjectId
    });
    this._bindView("/" + sObjectPath);
  }.bind(this));
},

```

Figure 65: Handle Navigation in the Navigation Target

The figure shows the used code for handling navigation in the navigation target.

Navigation Triggering without Changing the Hash



- Access the targets of the target configuration and call the display method

```
_onBindingChange: function() {
    var oView = this.getView(),
        oElementBinding = oView.getElementBinding();

    // No data for the binding
    if (!oElementBinding.getBoundContext()) {
        this.getRouter().getTargets().display("notFound");
        return;
    }
},
```

Figure 66: Trigger Navigation without Changing the Hash

Sometimes you want to navigate to a target without changing the hash. This might be necessary when you don't want to provide the possibility of creating a bookmark, for example when displaying a *NotFound*-page.

The figure shows you how to access the configured targets using the *getTarget*-method of the *Routing-API* and how to show a specific target using the *display*-method.

Passing Data to a Target



Pass data to a navigation target

```
handleCreatePressed: function() {
    var dateFormat = DateFormat.getDateInstance({
        pattern: "yyyy-MM-dd hh:mm"
    });
    var oFlDate = encodeURIComponent(dateFormat.format(new Date.parse(this.sFlDate)));

    this.getRouter().getTargets().display("createbooking", {
        carrierId: this.sObjectId,
        connid: this.sConnId,
        fldate: oFlDate
    });
},
```

Attaching an event handler in the controller of the target

```
onInit: function() {
    var oRouter, oTarget;
    oRouter = this.getRouter();
    oTarget = oRouter.getTarget("createbooking");
    oTarget.attachDisplay(this._onDisplayMatched, this);
},
```

Figure 67: Passing Data to a Target

The above slide shows you how to pass data to a navigation target and how to register an event handler for the *display*-event.



LESSON SUMMARY

You should now be able to:

- Implement in App Navigation and deep Linking

Implementing a Full-screen Application



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement a full-screen application

Basic Architecture of a Full-screen Application

The sap.m.App



- The root control for implementing a full-screen app is the `sap.m.App` control.
 - `sap.m.App` adds certain header tags to the HTML page that are necessary for proper display.
 - `sap.m.App` offers basic functionality to navigate between pages.
 - The content entities between which `sap.m.App` navigates can be of type `sap.m.Page`, `sap.ui.core.View`, `sap.m.Carousel` or any other control with fullscreen/page semantics.

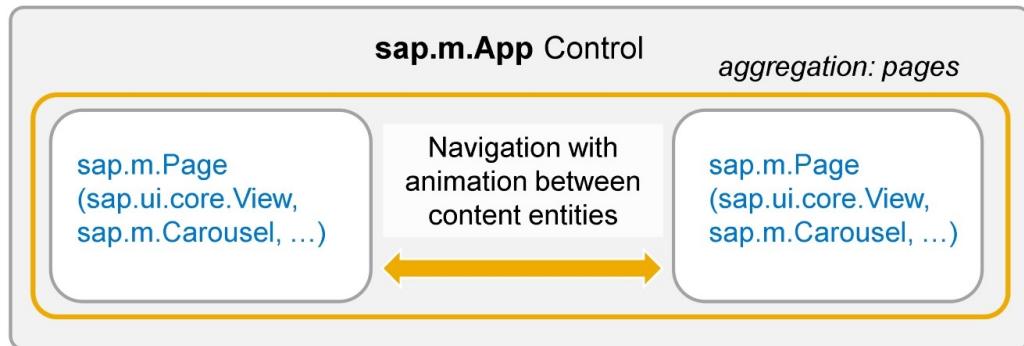
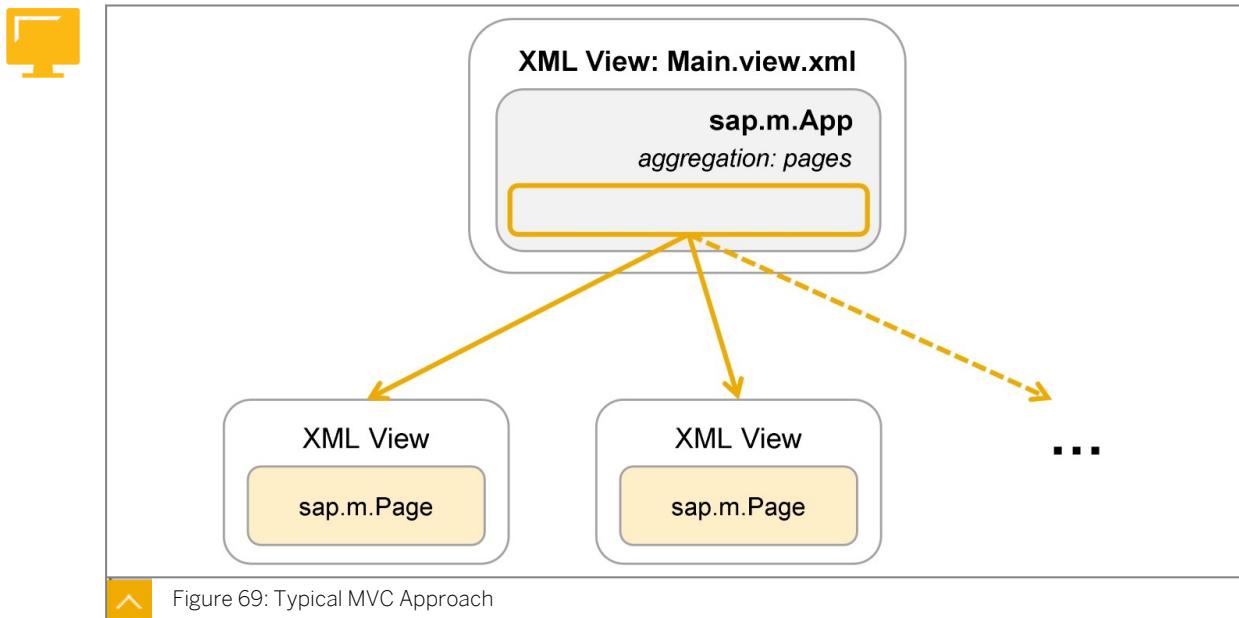


Figure 68: sap.m.App

Apps are often composed of several pages and the user can drill-down to detail pages and go back up again. The transition between the pages when navigating is often animated (for example, horizontal slide animation).

SAPUI5 supports this pattern by providing the control `sap.m.App`. The default type of transition when navigating is "slide". Other predefined options are "fade", "flip", and "show".

`sap.m.App` inherits the navigation capabilities from the `sap.m.NavContainer` control. Although the `sap.m.App` control provides functionality for navigation, it is not recommended to use this approach for productive applications (see routing).



The granularity of Views can be freely chosen. The most common approach is to use one View per page. The figure illustrates such a usage of the MVC concept in order to decouple the different aspects of the application.

The recommended application design is to have one XML that only contains the implementation of the `sap.m.App` control as a root control. The remaining Views (also XML Views) are used for the pages.

The `sap.m.App` control provides an aggregation called `pages`. The controls aggregated by the `pages` aggregation of `sap.m.App` receive navigation events like `beforeShow`. These navigation events are documented in the pseudo interface `sap.m.NavContainerChild`.

The Main.view.xml File



Description:

- The Main view only contains the empty `App` tag (1).
- Specify the App view in the Application Descriptor (`manifest.json`) as root View that shall be opened.
- The pages will be added automatically into the App control according to the current URL by the router.
- The router identifies the App control by means of the id.
- `displayBlock="true"` prevents vertical scrollbars appearing with Views that are set to 100% height (2).

Example Page of Carrier.view.xml



```

Carrier.view.xml x
1   <mvc:View
2     controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Carrier"
3     xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
4     <Page title="{i18n>overviewPageTitle}">
5       <content>
6         <Table items="{/CarrierCollection}">
7           <columns>
8             <Column minScreenWidth="Tablet" demandPopin="true">
9               <Text text="{i18n>columnName}"/>
10            </Column>
11            <Column minScreenWidth="Tablet" demandPopin="true">
12              <Text text="{i18n>columnName}"/>
13            </Column>
14          </columns>
15          <items>
16            <ColumnListItem press="onPress" type="Navigation">
17              <cells>
18                <ObjectIdentifier title="{AirLineID}">
19                  <Text text="{AirLineName}"/>
20                </cells>
21            </ColumnListItem>
22          </items>
23        </Table>
24      </content>
25    </Page>
26  </mvc:View>

```

Figure 71: Example Page

sap.m.Page provides a basic container for a mobile application screen.

sap.m.Page provides an aggregation with the name *content*. The aggregation takes to 0..N controls.

The index.html File



```

index.html x
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>ux402_fullscreen</title>
7      <script id="sap-ui-bootstrap"
8        src=".//resources/sap-ui-core.js"
9        data-sap-ui-theme="sap_belize"
10       data-sap-ui-resourceroots='{"student00.com.sap.training.ux402.fullscreen.ux402_fullscreen": "./"}'
11       data-sap-ui-compatVersion="edge"
12       data-sap-ui-oninit="module:sap/ui/core/ComponentSupport"
13       data-sap-ui-async="true"
14       data-sap-ui-frameOptions="trusted">
15    </script>
16  </head>
17  <body class="sapUiBody">
18    <div
19      data-sap-ui-component data-name="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen"
20      data-id="container" data-settings='{"id": "ux402_fullscreen"}'>
21    </div>
22  </body>
23 </html>

```

Figure 72: index.html

Routing and Navigation

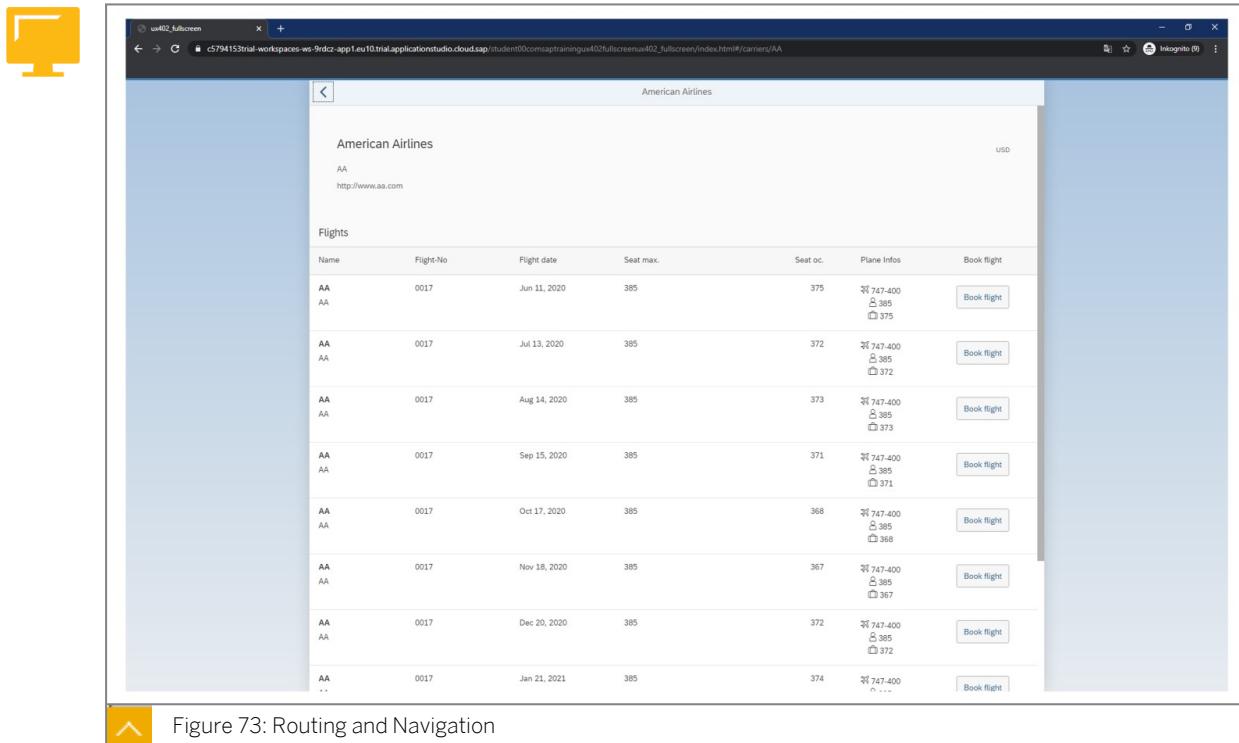


Figure 73: Routing and Navigation

SAPUI5 allows building apps where the navigation is done by changing the hash.

A route is used to notify your application that the hash has changed to a certain value. The pattern of the route is matched against the hash. If the pattern matches, the route provides the data passed over the hash to a defined handler.

Hash-based navigation enables you to use the current hash as a bookmarkable URL.

Routing, Configuration



- SAPUI5 applications can use a so-called **router** to dispatch hash-based URLs to one or more views of the app.
- To configure the **router**, add a *routing* section to the *sap.ui5* section in *manifest.json*.

```
manifest.json x
1 * {
2   "_version": "1.5.0",
3   "sap.app": { },
4   "sap.ui": { },
5   "sap.ui5": {
6     "rootView": { },
7     "dependencies": { },
8     "contentDensities": { },
9     "models": { },
10    "resources": {
11      "css": [
12        { "uri": "css/style.css" }
13      ]
14    },
15    "routing": {
16      "config": { },
17      "routes": [ ],
18      "targets": { }
19    }
20  }
21}
```

Figure 74: Routing Configuration

The pattern of a route is basically the hash part of the URL that matches the route.

As of SAPUI5 version 1.30, SAP recommends that you define the routing in the *manifest.json* descriptor file using routes and targets. In older versions of SAPUI5, the routing configuration had to be done directly in the metadata section of the Component, and with different syntax.

There are three properties to control the application flow when the user triggers a navigation action or opens a link to the application directly:

- The config configuration object contains the global router configuration and default values that apply for all routes and targets.
- The routes configuration object contains configuration object for navigation routes. Each route defines a name, a pattern, and one or more targets to navigate to when the route has been hit.
- The targets configuration contains target config. Objects for each target. A target defines a view that is displayed.

The Config Section of the Routing Definition



- All parameters of the **config** section can be overruled in the individual route and target definitions, if needed.

```

"routing": {
  "config": {
    "routerClass": "sap.m.routing.Router",
    "viewType": "XML",
    "async": true,
    "viewPath": "student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.view",
    "controlAggregation": "pages",
    "controlId": "app",
    "transition": "slide",
    "bypassed": {
      "target": ["notFound"]
    }
  },
}

```

1

2

Figure 75: Config Section - of the Routing Definition

(1) The default value for `routerClass` is `sap.ui.core.routing.Router`. In this example, the `routerClass` is set to `sap.m.routing.Router` because we implement an app based on `sap.m`. Compared to `sap.ui.core.routing.Router` the `sap.m.routing.Router` is optimized for mobile apps and adds the properties `viewLevel`, `transition` and `transitionParameters` which can be specified for every route or target created by the `sap.m.routing.Router`. Please check the API Reference for more information.

(2) The property `viewPath` specifies where the views of the application are located. The `controlId` describes the id of the control that should be used as the main target control. The `controlAggregation` property defines the name of the aggregation property of the control defined with the `controlId` property.

The transition property can be used to define the animation type when navigating between views; Possible values are slide (default), flip, fade, and show.

Routes Section: Example



- The sequence of the routes in the **routes** section is important because only the first matched route is used by the router.

```

"routes": [
  {
    "name": "overview",
    "pattern": "",
    "titleTarget": "",
    "greedy": false,
    "target": ["overview"]
  },
  { }
]

```

1

2

Figure 76: Routes Section - Example

The above code snippet shows you the definition for an empty pattern. This empty pattern matches the empty hash.

(1) Definition of a unique route name that is used to navigate to a certain route or to determine the matched route in one of the matched handlers.

The target property references one or more targets from the targets section that will be displayed when the route has been matched.

Targets Section: Example



- A target can be assigned to one or more routes, but it's not necessary. Targets can also be displayed directly in the app without hitting a route.

```

"routing": {
    "config": { },
    "routes": [ ],
    "targets": {
        "overview": { 1
            "viewType": "XML",
            "transition": "slide",
            "viewName": "Carrier", 2
            "viewLevel": 1
        },
        "flights": { 3
            "viewType": "XML",
            "transition": "slide",
            "viewName": "Flights",
            "viewLevel": 2
        }
    }
}
  
```

Figure 77: Targets Section - Example

Whenever a target is displayed, the corresponding view is loaded and added to the aggregation configured with the controlAggregation option of the control. The control is configured using `controlId`.

- (1) Each target configuration has a unique key as an identifier.
- (2) Additionally the property `viewName` needs to be specified. The value of the property and the value of the property `viewPath` from the config section will be concatenated to load the necessary view.
- (3) The `viewLevel` is relevant for flip and slide transitions. It helps the router to determine the direction of the transition from one page to another.

Router, Initialization



- The Router needs to be initialized by the Component: In the `init()` method of `Component.js`, get a reference to the Router and call `initialize()` on it.
- After initialization, the routing configuration in `manifest.json` is automatically enabled in the app.
- Initializing the Router will evaluate the current URL and load the corresponding view automatically.

```
return UIComponent.extend("com.sap.training.ux402.fullscreenUX402_FullScreenExercise.Component", {

    metadata: {},

    init: function() {
        this.getRouter().initialize();
    }
});
```



Figure 78: Initializing the Router

You do not need to instantiate the Router manually, it is automatically instantiated based on the configuration in `manifest.json` and assigned to the Component.

But, when using the SAPUI5 template of the SAP Business Application Studio you have to invoke the initialize function by your own. This is not generated.

Catching Invalid Hashes

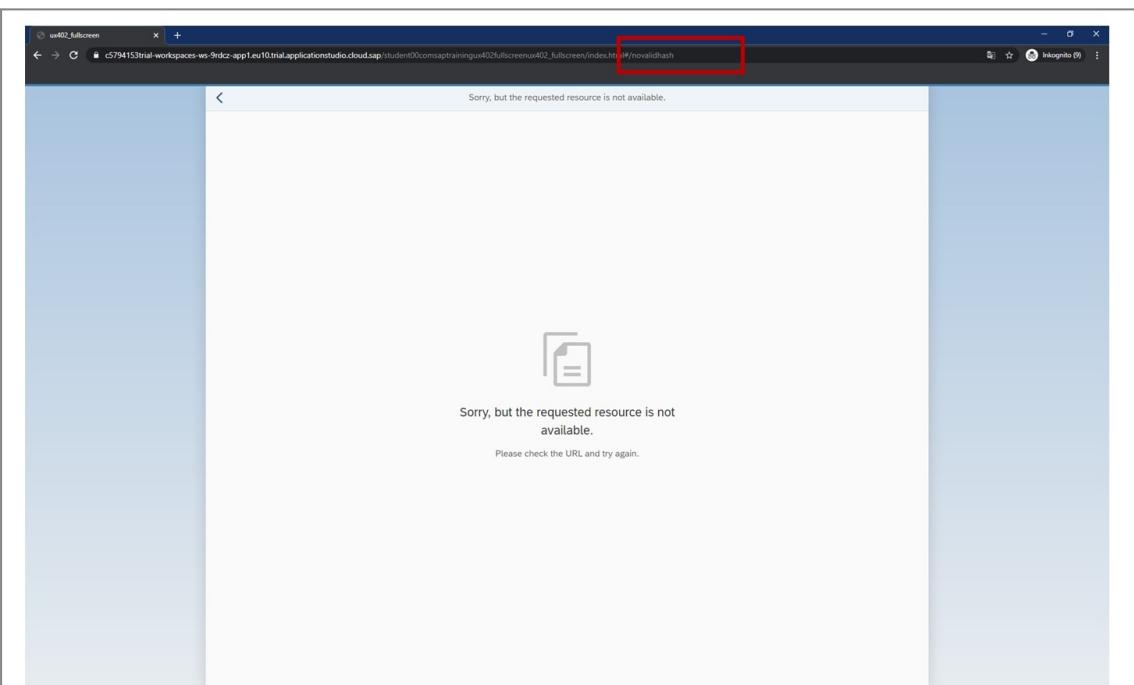
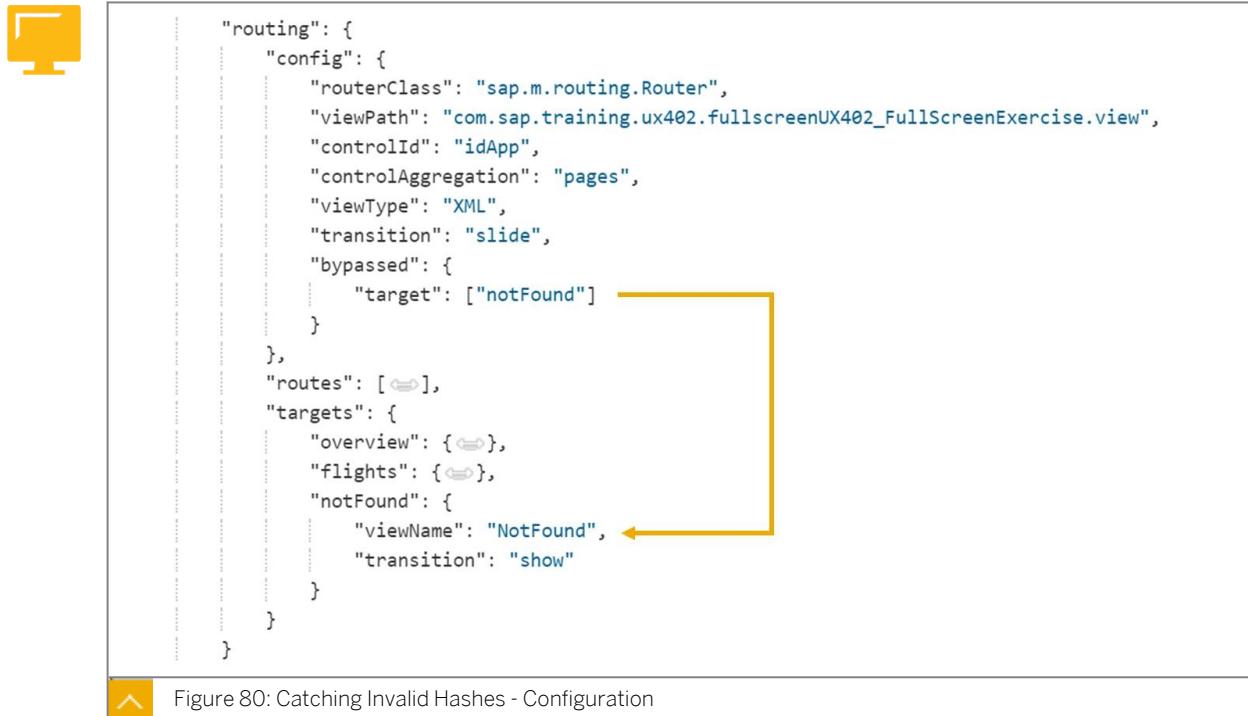


Figure 79: Catching Invalid Hashes

If a user tries to access an invalid pattern which does not match any of the configured routes, the user should get an indication that the requested resource was not found.

Implementation of the Back Navigation in the NotFound View Controller



The screenshot shows the SAP Fiori Launchpad interface. On the left, there's a yellow icon representing a monitor. The main area displays the configuration code for the Router. A yellow box highlights the 'bypassed' section under 'routing'. Another yellow box highlights the 'notFound' target under 'targets'. A third yellow box highlights the 'show' transition under the 'notFound' target.

```

"routing": {
    "config": {
        "routerClass": "sap.m.routing.Router",
        "viewPath": "com.sap.training.ux402.fullscreenUX402_FullScreenExercise.view",
        "controlId": "idApp",
        "controlAggregation": "pages",
        "viewType": "XML",
        "transition": "slide",
        "bypassed": {
            "target": ["notFound"]
        }
    },
    "routes": [ ],
    "targets": {
        "overview": { },
        "flights": { },
        "notFound": {
            "viewName": "NotFound",
            "transition": "show"
        }
    }
}

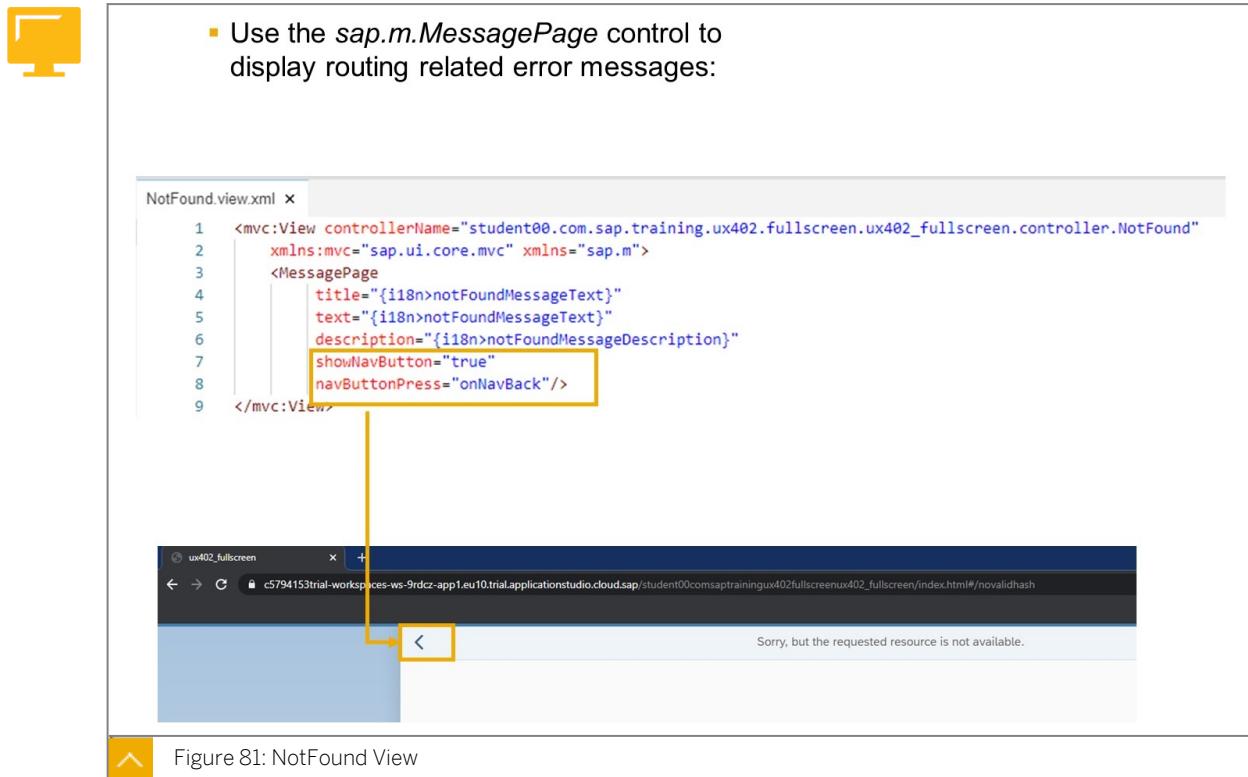
```

Figure 80: Catching Invalid Hashes - Configuration

Using the bypassed property, you tell the Router to display a special target (here: *notFound*) in case no route was matched to the current hash.

The *notFound* target configures a *NotFound* view with a show transition.

The NotFound View



The screenshot shows the SAP Fiori Launchpad interface. On the left, there's a yellow icon representing a monitor. The main area displays the XML code for the *NotFound* view. A yellow box highlights the *MessagePage* control. Another yellow box highlights the 'showNavButton="true"' and 'navButtonPress="onNavBack"/>' attributes. Below the code, a screenshot of a browser window shows a blue header bar with a back arrow icon. The main content area displays the message 'Sorry, but the requested resource is not available.'

```


  1 <mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.NotFound"
  2   xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
  3   <MessagePage
  4     title="{i18n>notFoundMessageText}"
  5     text="{i18n>notFoundMessageText}"
  6     description="{i18n>notFoundMessageDescription}"
  7     showNavButton="true"
  8     navButtonPress="onNavBack"/>
  9 </mvc:View>

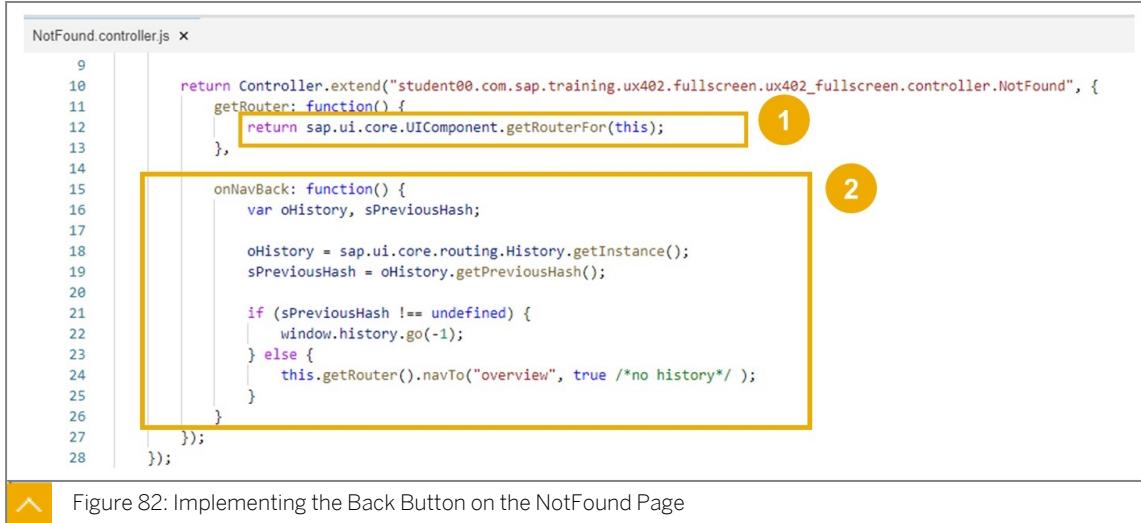
```

Figure 81: NotFound View

When displaying the *NotFound*-view it is important to provide the possibility to navigate back to the overview page. Therefore we have to visualize (not matter on what device) the back button.

When the user clicks on the button the *navButtonPress* is fired and handled by the assigned function in the controller.

Implementation of the Back Button on the NotFound page



```

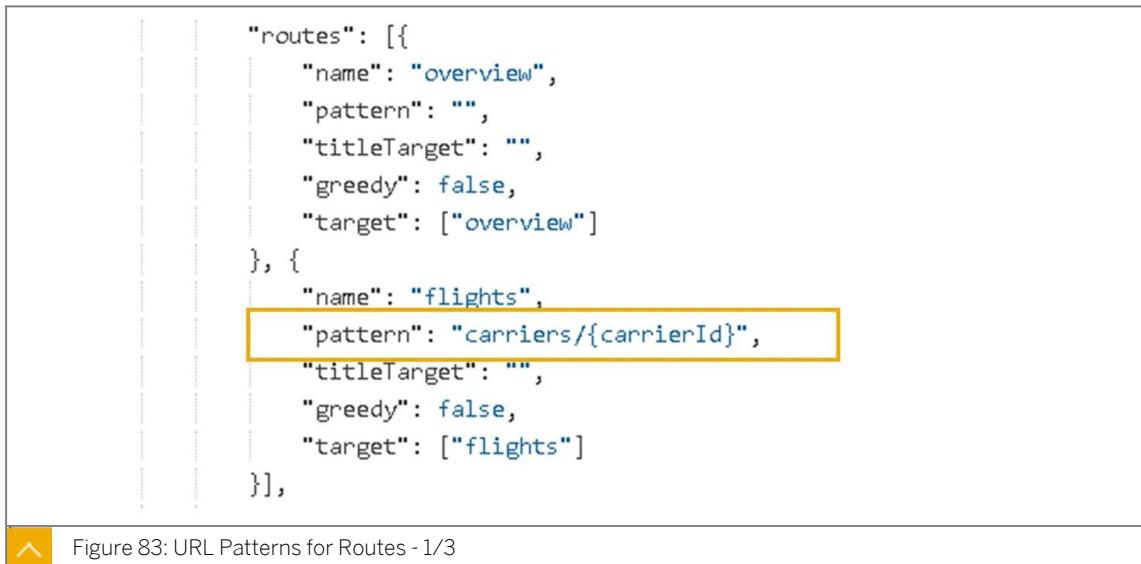
    9
  10   return Controller.extend("student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.NotFound", {
  11     getRouter: function() {
  12       return sap.ui.core.UIComponent.getRouterFor(this);
  13     },
  14
  15     onNavBack: function() {
  16       var oHistory, sPreviousHash;
  17
  18       oHistory = sap.ui.core.routing.History.getInstance();
  19       sPreviousHash = oHistory.getPreviousHash();
  20
  21       if (sPreviousHash !== undefined) {
  22         window.history.go(-1);
  23       } else {
  24         this.getRouter().navTo("overview", true /*no history*/ );
  25       }
  26     }
  27   });
  28 }

```

Figure 82: Implementing the Back Button on the NotFound Page

1. Implement a function that returns the router instance for your component. It is also possible to use *this.getOwnerComponent().getRouter()*.
2. Implement the event handler function *onNavBack*. This function checks if there is a previous hash value in the app history, if so, redirect to the previous hash via the browser's native History API. If not, use the Router to navigate to the route overview which is the Overview view. The function *navTo* of the router is invoked using the name of the route and a boolean parameter indicating if a browser history entry should be created. *true*: the hash is replaced (no browser history entry) *false*: the hash is set (browser history entry).

URL Patterns for Routes



```

  "routes": [
    {
      "name": "overview",
      "pattern": "",
      "titleTarget": "",
      "greedy": false,
      "target": ["overview"]
    },
    {
      "name": "flights",
      "pattern": "carriers/{carrierId}",
      "titleTarget": "",
      "greedy": false,
      "target": ["flights"]
    }
  ],

```

Figure 83: URL Patterns for Routes - 1/3

The pattern parameter of a Route defines the string that is matched against the hash.

A pattern may consist of the following:

- Hardcoded parts
- Mandatory parameters
- Optional parameters
- Mandatory query parameters
- Optional query parameters

URL Patterns for Routes

The following are URL Patterns for Routes:



Table 3: URL Patterns for Routes

Pattern type	Example	
hardcoded parts	"carrier/settings"	This pattern will only match if the hash of the browser is carrier/settings. No arguments will be passed to the events of the Route.
mandatory parameters	"carrier/{carrierid}"	{carrierid} is a mandatory parameter. For example, the following hashes would match: carrier/AA. The matched event of the Route will get AA passed as carrierid in its arguments. The hash carrier/ will not match.
optional parameters	"carrier/{carrierid}/show-tab/:tabid:"	:tabid: is an optional parameter For example, the following hashes would match: carrier/AA/bookings
mandatory query parameters	"carrier{?query}"	{?query} allows you to pass queries with any parameters For example, the following hashes would match: carrier?first=value1 carrier?first=value1&second=value2
optional query parameters	"carrier/{carrierid}/flights?:query:"	:?query: is an optional query parameter. e. g. the following hashes would match: carrier/AA/flights?from=New York carrier/AA/flights

Example: Navigate to Routes with Mandatory Parameters

The screenshot displays two consecutive pages from a web application, likely built with SAPUI5, illustrating the navigation between routes with mandatory parameters.

Page 1: Carrier (18)

- The URL in the browser is `c5794153trial-workspaces-ws-9rdz-app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingu402templateusageeu402_usingtemplate/index.html#`.
- The page title is "Carrier (18)".
- A search bar at the top left contains the placeholder "Search".
- A list of airline names is shown, with "American Airlines" highlighted and circled with a yellow circle labeled "1".
- Other listed airlines include Air Canada, Air France, Alitalia, British Airways, and Air Pacific.

Page 2: American Airlines

- The URL in the browser is `c5794153trial-workspaces-ws-9rdz-app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingu402templateusageeu402_usingtemplate/index.html#/CarrierCollection/AA`.
- The page title is "American Airlines".
- The subtitle is "Carrier Flights (26)".
- A table titled "Flight Details" lists flight segments with their destination cities and total booking sums:

Flight Details	Total Booking Sum
new york 0017	194083.08 USD
new york 0017	192865.11 USD
new york 0017	192949.60 USD
new york 0017	191871.11 USD
new york 0017	191943.12 USD
new york 0017	189566.13 USD

Figure 84: Example: Navigate to Routes with Mandatory Parameters

In the example, we implement a feature that allows the user to click on an airline in the list to see additional details of the airline.

The detail page has to read the ID of the airline from the URL to fetch and display the airline data from the server. If the airline was not found, for example, because an invalid ID was passed on, we want to inform the user by displaying the *notFound* target. Of course, the back navigation has to work as well for the detail page.

Definition of the Route



```

    "routes": [
        {
            "name": "overview",
            "pattern": "",
            "titleTarget": "",
            "greedy": false,
            "target": ["overview"]
        },
        {
            "name": "flights",
            "pattern": "carriers/{carrierId}",
            "titleTarget": "",
            "greedy": false,
            "target": ["flights"]
        }
    ],
    "targets": {
        "overview": {
            "viewType": "XML",
            "transition": "slide",
            "viewName": "Carrier",
            "viewLevel": 1
        },
        "flights": {
            "viewType": "XML",
            "transition": "slide",
            "viewName": "Flights",
            "viewLevel": 2
        }
    }

```

Figure 85: Definition of the Route

The following are properties of the example:

- Each Airline entity is identified by an ID.
- The {carrierId} part of the pattern is a mandatory parameter as indicated by the curly brackets. The hash that contains an actual carrier id is matched against that pattern at runtime (1).
- The target (2) of the route is flights. We create the target flights with viewLevel 2. With that, we make sure that we have the correct slide animation direction.

Carrier View, Implementation

The screenshot shows the SAPUI5 code editor with the file `Carrier.view.xml` open. The code defines a `mvc:View` with a `content` aggregation. Inside the `content` aggregation, there is a `Table` control (1) with `items` bound to `/CarrierCollection`. Each item in the collection is represented by a `ColumnListItem` (2) containing two `Text` elements: `{i18n>columnId}` and `{i18n>columnName}`. The `Table` has two columns defined in its `columns` section.

```

<mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Carrier" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
  <Page title="{i18n>overviewPageTitle}">
    <content>
      <Table items="/CarrierCollection">
        <columns>
          <Column minScreenWidth="Tablet" demandPopin="true">
            <Text text="{i18n>columnId}" />
          </Column>
          <Column minScreenWidth="Tablet" demandPopin="true">
            <Text text="{i18n>columnName}" />
          </Column>
        </columns>
        <items>
          <ColumnListItem press="onPress" type="Navigation">
            <cells>
              <ObjectIdentifier title="{AirLineID}" />
              <Text text="{AirLineName}" />
            </cells>
          </ColumnListItem>
        </items>
      </Table>
    </content>
  </Page>
</mvc:View>

```

Carrier Collection Data:

Carriers	
Id	Name
AA	American Airlines
AC	Air Canada
AF	Air France
AZ	Alitalia

Figure 86: Implementing the Carrier View

The `sap.m.Page` holds an aggregation with the name `content`. The content aggregation is of type `sap.ui.core.Control`. In the above sample a `sap.m.Table` control (1) is added to the content aggregation.

The table is bound to an entity `CarrierCollection`. For each entity of the collection an object of type `ColumnListItem` (2) is created and added to the table.

Example: Handler for the Press Event in the Carrier View Controller

The screenshot shows a SAP Fiori application interface titled "Carriers". On the left is a sidebar with carrier codes: AA, AC, AF, AZ, BA, and FJ. The main area displays a table with columns "Id" and "Name". The first row, "AA American Airlines", is highlighted with a yellow box and has a circled '1' above it. The second row, "AC Air Canada", and the third row, "AF Air France", both have circled '1' above them. Below the table is a code editor window for "Carrier.controller.js". A yellow box highlights the "onPress" event handler, which contains code to get the source item, its binding context, and the carrier ID, then navigate to the "flights" route with the carrier ID. This section is circled with a circled '2'.

```

sap.ui.define([
    "sap/ui/core/mvc/Controller"
], /**
 * @param {typeof sap.ui.core.mvc.Controller} Controller
 */
function (Controller) {
    "use strict";

    return Controller.extend("student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Carrier", {
        getRouter: function() {
            return sap.ui.core.UIComponent.getRouterFor(this);
        },
        onPress: function(oEvent) {
            var oItem = oEvent.getSource();
            var oCtx = oItem.getBindingContext();
            var sCarrid = oCtx.getProperty("AirLineID");

            this.getRouter().navTo("flights", {
                carrierId: sCarrid
            }, false);
        }
    });
});

```

Figure 87: Handler for the Press Event in the Carrier View Controller

In the example above, the code is doing:

- Determine the carrier id of the clicked item (1) by querying the binding context (2) and accessing the `AirLineID` from the data model.
- Navigate to the flights route and pass a configuration object on to the `navTo()` method with the mandatory parameter `carrierId` filled with the correct carrier id.

Example: Detect the Carrier in the Flights View Controller

```

1  sap.ui.define([
2    "sap/ui/core/mvc/Controller",
3    "student00/com.sap.training.ux402/fullscreen/ux402_fullscreen/control/HoverButton",
4    "sap/m/MessageToast",
5    "student00/com.sap.training.ux402/fullscreen/ux402_fullscreen/control/PlaneInfo",
6    "sap/m/MessageBox"
7  ],
8  function (Controller, HoverButton, MessageToast, PlaneInfo, MessageBox) {
9    "use strict";
10
11    return Controller.extend("student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Flights", [
12      {
13        name: "onInit",
14        value: function () {
15          var oRouter = this.getRouter();
16          oRouter.getRoute("flights").attachMatched(this._onObjectMatched, this);
17        }
18      },
19      {
20        name: "getRouter",
21        value: function () {
22          return sap.ui.core.UIComponent.getRouterFor(this);
23        }
24      }
25    ]);
26
27    /**
28     * @param {sap.ui.event.Event} oEvent
29     */
30    var _onObjectMatched = function(oEvent) {
31      var oArgs = oEvent.getParameter("arguments");
32      this._sCarrierId = oArgs.carrierId;
33      var oView = this.getView();
34
35      oView.bindElement({
36        path: "/CarrierCollection('" + this._sCarrierId + "')",
37        events: {
38          change: this._onBindingChange.bind(this),
39          dataRequested: function() {
40            oView.setBusy(true);
41          },
42          dataReceived: function() {
43            oView.setBusy(false);
44          }
45        }
46      });
47    };
48  });

```

Figure 88: Detect the Carrier in the Flights View Controller

The example works as follows:

(1) Instead of calling `attachMatched(...)` on a route we could also call `attachRouteMatched(...)` directly on the router. However, the event for the latter one is fired for every matched event of any route in the whole app. We don't use the latter one because we would have to implement an additional check for making sure that the current route is the route that has been matched. We want to avoid this extra overhead and register on the route instead.

(2) In the event handler function of the matched event we access the `arguments` parameter from the `oEvent` parameter that contains all parameters of the pattern, that is, the mandatory parameter `carrierId` is available as a key in `arguments`. Call `bindElement()` on the view to make sure that the data of the specified carrier is available in the view and its controls.

The Flights view should show the details of the selected carrier and the flights for that carrier:

- The ODataModel will handle the necessary data requests to the backend in the background. While the data is loading, it would be nice to show a busy indicator by simply setting the view to busy. Therefore we pass an events object to `bindElement()` to listen to the events `dataRequested` and `dataReceived`. The attached functions handle the busy state by calling `oView.setBusy(true)` and `oView.setBusy(false)` respectively.

Flights View, Implementation

The screenshot shows the implementation of the Flights View. On the left, the XML code for `Flights.view.xml` is displayed, highlighting line 4 which contains the `navButtonPress` event handler. On the right, a screenshot of the application interface shows the carrier details for American Airlines (AA) and a table of flights.

American Airlines Carrier Details:

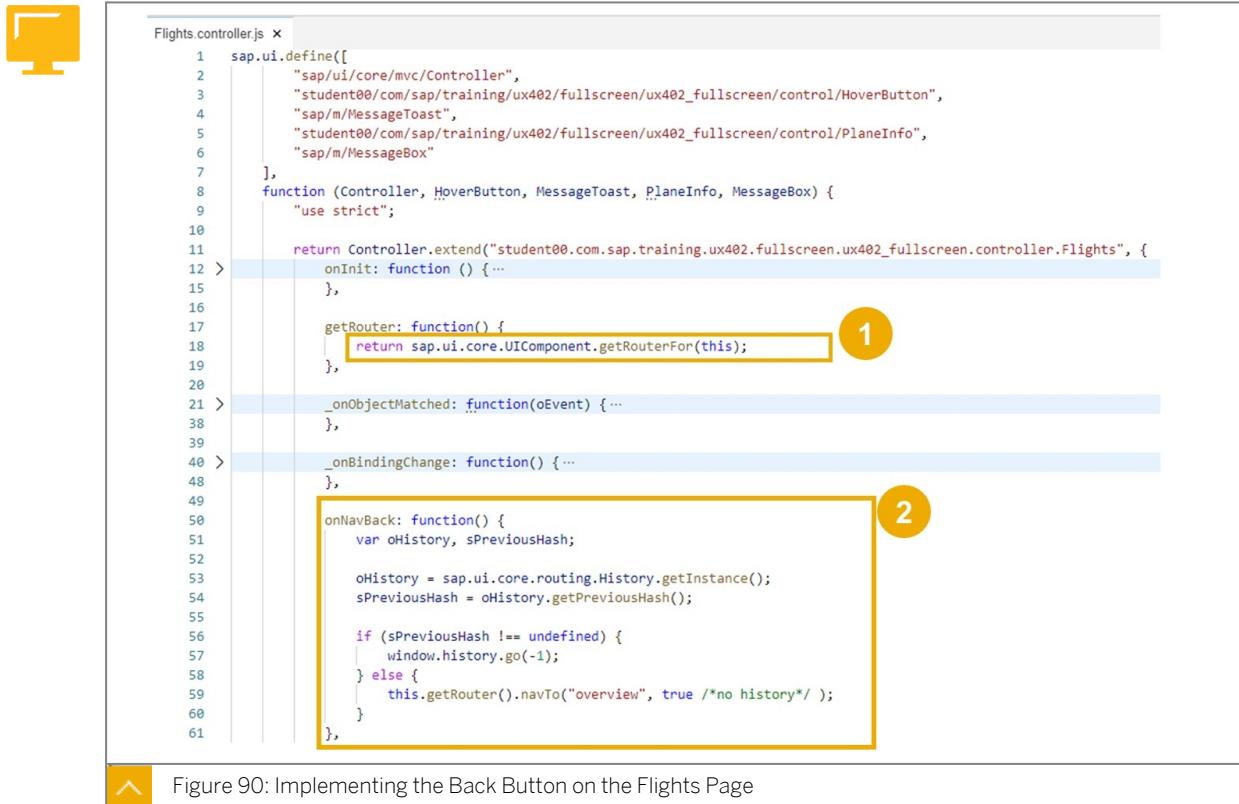
- Title: American Airlines
- Code: AA
- URL: <http://www.aa.com>
- Currency: USD

Flights Table:

Name	Flight-No	Flight date	Seat max.	Seat oc.	Plane Infos	Book flight
AA	0017	Jun 11, 2020	385	375	747-400 385 375	<button>Book flight</button>
AA	0017	Jul 13, 2020	385	372	747-400	<button>Book flight</button>
..						

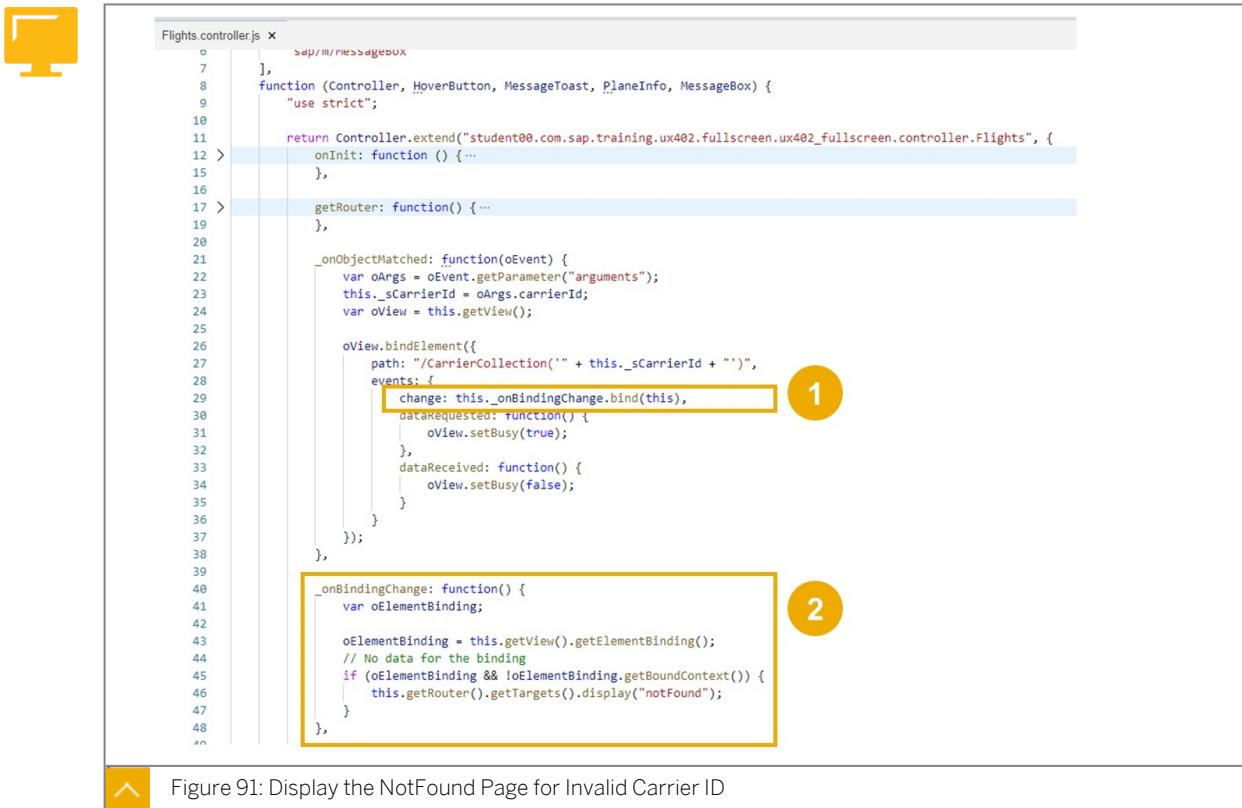
The Flights view implementation shows the details of the selected carrier and the list of flights for the selected carrier in a table. The data comes from a relative data binding that is set in the Flights view controller. To provide the functionality for navigating back to the carrier view we add the property `showNavButton` with the value `true` to the page object. To handle the back navigation we assign an event handler for the `navButtonPress` event (1). The event handler is implemented inside the flights controller.

Implementation of the Back Button on the Flights Page



1. Implement a function that returns the router instance for your component. It is also possible to use `this.getOwnerComponent().getRouter()`.
 2. Implement the event handler function `onNavBack`. This function checks if there is a previous hash value in the app history, if so, redirect to the previous hash via the browser's native History API. If not, use the Router to navigate to the route overview which is the carriers overview view. The function `navTo` of the router is invoked using the name of the route and a boolean parameter indicating if a browser history entry should be created.
true: the hash is replaced (no browser history entry)
false: the hash is set (browser history entry).

Display the NotFound Page for Invalid Carrier ID



To react on the change event of the binding operation you can attach an event handler (1).

You can display targets manually without referencing them in a navigation route. Good examples for this are temporary errors, switching to an edit page for a business object, or going to a "Settings" page.

Inside the `_onBindingChange()` (2) handler we get a reference to the Targets helper object of the router and simply call `display("notFound")`. The view associated to the target with the name `notFound` from the routing configuration will be displayed by the router without changing the hash.

The `sap.m.routing.Targets` object is retrieved by calling `getTargets()` on the router. However, you could also get a reference to the `sap.m.routing.Targets` object by calling `this.getOwnerComponent().getRouter().getTargets()` or `this.getOwnerComponent().getTargets()`.



LESSON SUMMARY

You should now be able to:

- Implement a full-screen application

Unit 2

Lesson 6

Implementing a Master-Detail Application



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement a Master-Detail-Application

Basic Architecture of a Master-detail Application

The sap.f.FlexibleColumnLayout



The figure shows a screenshot of the sap.f.FlexibleColumnLayout. It consists of three columns. The left column, labeled 'Master area', contains a list of products with details like name, price, and a link to view more. The middle column, labeled 'Detail view first level', shows a detailed view of a selected product (Audio/Video Cable Kit) with tabs for General Information and Suppliers. The right column, labeled 'Detail view second level', shows a detailed view of a supplier (Technocom) with tabs for Product ID, Description, Supplier, and Suppliers. Arrows point from the labels to their respective parts in the screenshot.

Figure 92: The sap.f.FlexibleColumnLayout

The figure shows the basic idea of the master-detail pattern when using the `sap.f.FlexibleColumnLayout`. The layout control consists of three areas. The `beginColumnPages` aggregation contains the master-view. The `midColumnPages` contains the details of the first level. In some cases it is necessary to show another level of details. Therefore the `endColumnPages` can be used.

Master Detail Pattern, Use

Usage of Master Detail Pattern:



When to use

You need to review and process different items quickly with minimal navigation.

When not to use

- You need to offer complex filters for the list of items.
- You need to see different attributes for each item in the list, and compare these values across items.
- You want to display a single object. Do not use the master list to display different facets of the same object.

Usage of sap.f.FlexibleColumnLayout

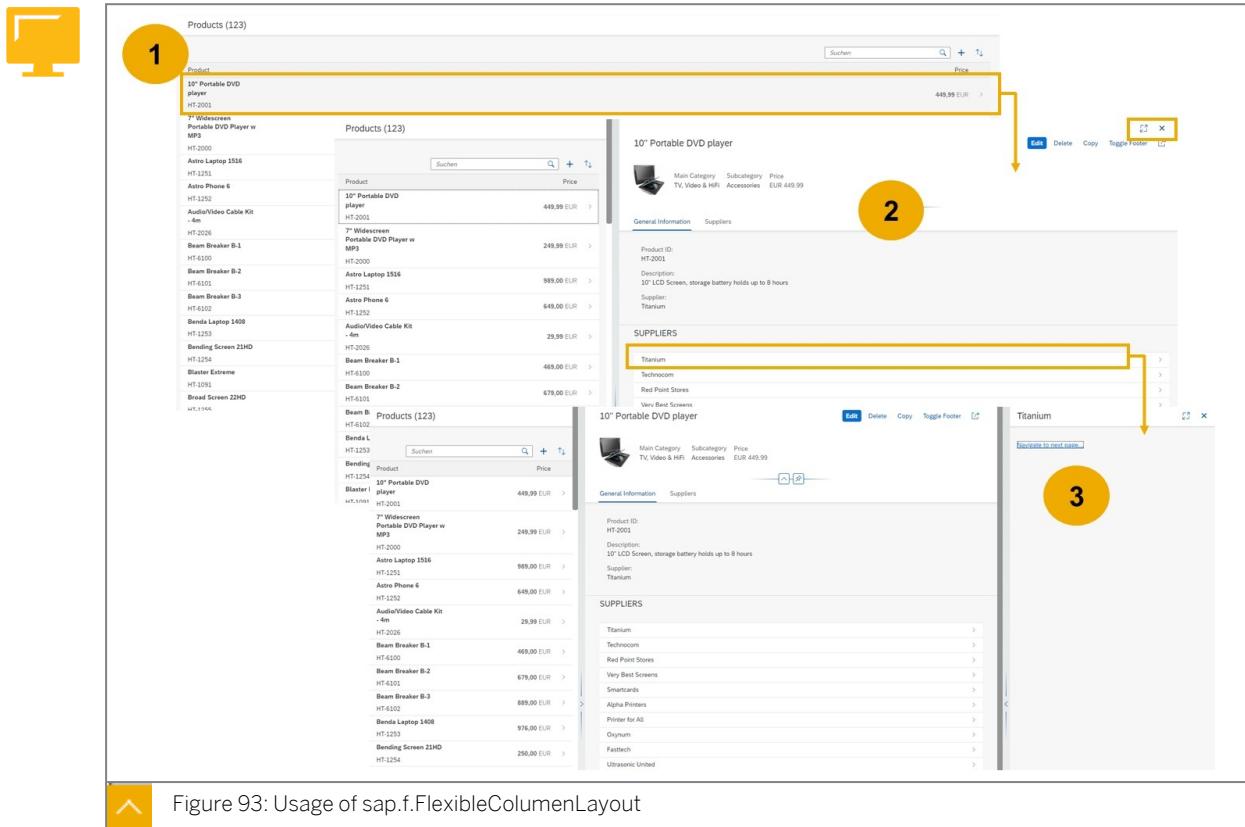
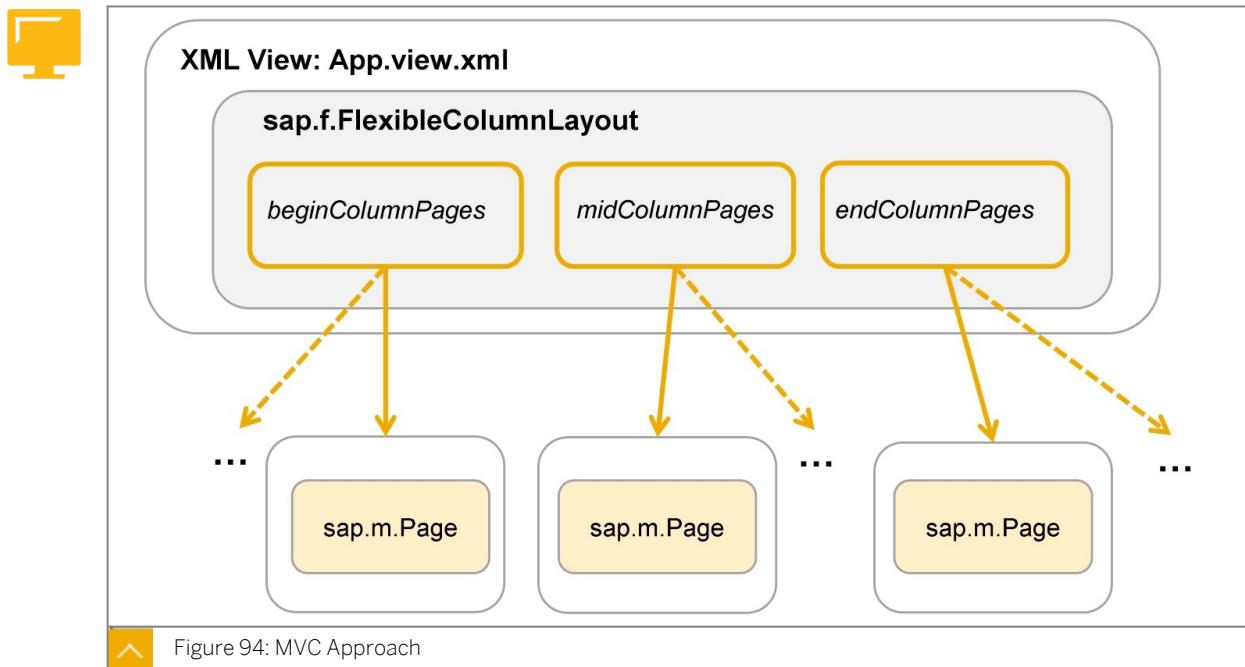


Figure 93: Usage of sap.f.FlexibleColumnLayout

The figure shows the usage of the *sap.f.FlexibleColumnLayout*. It is possible to implement a fullscreen behavior for the detail-pages. This might be useful when the details page contains a lot of information in various formats like charts or documents.

MVC Approach



The `beginColumnPages`, `midColumnPages` and `endColumnPages` aggregations of the Layout control aggregates the content entities.

The content entities can be of type `sap.f.DynamicPage`, `sap.ui.core.View`, `sap.m.Carousel` or any other control with fullscreen/page semantics.

The App View

```

App.view.xml X
flexiblecolumnlayout > webapp > view > o App.view.xml
1   <mvc:View controllerName="student00.sap.training.flexiblecolumnlayout.controller.App" xmlns:mvc="sap.ui.core.mvc"
2     displayBlock="true" xmlns="sap.m" xmlns:f="sap.f">
3     <Shell id="shell">
4       <App id="app">
5         <f:FlexibleColumnLayout id="idFlexLayout" layout="{appView>/layout}" backgroundDesign="Translucent"/>
6       </App>
7     </Shell>
8   </mvc:View>

```

Figure 95: The App View

1. The `App.view.xml` contains the container Control `sap.m.Shell`. The Shell-controls handles the correct visualization on different devices and screensizes using the letterboxing-concept. The `sap.m.Shell-control` contains the `sap.m.App-control`. Inside the App-control the `sap.f.FlexibleLayoutControl` is embedded..
2. Specify the App View in the Application Descriptor (`manifest.json`) as root View that shall be opened.
3. The pages will be added automatically using the rooting configuration or if necessary by code.
4. The router identifies the `sap.f.FlexibleColumnLayout` control by means of the id.

Root View Configuration



The screenshot shows the manifest.json file in a code editor. The rootView configuration is highlighted with a yellow box. The code snippet is as follows:

```

    "flexEnabled": false,
    "rootView": {
        "viewName": "student00.sap.training.flexiblecolumnlayout.view.App",
        "type": "XML",
        "async": true,
        "id": "App"
    },

```

 Figure 96: Root View Configuration

The `rootView-configuration` object specifies the full qualified path to the view with the overall layout control.

index.html



The screenshot shows the index.html file in a code editor. The SAPUI5 configuration section is highlighted with a yellow box. The code snippet is as follows:

```

    <!DOCTYPE html>
    <html>
        <head>
            <meta charset="utf-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>flexiblecolumnlayout</title>
            <script id="sap-ui-bootstrap"
                    src="./resources/sap-ui-core.js"
                    data-sap-ui-theme="sap_fiori_3"
                    data-sap-ui-resourceroots="{'student00.sap.training.flexiblecolumnlayout': './'}"
                    data-sap-ui-compatVersion="edge"
                    data-sap-ui-oninit="module:sap/ui/core/ComponentSupport"
                    data-sap-ui-async="true"
                    data-sap-ui-frameOptions="trusted">
            </script>
        </head>
        <body class="sapUiBody">
            <div data-sap-ui-component data-name="student00.sap.training.flexiblecolumnlayout" data-id="container" data-settings='{"id": "flexiblecolumnlayout"}'></div>
        </body>
    </html>

```

 Figure 97: index.html



Figure 98: App.controller.js

Routing Configuration - Config Section - Example



- All parameters of the *config* section can be overruled in the individual route and target definitions, if needed.

```
manifest.json X
flexiblecolumnlayout > webapp > manifest.json > sap.ui5 > routing > ...
88 },
89 "routing": {
90   "config": {
91     "routerClass": "sap.m.routing.Router",
92     "viewType": "XML",
93     "async": true,
94     "viewPath": "student00.sap.training.flexiblecolumnlayout.view",
95     "controlAggregation": "beginColumnPages",
96     "controlId": "idFlexLayout",
97     "clearControlAggregation": true,
98     "bypassed": {
99       "target": [
100         "Overview",
101         "NotFound"
102       ]
103     }
104 },
105 }
```

Figure 99: Routing Configuration - Config Section - Example

The default value for `routerClass` is `sap.ui.core.routing.Router`. In this example, the `routerClass` is set to `sap.m.routing.Router` because we implement an app based on `sap.m`. Compared to `sap.ui.core.routing.Router` the `sap.m.routing.Router` is optimized for mobile apps and adds the properties `viewLevel`, `transition` and `transitionParameters` which can be specified for every route or target created by the `sap.m.routing.Router`. Please check the API Reference for more information:

- The `viewPath` property specifies the namespace to which the views are assigned to.
- The `controlId` specifies the id of the control that is used as a root control for displaying the UI-aspects. In an implementation using the `FlexibleColumnLayout-control` the `controlId` property contains the id of the `sap.f.FlexibleColumnLayout` controls that is embedded in the `sap.m.App-control`.
- The `controlAggregation` attribute contains the name of the default aggregation of the control defined by the `controlId` property. The specified aggregation will be filled when a new UI-aspect is displayed.

Routing Configuration - Routes Section - Example



```

manifest.json x
flexiblecolumnlayout > webapp > manifest.json > sap.ui5 > routing >
104   },
105   "routes": [
106     {
107       "name": "Overview",
108       "pattern": "",
109       "titleTarget": "",
110       "greedy": false,
111       "target": ["Overview"]
112     },
113     {
114       "name": "Carrier",
115       "pattern": "carriers/{carrierId}",
116       "titleTarget": ,
117       "greedy": false,
118       "target": ["Carrier"]
119     }
]

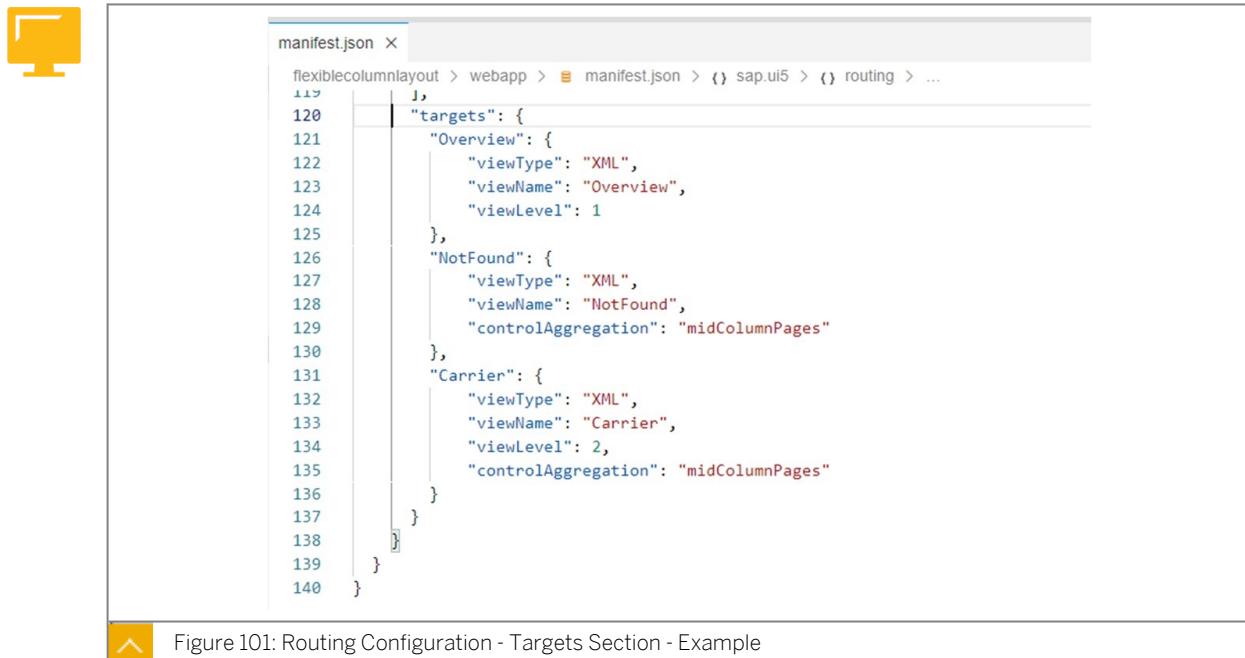
```

Figure 100: Routing Configuration - Routes Section - Example

(1) The empty pattern matches the empty hash. Unique route name that is used to navigate to a certain route or to determine the matched route in one of the matched handlers. Array of targets from the targets section that will be displayed when the route has been matched.

(2) Route that expects a mandatory objectId in its pattern to address a product.

Routing Configuration - Targets Section - Example



The screenshot shows a code editor with the file 'manifest.json' open. The code defines targets for routes:

```

119     ],
120     "targets": {
121       "Overview": {
122         "viewType": "XML",
123         "viewName": "Overview",
124         "viewLevel": 1
125       },
126       "NotFound": {
127         "viewType": "XML",
128         "viewName": "NotFound",
129         "controlAggregation": "midColumnPages"
130       },
131       "Carrier": {
132         "viewType": "XML",
133         "viewName": "Carrier",
134         "viewLevel": 2,
135         "controlAggregation": "midColumnPages"
136       }
137     }
138   }
139 }
140 }

```

A yellow icon of a computer monitor is visible on the left side of the editor.

Figure 101: Routing Configuration - Targets Section - Example

A target can be assigned to one or more routes, but it's not necessary. Targets can also be displayed directly in the app without hitting a route.

Each target has a unique key that is used to identify a target.

When the target is called the viewName and the viewPath (namespace) configured in the routing config-object is concatenated and loaded.

The *viewLevel* is relevant for flip and slide transitions. It helps the router to determine the direction of the transition from one page to another.

The *controlAggregation* at the target level will overwrite the *controlAggregation* property at routing config-object for this specific target.

Initializing the Router



- The *Router* needs to be initialized by the *Component*: In the *init()* method of *Component.js*, get a reference to the *Router* and call *initialize()* on it.
- After initialization, the routing configuration in *manifest.json* is automatically enabled in the app.
- Initializing the *Router* will evaluate the current URL and load the corresponding views automatically.

```
Component.js ×
flexiblecolumnlayout > webapp > Component.js > ...
1  sap.ui.define([
2    "sap/ui/core/UIComponent",
3    "sap/ui/Device",
4    "student00/sap/training/flexiblecolumnlayout/model/models",
5    "student00/sap/training/flexiblecolumnlayout/controller>ListSelector"
6  ], function (UIComponent, Device, models, ListSelector) {
7    "use strict";
8
9    return UIComponent.extend("student00.sap.training.flexiblecolumnlayout.Component", {
10
11      metadata: {
12        manifest: "json"
13      },
14
15      /**
16       * The component is initialized by UI5 automatically during the startup of the app and calls the init method once.
17       * @public
18       * @override
19       */
20      init: function () {
21        this.oListSelector = new ListSelector();
22        // call the base component's init function
23        UIComponent.prototype.init.apply(this, arguments);
24        // enable routing
25        this.getRouter().initialize();
26        // set the device model
27        this.setModel(models.createDeviceModel(), "device");
28      }
29    });
30  });
```

```

Figure 102: Initializing the Router

To activate the routing you have to instantiate the Router manually, it is automatically instantiated based on the configuration in *manifest.json* and assigned to the Component.

Please be aware of the fact, that the SAPUI5 template of the SAP Business Application Studio does not add the marked line to the generated code.

## The Master View

### The sap.f.DynamicPage

The *DynamicPage* control, representing a web page, consisting of a title, header with dynamic behavior, a content area, and an optional floating footer.

#### Overview

The control consist of several components:

- *DynamicPageTitle* - consists of a heading on the left side, content in the middle, and actions on the right. The displayed content changes based on the current mode of the *DynamicPageHeader*.
- *DynamicPageHeader* - a generic container, which can contain a single layout control and does not care about the content alignment and responsiveness. The header works in two modes - expanded and snapped and its behavior can be adjusted with the help of different properties.
- *Content area* - a generic container, which can have a single UI5 layout control and does not care about the content alignment and responsiveness.

- Footer - positioned at the bottom with a small offset and used for additional actions, the footer floats above the content. It can be any `sap.m.IBar` control.

## Usage

Use the `DynamicPage` if you need to have a title, that is always visible and a header, that has configurable Expanding/Snapping functionality. If you don't need the Expanding/Snapping functionality it is better to use the `sap.m.Page` as a lighter control.

If you're displaying a `sap.m.FlexBox` with non-adaptive content (doesn't stretch to fill the available space), it is recommended to set the `fitContainer` property of the `FlexBox` to `false`. If you are displaying a `sap.ui.table.Table`, keep in mind that it is non-adaptive and may cause unpredicted behavior for the `DynamicPage` on smaller screen sizes, such as mobile.

Snapping of the `DynamicPageTitle` is not supported in the following case: When the `DynamicPage` has a scroll bar, the control usually scrolls to the snapping point - the point, where the `DynamicPageHeader` is scrolled out completely. However, when there is a scroll bar, but not enough content to reach the snapping point, the snapping is not possible using scrolling.

When using `sap.ui.layout.form.Form`, `sap.m.Panel`, `sap.m.Table` and `sap.m.List` controls in the content of `DynamicPage`, you need to adjust their left text offset if you want to achieve vertical alignment between the `sap.f.DynamicPageHeader`'s content and `DynamicPage`'s content. For more information, see the documentation for the content aggregation.

## Responsive Behavior

The responsive behavior of the `DynamicPage` depends on the behavior of the content that is displayed. To adjust the `DynamicPage` content padding, the `sapUiContentPadding`, `sapUiNoContentPadding`, and `sapUiResponsiveContentPadding` CSS classes can be used.



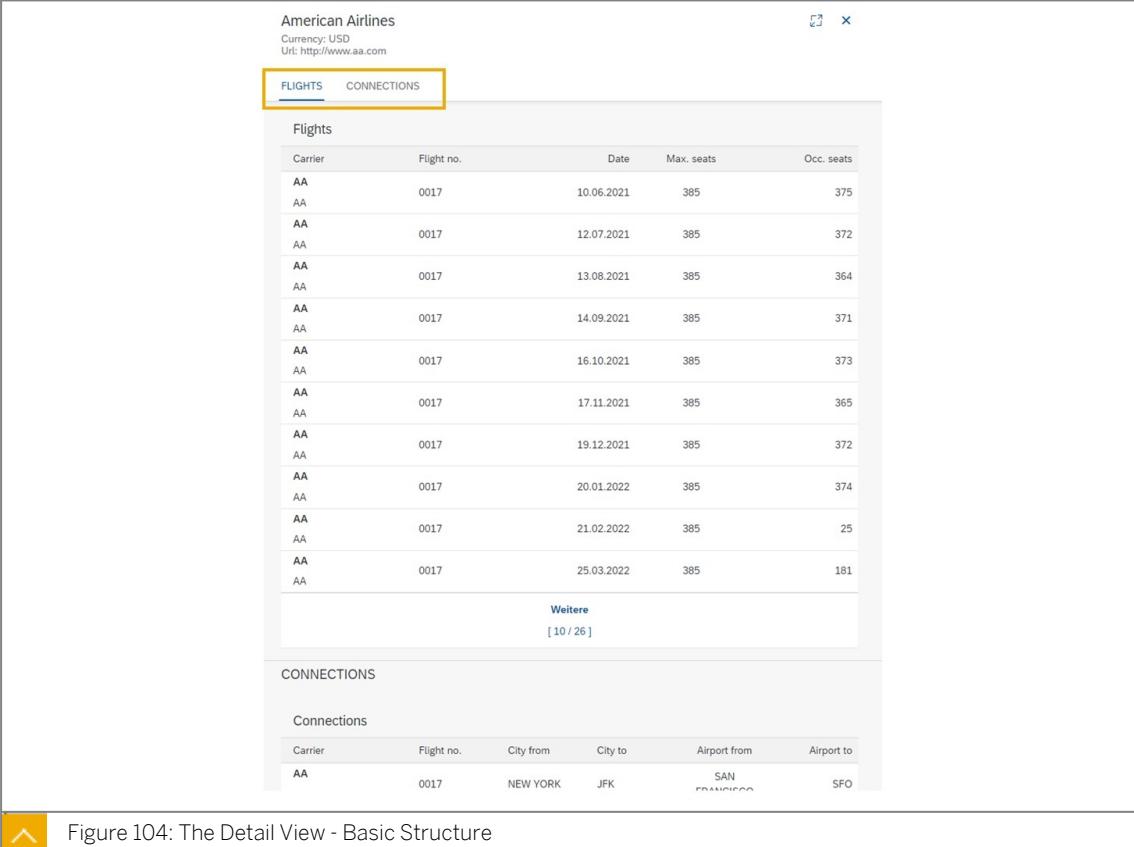
Figure 103: The Master View

```

Overview.view.xml <
flexiblecolumnlayout > webapp > view > Overview.view.xml
1 <mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
2 controllerName="student00.san_training.flexiblecolumnlayout.controller.Overview">
3 <f:DynamicPage id="dynamicPageOverviewId" headerExpanded="true" toggleHeaderOnTitleClick="true">
4 <f:content>
5 <Table items="{{/UX_C_Carrier_TP}}"
6 mode="{{= ${device}/system/phone} ? 'None' : 'SingleSelectMaster'}"
7 selectionChange="onSelectionChange" id="idCarrierList">
8 <headerToolbar>
9 <Toolbar>
10 <content>
11 <Title text="{i18n>overviewPageTitle}" level="H2"/>
12 </content>
13 </Toolbar>
14 </headerToolbar>
15 <columns>
16 <Column>
17 <Text text="{{i18n>columnId}}"/>
18 </Column>
19 <Column>
20 <Text text="{{i18n>columnName}}"/>
21 </Column>
22 </columns>
23 <items>
24 <ColumnListItem press="onPress" type="Navigation">
25 <cells>
26 <ObjectIdentifier title="{{Carrid}}"/>
27 <Text text="{{Carrname}}"/>
28 </cells>
29 </ColumnListItem>
30 </items>
31 </Table>
32 </f:content>
33 </f:DynamicPage>
34 </mvc:View>

```

## The Detail View - Basic structure



The screenshot shows a SAP UI5 application window titled "American Airlines". The header contains the carrier name, currency (USD), and URL (http://www.aa.com). Below the header is an anchor bar with two tabs: "FLIGHTS" (selected) and "CONNECTIONS". The "FLIGHTS" section displays a table of flight information with columns: Carrier, Flight no., Date, Max. seats, and Occ. seats. The "CONNECTIONS" section shows a single connection entry for flight 0017 from New York (JFK) to San Francisco (SFO). A navigation bar at the bottom indicates "Weitere [ 10 / 26 ]".

## The sap.uxap.ObjectPageLayout

The *ObjectPageLayout* layout is composed of a header (title and content), an optional anchor bar and block content wrapped in sections and subsections that structure the information.

### Structure

An *ObjectPageLayout* control is used to put together all parts of an Object page - Header, optional Anchor Bar and Sections/Subsections.

### Header

The *ObjectPageLayout* implements the snapping header concept. This means that the upper part of the header (Header Title) always stays visible, while the lower part (Header Content) can scroll out of view.

Header Title is displayed at the top of the header and always remains visible above the scrollable content of the page. It contains the title and most prominent details of the object.

The Header Content scrolls along with the content of the page until it disappears (collapsed header). When scrolled back to the top it becomes visible again (expanded header). It contains all the additional information of the object.

### Anchor Bar

The Anchor Bar is an automatically generated internal menu that shows the titles of the sections and subsections and allows the user to scroll to the respective section and subsection content.

### Sections, Subsections, Blocks

The content of the page that appears below the header is composed of blocks structured into sections and subsections.

## Usage

Use the **ObjectPageLayout** if:

- The users need to see, edit, or create an item with all its details.
- Users need to get an overview of an object and interact with different parts of the object.

## The Detail View - Implementation Example



```

Carrier.view.xml ×
dynamicpage > webapp > view > Carrier.view.xml
 2 xmlns:uxap="sap.uxap" controllerName="student00.sap.training.dynamicpage.controller.Carrier"
 3 xmlns:html="http://www.w3.org/1999/xhtml"
 4 <uxap:ObjectPageLayout id="dynamicPageId" showTitleInHeaderContent="true" alwaysShowContentHeader="false"
 5 preserveHeaderStateOnScroll="false" headerContentPinnable="true" isChildPage="true" enableLazyLoading="false">
 6 <uxap:headerTitle>
 7 <uxap:objectPageDynamicHeaderTitle>
 8 <uxap:expandedHeading>
 9 <Title text="{Carrname}" />
10 </uxap:expandedHeading>
11 <uxap:snappedHeading>
12 <Title text="{Carrname}" />
13 </uxap:snappedHeading>
14 </uxap:expandedContent>
15 <FlexBox alignItems="Start" justifyContent="SpaceBetween">
16 <items>
17 <layout:HorizontalLayout allowWrapping="true">
18 <layout:VerticalLayout class="sapUiMediumMarginEnd">
19 <ObjectAttribute title="{i18n:currLabelText}" text="{CurrCode}" />
20 <ObjectAttribute title="{i18n:urllabelText}" text="{Url}" />
21 </layout:VerticalLayout>
22 </layout:HorizontalLayout>
23 </items>
24 </FlexBox>
25 </uxap:expandedContent>
26 <uxap:navigationActions>
27 <OverflowToolbarButton type="Transparent" icon="sap-icon://full-screen"
28 tooltip="Enter Full Screen Mode" id="idExitFullscreen"
29 visible="{! ${device/system/phone} && ${appView/actionButtonsInfo/midColumn/fullScreen}}"
30 press="onToggleFullscreen"/>
31 <OverflowToolbarButton type="Transparent" icon="sap-icon://full-screen" tooltip="Enter Full Screen Mode"
32 visible="{! ${device/system/phone} && ${appView/actionButtonsInfo/midColumn/fullScreen}}"
33 press="onToggleFullscreen"/>
34 <OverflowToolbarButton type="Transparent" press="onCloseDetailPress" icon="sap-icon://decline"
35 tooltip="Close column"/>
36 </uxap:navigationActions>
37 <uxap:objectPageDynamicHeaderTitle>
38 </uxap:headerTitle>
39 <uxap:sections>
40 <uxap:objectPageSection title="{i18n:tblFlights}">
41 <uxap:subSections>
42 <uxap:objectPageSubSection>
43 <uxap:blocks>
44 <Table id="idFlights" items="{ path: 'to_Flight', sorter: { path: 'Carrid' }}"
45 growing="true" growingThreshold="10" visible="true">
46 <headerToolbar>
47 <OverflowToolbar>
48 <Title text="{i18n:tblFlights}" level="H2"/>

```

Figure 105: The Detail View - Implementation Example

The figure gives an example of the used code.

## Device Adaptation

SAPUI5 applications can be run with phone, tablet, and desktop devices and you can configure the application to make best use of the screen estate for each scenario.

## Device Model

 By introducing a *device model*, you can bind control properties to the device's capabilities:



```

models.js x
1 sap.ui.define([
2 "sap/ui/model/json/JSONModel",
3 "sap/ui/Device"
4], function (JSONModel, Device) {
5 "use strict";
6
7 return {
8
9 createDeviceModel: function () {
10 var oModel = new JSONModel(Device);
11 oModel.setDefaultBindingMode("OneWay");
12 return oModel;
13 }
14 };
15 });
16 });

Component.js x
1 sap.ui.define([
2 "sap/ui/core/UIComponent",
3 "sap/ui/Device"
4], ["student00/com.sap.training.ux402/masterdetail/ux402_masterdetail/model/models",
5 "student00/com.sap.training.ux402/masterdetail/ux402_masterdetail/controller/ux402Controller"
6], function (UIComponent, Device, models, ListSelector) {
7 "use strict";
8
9 return UIComponent.extend("student00.com.sap.training.ux402.masterdetail.Component", {
10 metadata: {
11 ...
12 },
13 /**
14 * ...
15 */
16 init: function () {
17 // call the base component's init function
18 UIComponent.prototype.init.apply(this, arguments);
19
20 // instantiation of the listselector
21 this.oListSelector = new ListSelector();
22
23 // enable routing
24 this.getRouter().initialize();
25
26 // set the device model
27 this.setModel(models.createDeviceModel(), "device");
28 }
29 });
30 });
31 });
32 });
33 });
34 });

```

Figure 106: Device Model

The SAPUI5 template of SAP Business Application Studio generates the code of setting up a device model for you.

The above code shows you the code generated by SAP Business Application Studio.

The main part of creating the device model is generated in a file called `models.js`. This file is added to a folder of your project called `model`.

The implementation of the `model.js` has dependencies to `sap.ui.model.json.JSONModel` and `sap.ui.Device`.

The function `createDeviceModel` creates an object of the type `JSONModel` and passes the reference to the `sap.ui.Device` object to the `JSONModel` instance.

The `JSONModel` is of binding mode `OneWay`.

The `createDeviceModel` function is called by the *init-function* of the `Component.js` implementation. The result of the invocation is stored in a model with id `device`.

## BaseController

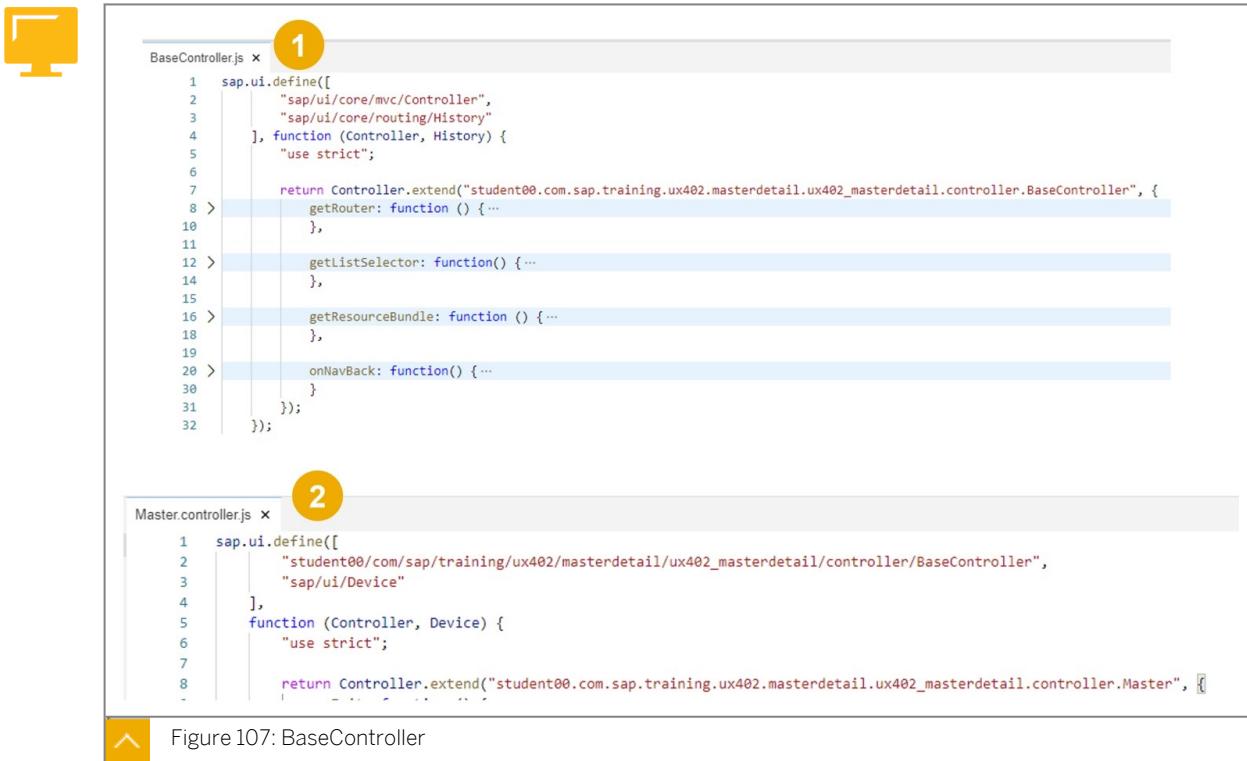


Figure 107: BaseController

```

sap.ui.define([
 "sap/ui/core/mvc/Controller",
 "sap/ui/core/routing/History"
], function (Controller, History) {
 "use strict";

 return Controller.extend("student00.com.sap.training.ux402.masterdetail.ux402_masterdetail.controller.BaseController", {
 getRouter: function () { ... },
 getListSelector: function() { ... },
 getResourceBundle: function () { ... },
 onNavBack: function() { ... }
 });
});

```

```

sap.ui.define([
 "student00/com/sap/training/ux402/masterdetail/ux402_masterdetail/controller/BaseController",
 "sap/ui/Device"
],
function (Controller, Device) {
 "use strict";

 return Controller.extend("student00.com.sap.training.ux402.masterdetail.ux402_masterdetail.controller.Master", [
 ...
]);
}
);

```

To reuse functionality across view controllers, it is a common practice to introduce a base controller implementation. The base controller is implemented in a standard js-file (1).

Each controller (for example the master list controller) will be derived from the base controller implementation (2).

## Master List

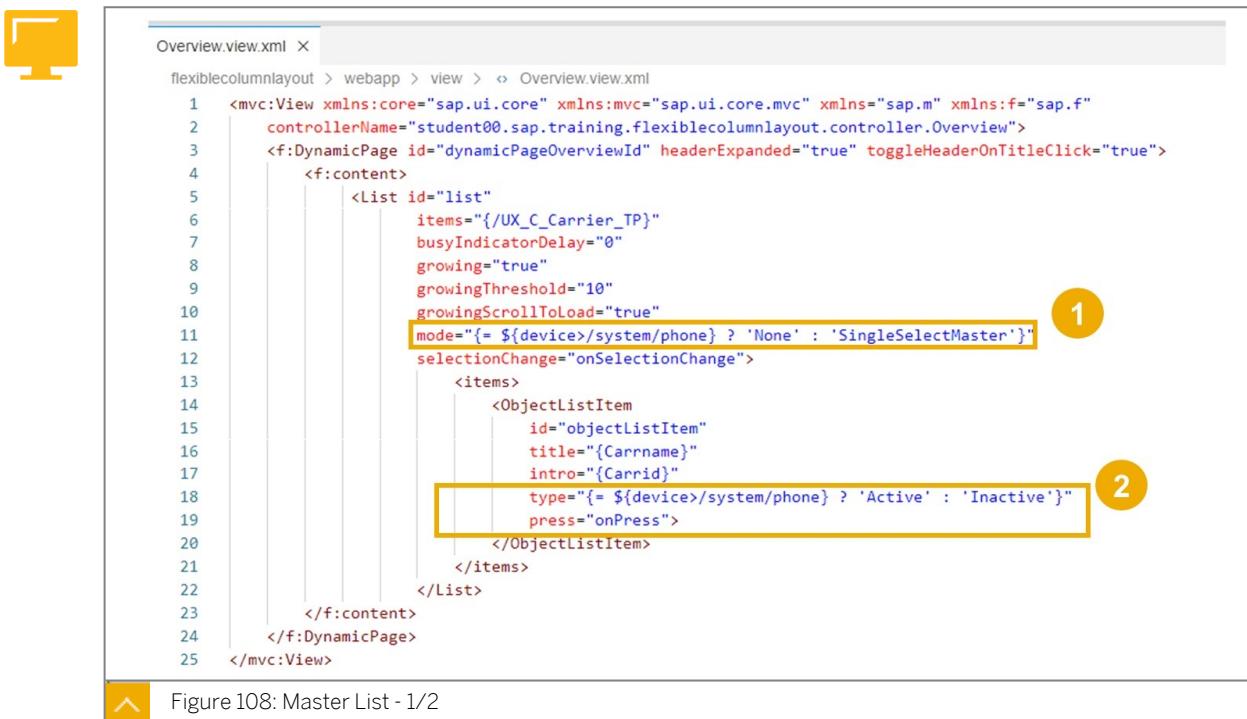


Figure 108: Master List - 1/2

```

<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.f"
 controllerName="student00.sap.training.flexiblecolumnlayout.controller.Overview">
 <f:DynamicPage id="dynamicPageOverviewId" headerExpanded="true" toggleHeaderOnTitleClick="true">
 <f:content>
 <List id="list"
 items="{/UX_C_Carrier_TP}"
 busyIndicatorDelay="0"
 growing="true"
 growingThreshold="10"
 growingScrollToLoad="true"
 mode="{${device}/system/phone} ? 'None' : 'SingleSelectMaster'"
 selectionChange="onSelectionChange">
 <items>
 <ObjectListItem
 id="objectListItem"
 title="{Carrname}"
 intro="{Carrid}">
 <!-- Annotation 1 -->
 type="{${device}/system/phone} ? 'Active' : 'Inactive'"
 press="onPress">
 </ObjectListItem>
 </items>
 </List>
 </f:content>
 </f:DynamicPage>
</mvc:View>

```

The List's mode property and the *ObjectList/Item*'s type property control how items are selectable.

On a phone device, the selection should be via the item (list mode "None", list item type "Active"), otherwise it should be via the List itself (list mode "SingleSelectMaster", list item type "Inactive").

The *selectionChange* event is only fired on a desktop or on a tablet device in the landscape mode. The event handler function will have access to the selected item by the *listItem* event parameter.

On a phone or tablet in portrait mode the event press on the selected list item will be thrown. The event handler function has access to the selected item using the *getSource* method on the event object provided to the event handler function.

The necessary case distinction can be implemented by means of the device model and the expression binding syntax.

```

sap.ui.define([
 "sap/ui/core/mvc/Controller"
], /**
 * @param {typeof sap.ui.core.mvc.Controller} Controller
 */
function (Controller) {
 "use strict";

 return Controller.extend("student00.sap.training.flexiblecolumnlayout.controller.Overview", {
 onInit : function() {
 },
 getRouter: function () {
 return sap.ui.core.UIComponent.getRouterFor(this);
 },
 onPress: function (oEvent) {
 var oItem, oCtx, sCarrid;
 oItem = oEvent.getSource();
 oCtx = oItem.getBindingContext();
 sCarrid = oCtx.getProperty("Carrid");
 this._showCarrierDetails(sCarrid);
 },
 onBypassed : function () {
 },
 onMasterMatched : function() {
 },
 onSelectionChange : function(oEvent) {
 var oItem, oCtx, sCarrid;
 oItem = oEvent.getParameter("listItem");
 oCtx = oItem.getBindingContext();
 sCarrid = oCtx.getProperty("Carrid");
 this._showCarrierDetails(sCarrid);
 },
 _showCarrierDetails : function(sCarrid) {
 var oRouter = this.getRouter();
 oRouter.navTo("Carrier", {
 carrierId: sCarrid
 }, false /*with history*/);
 }
 });
}

```

Figure 109: Master List - 2/2

- (1) Add a dependency to *sap.ui.Device* (device and feature detection API of SAPUI5).
- (2) Event handler for the list selection event: Get the list item, either from the *listItem* parameter or from the event's source itself (depends on the list mode and the list item type).
- (3) Read *AirLineID* from the selected item on the detail page; On phones an additional history entry is created.
- (4) Trigger the navigation the details view for carriers. Therefore get a reference to the router using the *getRouter* function inherited by the base controller.

## Detect the Carrier in the Detail View Controller

The screenshot shows the SAP Studio interface with the file 'Carrier.controller.js' open. The code implements a detail view controller for a carrier. It includes methods for getting the router, handling object matching, and binding elements to a view. Three specific sections of the code are highlighted with yellow boxes and numbered 1, 2, and 3.

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

Figure 110: Detect the Carrier in the Detail View Controller

In the Detail View Controller, it is necessary to detect which carrier shall be displayed in order to show the carrier's data in the view.

The ODataModel will handle the necessary data requests to the backend in the background. While the data is loading, it would be nice to show a busy indicator by simply setting the view to busy. Therefore we pass an events object to `bindElement()` to listen to the events `dataRequested` and `dataReceived`. The attached functions handle the busy state by calling `oView.setBusy(true)` and `oView.setBusy(false)` respectively.

- (1) Query the Router for the route object from the `BaseController`'s `getRouter`-method and attach a private event listener function `_onObjectMatched` to the matched event of this route.
- (2) Access the arguments parameter from the `oEvent` parameter that contains all parameters of the pattern, that is, the mandatory parameter `objectId` is available as a key in `arguments`.
- (3) Call `bindElement()` on the view to make sure that the data of the specified product is available in the view and its controls.

## Synchronizing the Master and the Detail View

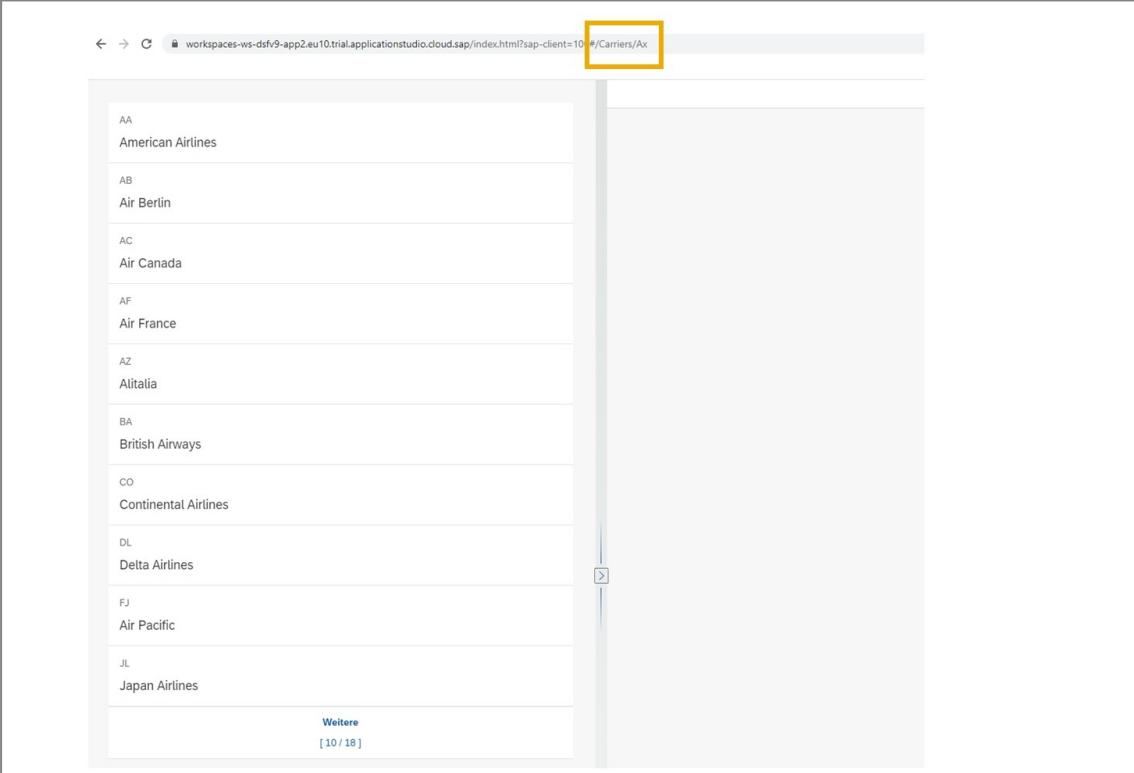


Figure 111: Synchronizing the Master and the Detail View - 1/3

If the user enters an invalid carrier id in the browser, he should get an indication that the requested object was not found.

Furthermore you should remove a potentially existing selection in the master list in this case.

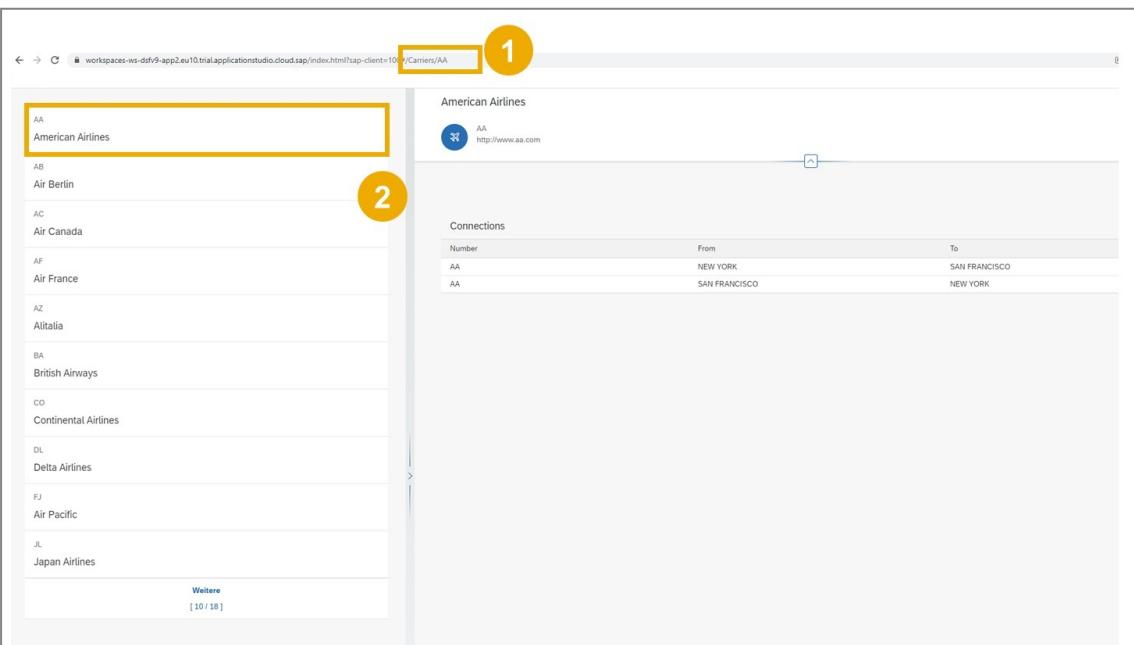


Figure 112: Synchronizing the Master and the Detail View - 2/3

If the user enters a valid carrier id in the browser (1), you should make sure that the corresponding list item is selected in the master list (2).

### To synchronize the Master view with the Detail view:



- The Detail view controller needs access to the master list.
- For that purpose, we expose the master list using a wrapper object named *ListSelector* which provides a convenience API for working with the master list.

| ListSelector                    |
|---------------------------------|
| _oList: sap.m.List              |
| _oWhenListHasBeenSet: Promise   |
| oWhenListLoadingIsDone: Promise |
| setBoundMasterList(oList)       |
| selectAListItem(sBindingPath)   |
| clearMasterListSelection()      |

The following are further details:

- The view controllers get access to the *ListSelector* via the Component (*Component.js*).
- setBoundMasterList()* is used by the Master view controller to pass the master list, it sets the list on which the other functions will be invoked
- selectAListItem* tries to select a list item using the passed binding path
- clearMasterListSelection* will remove selections from the master list e.g. when an invalid hash is provided.
- As you cannot foresee the order in which the methods will be called, the execution of *selectAListItem()* and *clearMasterListSelection()* has to wait until the list is set by *setBoundMasterList()*. This can be achieved by of JavaScript Promises.

### JavaScript Promises

The following are further information about promises:

- A promise is an object that may produce a single value some time in the future.
- A Promise can be in one of 3 states:
  - Pending** - the Promise's outcome hasn't yet been determined, because the asynchronous operation that will produce its result hasn't completed yet.
  - Fulfilled** - the asynchronous operation has completed, and the Promise has a value.
  - Rejected** - the asynchronous operation failed, and the Promise will never be fulfilled. In the rejected state, a Promise has a reason that indicates why the operation failed.
- Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.
- When a Promise is pending, it can transition to the fulfilled or rejected state. Once a Promise is fulfilled or rejected, however, it will never transition to any other state, and its value or failure reason will not change.

Promises are useful for async success/failure, because you're less interested in the exact time something became available, and more interested in reacting to the outcome.

### Creating a Promise



The *Promise* constructor takes one argument, a callback with two parameters, *fnResolve* and *fnReject* (*fnReject* is optional).

```
samplePromise : function() {
 var oPromise = new Promise(function(fnResolve, fnReject) {
 if(this.bSuccessful) {
 fnResolve({sMsg:"It was successful"});
 } else {
 fnReject(Error("It was not successful"));
 }
 });

 return oPromise;
}
```



Figure 113: Creating a Promise

- Do something within the callback function, perhaps async, then call *fnResolve* if everything worked, otherwise call *fnReject*.

### Using Promises



- The primary way of interacting with a *Promise* is through its *then()* method, which registers callbacks to receive either a *Promise*'s eventual value or the reason why the *Promise* cannot be fulfilled.
- *then()* takes two arguments, a callback for a success case, and another for the failure case. Both are optional, so you can add a callback for the success or failure case only.
- If a *Promise* has succeeded or failed and you later add a success/failure callback, the correct callback will be called, even though the event took place earlier.

```
handleSomething : function() {
 var oPromise = this.samplePromise();
 oPromise.then(
 function(oResult) {
 jQuery.sap.log.info(oResult.sMsg);
 },function(oError) {
 jQuery.sap.log.error(oError);
 }
);
}
```



Figure 114: Using Promises

## ListSelector

```

ListSelector.js x
1 sap.ui.define([
2 "sap/ui/base/Object"
3], function(BaseObject) {
4
5 return BaseObject.extend("student@com.sap.training.ux402.masterdetail.ux402_masterdetail.controller.ListSelector", {
6
7 constructor: function() {
8 this._okWhenListHasBeenSet = new Promise(function(fnResolveListHasBeenCalled) {
9 this._fnResolveListHasBeenCalled = fnResolveListHasBeenCalled;
10 }.bind(this));
11 },
12
13
14 setBoundMasterList: function(oList) {
15 this._oList = oList;
16 this._fnResolveListHasBeenCalled(oList);
17 },
18
19
20
21 clearMasterListSelection: function() {
22 this._okWhenListHasBeenSet.then(function() {
23 this._oList.removeSelections(true);
24 }.bind(this));
25 }
26 });
27 });
28 });

```

Figure 115: ListSelector - 1/3

1. Add dependency to the base class of all SAPUI5 objects.
2. In the constructor function for the new class: define a Promise to make sure that the master list is available. All functions will wait until the initial load of the master list passed by the `setBoundMasterList()` function.
3. Fulfils the Promise and sets the master list as value of the Promise.
4. The function `clearMasterListSelection` removes all selections from the master list (uses the Promise to make sure that `this._oList` is available).



```

constructor: function() {
 this._oWhenListHasBeenSet = new Promise(function(fnResolveListHasBeenSet) {
 | this._fnResolveListHasBeenSet = fnResolveListHasBeenSet;
 }.bind(this));
}

1 this._oWhenListLoadingIsDone = new Promise(function(fnResolve, fnReject) {
2 this._oWhenListHasBeenSet
 .then(function(oList) {
 oList.getBinding("items").attachEventOnce("dataReceived",
 function(oData) {
 if (!oData.getParameter("data")) {
 fnReject({
 list: oList,
 error: true
 });
 }
 var oFirstListItem = oList.getItems()[0];
 if (oFirstListItem) {
 fnResolve({
 list: oList,
 firstListItem: oFirstListItem
 });
 } else {
 fnReject({
 list: oList,
 error: false
 });
 }
 });
 });
 }.bind(this));
},

```

Figure 116: ListSelector - 2/3

- (1) Define a second Promise in the constructor function to make sure that the master list is not only set but also filled with data.
- (2) Wait until the `setBoundMasterList()` function is invoked.
- (3) Reject the Promise in case data cannot be received from the model.
- (4) Fulfill the Promise in case the first list item can be selected.
- (5) Reject the Promise in case there are no items in the list.



```

1 selectAListItem: function(sBindingPath) {
2 this._oWhenListLoadingIsDone.then(
3 function() {
4 var oList = this._oList,
 oSelectedItem;
4
4 if (oList.getMode() === "None") {
4 return;
4 }
4
4 oSelectedItem = oList.getSelectedItem();
4
4 if (oSelectedItem && oSelectedItem.getBindingContext().getPath() === sBindingPath) {
4 return;
4 }
4
4 oList.getItems().some(function(oItem) {
4 if (oItem.getBindingContext() && oItem.getBindingContext().getPath() === sBindingPath) {
4 oList.setSelectedItem(oItem);
4 return true;
4 }
4 });
 }.bind(this));
},

```

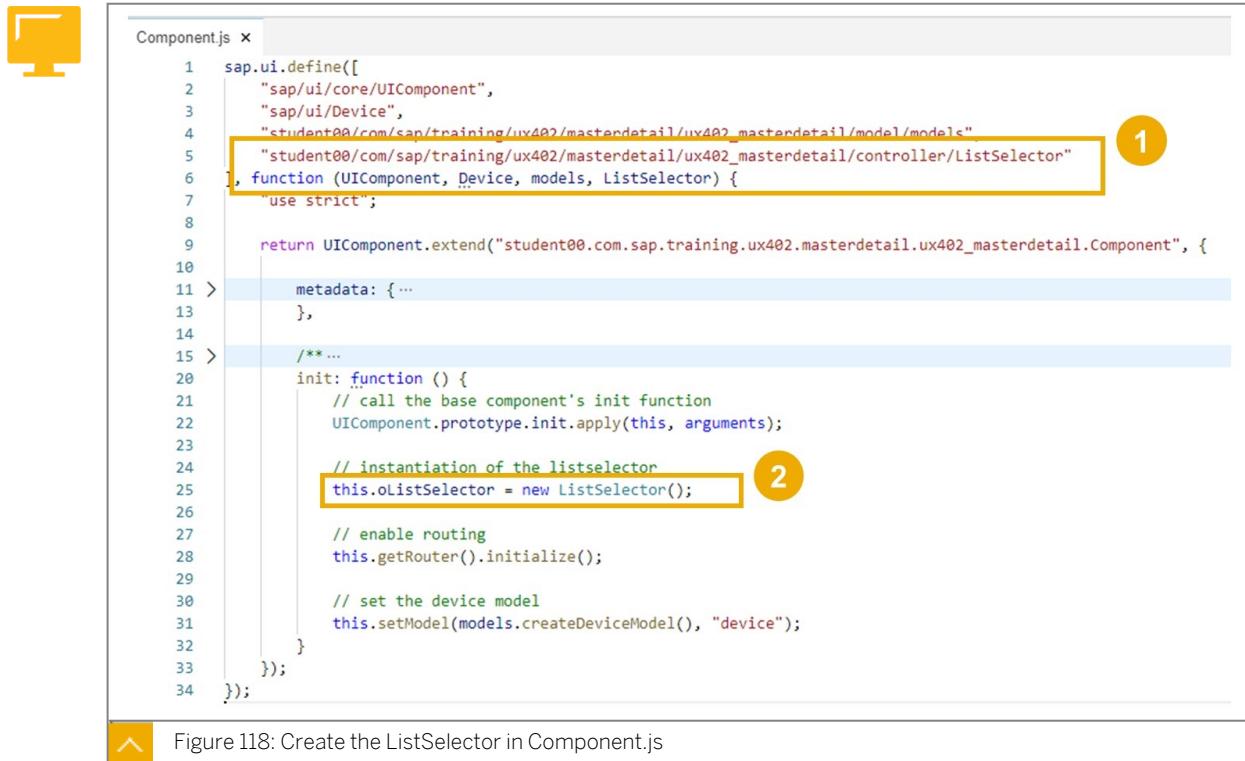
Figure 117: ListSelector - 3/3

The function `selectAListItem()` tries to select a list item with a matching binding path.

- (1) Wait until the master list is filled with data.

- (2) On a phone device, no selection will happen.
- (3) Skip update if the current selection is already matching the binding path.
- (4) Find a list item with a matching binding path and select it.

### Create the ListSelector in Component.js



```

Component.js x
1 sap.ui.define([
2 "sap/ui/core/UIComponent",
3 "sap/ui/Device",
4 "student00/com/sap/training/ux402/masterdetail/ux402_masterdetail/model/models"
5 "student00/com/sap/training/ux402/masterdetail/ux402_masterdetail/controller/ListSelector"
6],
7 function (UIComponent, Device, models, ListSelector) {
8 "use strict";
9
10 return UIComponent.extend("student00.com.sap.training.ux402.masterdetail.ux402_masterdetail.Component", {
11 metadata: {...},
12
13 /**
14 * ...
15 */
16 init: function () {
17 // call the base component's init function
18 UIComponent.prototype.init.apply(this, arguments);
19
20 // instantiation of the listselector
21 this.oListSelector = new ListSelector();
22
23 // enable routing
24 this.getRouter().initialize();
25
26 // set the device model
27 this.setModel(models.createDeviceModel(), "device");
28
29 }
30 });
31 });
32});
33 });
34 });

```

Figure 118: Create the ListSelector in Component.js

1. Add a dependency to the *ListSelector*.
2. Create a *ListSelector* instance and assign it to a member variable of the component.

### Pass the Master List to the ListSelector



```

Master.controller.js x
1 > sap.ui.define([
2],
3 function (Controller, Device) {
4 "use strict";
5
6 return Controller.extend("student00.com.sap.training.ux402.masterdetail.ux402_masterdetail.controller.Master", [
7 {
8 onInit: function () {
9 var oList = this.byId("list");
10 this._oList = oList;
11 this.getListSelector().setBoundMasterList(oList);
12 this.getRouter().getRoute("master").attachPatternMatched(this._onMasterMatched, this);
13 this.getRouter().attachBypassed(this.onBypassed, this);
14 },
15 }
16 });
17 });

```

Figure 119: Pass the Master List to the ListSelector

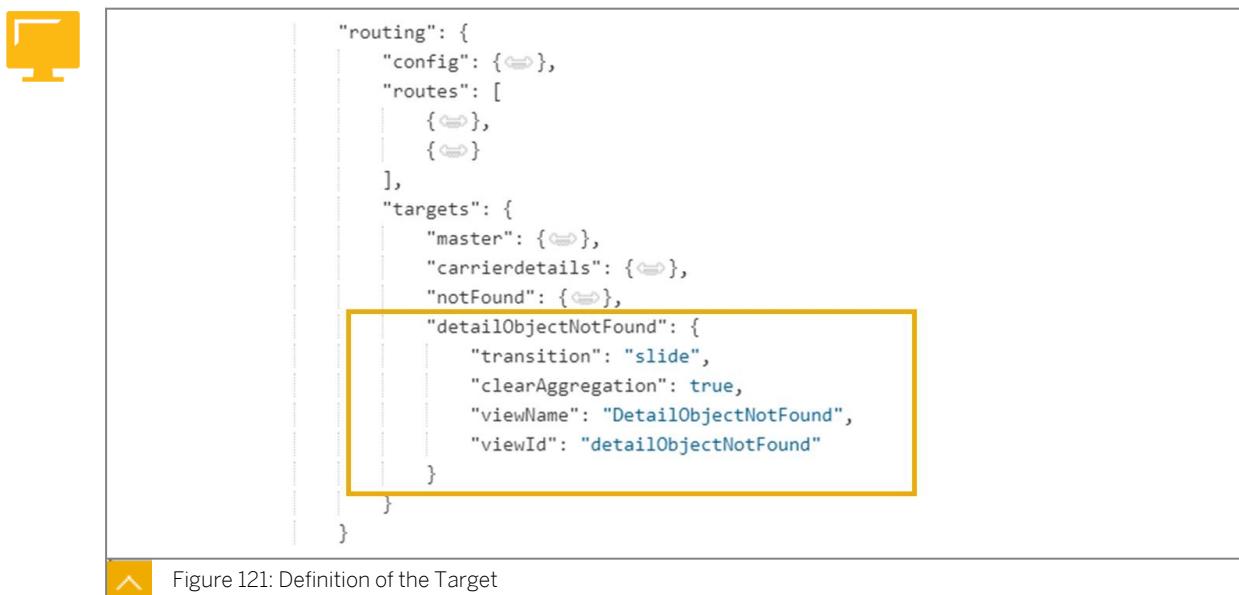
In the *onInit()* method of the Master view controller, pass the master list to the *ListSelector* instance.

**Use the ListSelector for Synchronizing Master and Detail view**



- (1) After the binding context changed call the private `_onBindingChange` function. Make sure that the reference is correct in the execution context of `_onBindingContext`.
  - (2) If the user enters an invalid product id in the browser the error page is displayed and a potentially existing selection is removed.
  - (3) If a valid product id is entered the corresponding list item is selected in the master list using the `selectAListItem` function from the `ListSelector`.

## Definition of the Target



- If the user enters an invalid product id in the browser, the target `detailObjectNotFound` shall be displayed.
- This target has to be defined in the routing section of `manifest.json`.

### Implementing DetailObjectNotFound view

The screenshot shows two code files in SAP Studio:

- DetailObjectNotFound.view.xml** (top):
 

```

1 <mvc:View controllerName="student00.com.sap.training.ux402.masterdetail.ux402_masterdetail.controller.DetailObjectNotFound"
2 xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m">
3 <MessagePage
4 title="{i18n>detailTitle}"
5 text="{i18n>noObjectFoundText}"
6 description=""
7 showNavButton="{device>/system/phone}"
8 navButtonPress="onNavBack">
9 </MessagePage>
10 </mvc:View>

```

A yellow box highlights the `showNavButton` and `navButtonPress` properties.
- BaseController.js** (bottom):
 

```

19
20 onNavBack: function() {
21 var sPreviousHash = History.getInstance().getPreviousHash();
22 if (sPreviousHash !== undefined) {
23 // The history contains a previous entry
24 history.go(-1);
25 } else {
26 // Otherwise we go backwards with a forward history
27 var bReplace = true;
28 this.getRouter().navTo("master", {}, bReplace);
29 }
30 });
31 });
32 });

```

A yellow arrow points from the `onNavBack` method in the XML file to the corresponding implementation in the JavaScript file.

Figure 122: Implementing DetailObjectNotFound view

The implementation of `DetailObjectNotFound` view uses the `MessagePage-Element`. The `MessagePage` is displayed when there is no data or matching content. There are different use cases where a `MessagePage` might be visualized, for example: - The search query returned no results - The app contains no items - There are too many items - The application is loading. The layout is unchanged but the text varies depending on the use case. To make sure, that the back button is only shown if the app is running on a phone device we are using the device model stored in the component.

When the back button is visualized it is necessary to implement an event handler for the `navButtonPress-event`. Usually this implementation is generalized inside the `BaseController` implementation of your app. The implementation checks if there is a previous hash value in the app history:

- If yes, redirect to this hash.
- If not, navigate to the master route without browser history entry.

### Preselection of the First List item

A very important aspect when implementing master detail is you have to make sure that the first item in the master list is preselected when the master route was hit (empty hash).

(1) Query the Router for the master route and attach a private event listener `function_onMasterMatched` to the matched event of this route.

(2) Wait until the master list is filled with data. When the promise is fulfilled.

(3) Show the first list item on the Detail view without browser history entry; The processed coding in the Detail view controller will make sure that the first list item is selected in the master list (see above).

### Catching Invalid Hashes

An important aspect of the routing implementation is to handle invalid hashes (1). The situation of an invalid hash may occur, when the user bookmarked a carrier, calls the bookmark later and the carrier is not valid anymore.

When an invalid hash was entered, it is important to remove any selection at the master list (2).

### Catching Invalid Hashes - Configuration

Using the `bypassed` property, you tell the Router to display the master target and the `notFound` target in case no route was matched to the current hash. The `notFound` target configures a `NotFound` view.

### NotFound View

The `NotFound` - view implementation is quite simple. Usually it contains a `MessagePage` element. The `MessagePage` is displayed when there is no data or matching content. There are different use cases where a `MessagePage` might be visualized, for example: - The search query returned no results - The app contains no items - There are too many items - The application is loading. The layout is unchanged but the text varies depending on the use case. To make sure, that the back button is only shown if the app is running on a phone device we are using the device model stored in the component.

When the back button is visualized it is necessary to implement an event handler for the `navButtonPress-event`. Usually this implementation is generalized inside the `BaseController` implementation of your app. The implementation checks if there is a previous hash value in the app history:

- If yes, redirect to this hash.
- If not, navigate to the master route without browser history entry.

### Remove Master List Selection

1. Attach event-handler to the `bypassed` event of the Router. The event will get fired, if none of the routes is matching.
2. If there was a product selected in the master list, that selection is removed.



### LESSON SUMMARY

You should now be able to:

- Implement a Master-Detail-Application

## Unit 2

### Lesson 7

# Working with Messages



#### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with Messages

#### Messages

##### Message Types



**UI messages** (client side) refer to a control; they are created from the input validation and by the application itself.

Product Id:  Enter a number above the minimum value of 1.

**Server messages** refer to a binding path; they are managed by the server and are changed every time the back end responds to a request.

! Resource not found for the segment 'Products'.



Figure 123: Message Types

Messages notify the user about specific states of the application. The above slide shows you what message types are known by SAPUI5.

#### Anatomy of a Message

A message contains at least the following information:



- ID: The ID is usually set automatically on message creation.
- Type: Possible types are error, warning, and info; see `sap.ui.core.MessageType` in the API reference.
- Target: Describes to what part of the application the message applies this could be, for example, a control property; if the target is empty, the message applies to the entire application.
- Message processor: Object that handles the message in the application.

- Message text: Text that is displayed to the user.

In addition to this information, a message can contain the following optional information:

- Description: Message long text.
- Code: Machine-readable version of the message; this is usually provided by the back end to allow to look-up details.
- Technical property: This property is set if a technical error occurs; it is usually set if an error code is sent from the back end, like a 404.
- Persistent property: If this property is set to true, the messages are not deleted automatically; the application handles the life cycle of these messages.

### Message Life Cycle

The **MessageManager** handles the message life cycle, which is different for UI and server messages:



- UI messages are added with a target referencing a control and its specific property. The messages are kept until a validation message for the property is created and assigned. If new data for the same property is received from the server, the UI messages are erased unless their persistent property is set to true.
- Server messages are kept until a message from the server for the same path arrives. The server always sends all messages for a specific target, which means that all current messages are replaced with the ones sent by the server.
- **MessageManager** can be accessed by calling  
`sap.ui.getCore().getMessageManager();`

### UI Messages



- The most common example for **UI messages** are validation messages.
- If a bound property has an assigned type, the validation can generate validation messages.
- The automatic message creation has to be activated.

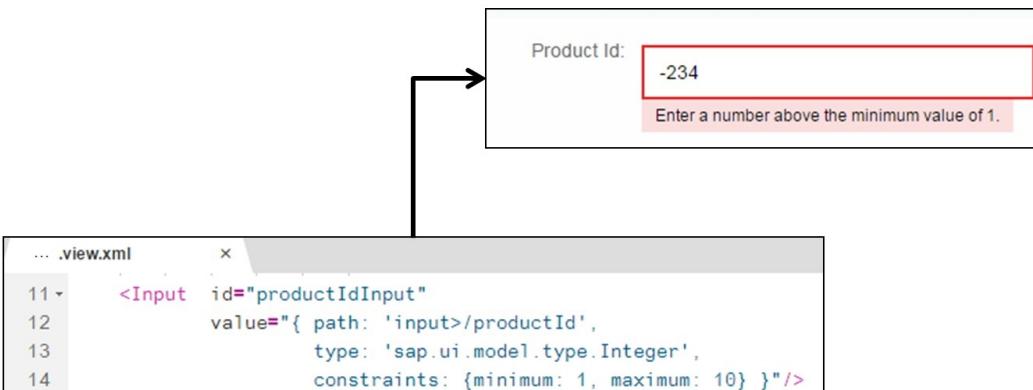


Figure 124: UI Messages

UI messages reference a control. They are handled and processed by the control message processor and are propagated to the message manager.

## Activating Automatic Message Creation



- To activate the automatic message creation for a Component, the following options exist:
- You can activate the automatic message generation in *manifest.json*:

```
manifest.json x
1 {
2 "_version": "1.7.0",
3 "sap.app": { },
17
18 "sap.ui": { },
39
40 "sap.ui5": {
41 "rootView": { },
45 "handleValidation": true,
46 "dependencies": { },
58 "contentDensities": { },
62 "models": { },
70 "resources": { }
75 }
76 }
```

Figure 125: Activating Automatic Message Creation - 1/3

Possible values: true or false (default); used to enable or disable validation handling by the message manager for this component.



- ... or as a parameter when instantiating the *Component*:

```
<body class="sapUiBody sapUiSizeCompact" id="content">
 <div
 data-sap-ui-component
 data-name="student00.com.sap.training.ux402.masterdetail.ux402masterdetail"
 data-id="container"
 data-settings='{"id" : "student00.com.sap.training.ux402.masterdetail.ux402masterdetail"}'
 data-handle-validation="true"
 ></div>
</body>
```

Figure 126: Activating Automatic Message Creation - 2/3

The figure shows the required code for activating the automatic message creation.



- You can also activate the automatic message creation for individual *Controls* by registering the control in the *message manager*:

```
onInit : function() {
 var oInput = this.getView().byId("idZipCode");
 sap.ui.getCore().getMessageManager().registerObject(oInput,true);
},
```

Figure 127: Activating Automatic Message Creation - 3/3

The message manager manages all messages for the application.

The message manager is available globally as a Singleton by calling `sap.ui.getCore().getMessageManager()`.

## UI Messages



- You can also create **UI messages** manually and add them to the *message manager*.

```
sap.ui.define([
 "sap/ui/core/mvc/Controller",
 "sap/ui/core/message/Message"
], function(Controller,Message) {
 "use strict";
 return Controller.extend("com.sap.ux402.demo.controller.DemoView", {
 _processMessage : function() {
 var oMessageManager = sap.ui.getCore().getMessageManager();
 var oMessageProcessor = new sap.ui.core.message.ControlMessageProcessor();
 oMessageManager.registerMessageProcessor(oMessageProcessor);

 var oZipField = this.getView().byId("idZipCode");

 oMessageManager.addMessage(
 new Message({
 message : "Invalid entry",
 description : "Zip codes must have at least 5 digits",
 type: sap.ui.core.MessageType.Error,
 target: oZipField.getId() + "/value",
 processor: oMessageProcessor
 })
);
 }
 });
});
```



Figure 128: Creating UI Messages

A message contains at least the following information:

- id: The ID is usually set automatically on message creation.
- type: Possible types are Error, Warning, Success, None, and Information; see `sap.ui.core.MessageType` in the API reference.
- target: Describes to what part of the application the message applies; if the target is empty, the message applies to the entire application; if a target is set, the target is a string consisting of a control ID, a slash ("/"), and the name of the property to which the message applies.
- processor: Object that handles the message in the application.
- message: Text that is displayed to the user.
- In addition to this information, a message can contain the following optional information:
  - description: Message long text.
  - code: Machine-readable version of the message; this is usually provided by the back end to allow to look-up details.
  - persistent: If this property is set to true, the messages are not deleted automatically; the application handles the life cycle of these messages.

The MessageManager handles the message life cycle, which is different for UI and server messages: UI messages are added with a target referencing a control and its specific property. If new data for the same property is received from the server, the UI messages are erased unless their persistent property is set to true. Server messages are kept until a message from the server for the same path arrives. The server always sends all messages for a specific target, which means that all current messages are replaced with the ones sent by the server.

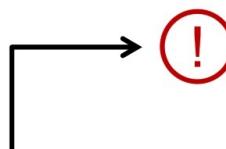
### Server Messages, Code



- The `v2.ODataModel` supports automatic parsing of server messages: `sap.ui.model.odata.ODataMessageParser` automatically parses message responses from the back-end.
- The parsed messages are propagated by `sap.ui.model.odata.v2.ODataModel`.

**Example of a standard OData error response:**

```
{
 "error": {
 "code": "",
 "message": {
 "lang": "en-US",
 "value": "Resource not found for the segment 'Products' ."
 }
 }
}
```



Resource not found for the segment "Products"



Figure 129: Server Messages

The figure shows the code for the message: Resource not found for the segment 'Products' (this is a variable).

Server messages are created by the message parser, then processed by the message processor and then propagated to the message manager.

The message parser is a simple interface that is implemented to allow the propagation of messages from back end services. For messages from OData services, the `sap.ui.model.odata.ODataMessageParser` is used.

The OData message parser is created automatically for all `sap.ui.model.odata.v2.ODataModel` instances and parses all responses from the server. The `ODataModel` implements the message processor interface and is used to propagate the messages to the message manager. In case of an error response, the response body is parsed for error messages. In case of a successful response, the "sap-message" header is parsed as a JSON-formatted error object. The name of the header field can be changed by calling the `setHeaderField()` method on the `ODataMessageParser`.

### Implementing Your Own Message Parser

If you have your own service implementation, for example, a JSON-based back end that also sends messages, you can implement your own message parser by implementing the `sap.ui.core.message.MessageParser` interface. The interface is very simple: It has only the `parse()` and the `setProcessor()` method. The `parse()` method takes at least one parameter, that is, the response object from the server. The method can take more model-specific

arguments. The setProcessor() method takes only one argument, the processor object that is used to propagate the messages, this is usually the model instance.

The main task of the message parser is to retrieve the messages from the back end response and then calculate the message delta that is handed over to the message processor by means of the two parameters oldMessages and newMessages of the messageChange event. The oldMessages parameter specifies the messages that are to be removed, and the newMessages parameter specifies the messages that are to be added.

The delta calculation must be a back end-specific implementation. In the OData implementation, for example, all messages for the requested resource(s) must be returned from the back end on every request. This means that all messages that were available before with a target that corresponds to the requested resources must be put in the oldMessages parameter of the event.

### Server Messages, Creation



- To create messages manually that are handled like server messages, use the v2.ODataModel model instance as *message processor*.
- If a *target* is set, it must correspond to a binding path, which is then used to propagate the message to the corresponding Binding. The Binding notifies each control that binds to that path.

```
_processMessage : function() {
 var oProductModel = this.getOwnerComponent().getModel("prodModel");

 var oMessageManager = sap.ui.getCore().getMessageManager();
 oMessageManager.registerMessageProcessor(oProductModel);

 oMessageManager.addMessage(
 new Message({
 message : "{i18n>errMessageNoData}",
 description : "{i18n>errProdNotFound}",
 type: sap.ui.core.MessageType.Error,
 target: "Product(5)/Name",
 processor: oProductModel
 })
);
}
```



Figure 130: Creating Server Messages

It is possible to leave the target of a server message empty. If the target is empty the server message is relevant for the whole application.

### Message Model

The following are characteristics of the message model:

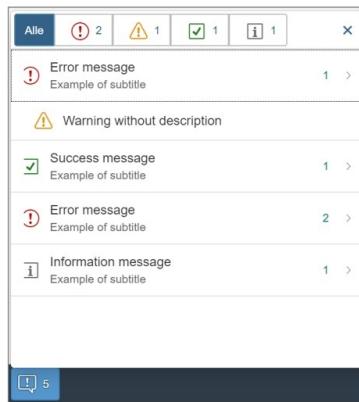
- The message model contains all messages and is used to bind to the messages to display them to the user.
- You use the message model like any other model to bind an aggregation to a root path (""/"), for example the items in a list, and add filters and sorters.
- The message model is retrieved from the message manager by calling the getMessageModel() method:

```
var oMessageModel =
sap.ui.getCore().getMessageManager().getMessageModel();
```

### Example of the Use of the Message Model



- In this example, the MessagePopover control is used to display the messages to the user:



```
sap.ui.define([
 "sap/ui/core/mvc/Controller",
 "sap/ui/core/message/Message",
 "sap/m/MessagePopoverItem",
 "sap/m/MessagePopover"
], function(Controller, Message, MessagePopoverItem, MessagePopover) {
 "use strict";
 return Controller.extend("com.sap.ux402.demo.controller.DemoView", {
 showPopover: function(oEvent) {
 var oMessageTemplate = new MessagePopoverItem({
 type: "{message>type}",
 title: "{message>title}",
 description: "{message>description}",
 subtitle: "{message>subtitle}",
 counter: "{message>counter}"
 });

 var oMessagePopover = new MessagePopover({
 items: {
 path: "/",
 template: oMessageTemplate
 }
 });
 var messageModel = sap.ui.getCore().getMessageManager().getMessageModel();
 oMessagePopover.setModel(messageModel, "message");
 oMessagePopover.openBy(oEvent.getSource());
 },
 });
});
```

Figure 131: Using the Message Model - Example



### LESSON SUMMARY

You should now be able to:

- Work with Messages



## Describing Key Responsive Design Controls



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe Key Responsive Design Controls

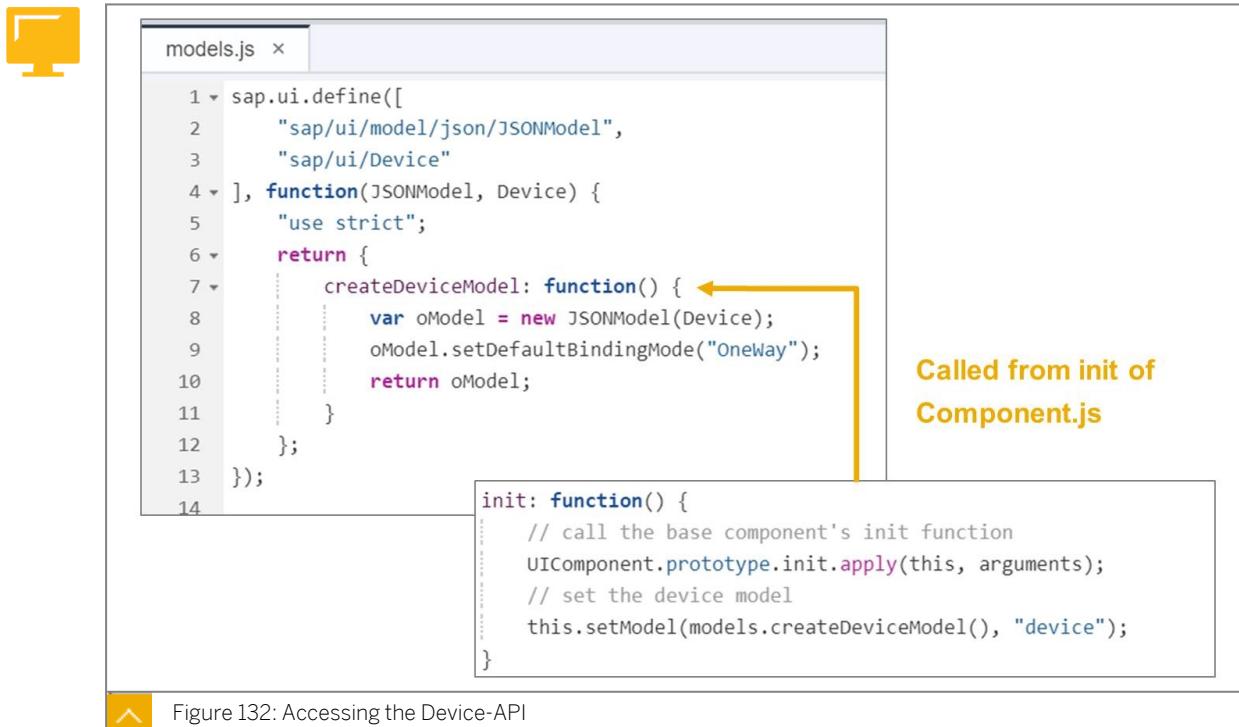
### Key Responsive Design Controls

#### Basics About the Device API:



- The device API is an API which provides information about device specifics.
- Device-API is implemented by the class `sap.ui.Device`.
- You can access for example:
  - Operating system and version
  - Browser and version
  - Screen size
  - Orientation
  - Touch specific features
  - Orientation change

## Device-API, Access



The screenshot shows a code editor window for a file named `models.js`. The code defines a model using the `sap.ui.define` function. It includes imports for `sap/ui/model/json/JSONModel` and `sap/ui/Device`, and a function `createDeviceModel` that creates a new `JSONModel` for the `Device` object, setting its default binding mode to `"OneWay"`. The code is annotated with a yellow arrow pointing from the `createDeviceModel` call in the main body to the `init` function in a callout box. The `init` function calls the base component's `init` method and sets the device model.

```

models.js x
1 sap.ui.define([
2 "sap/ui/model/json/JSONModel",
3 "sap/ui/Device"
4], function(JSONModel, Device) {
5 "use strict";
6 return {
7 createDeviceModel: function() {
8 var oModel = new JSONModel(Device);
9 oModel.setDefaultBindingMode("OneWay");
10 return oModel;
11 }
12 };
13 });
14
15 init: function() {
16 // call the base component's init function
17 UIComponent.prototype.init.apply(this, arguments);
18 // set the device model
19 this.setModel(models.createDeviceModel(), "device");
20 }

```

**Called from init of Component.js**

Figure 132: Accessing the Device-API

When using the SAPUI5-app template of SAP Business Application Studio, the IDE generated during project creation a file called `models.js` in the `model` folder of the project. This file contains a function `createDeviceModel`.

The function is called during the initialization of component. As you can see on the slide the model return by the `createDeviceModel` is stored as a model with id `device` in the context of the Component. This means, it is accessible in every development aspect of the application.

## Device-API, Usage



```
<semantic:MasterPage title="{i18n>masterTitle}">
 <semantic:content>
 <List id="list"
 items="/CarrierCollection"
 busyIndicatorDelay="0"
 growing="true"
 growingThreshold="10"
 growingScrollToLoad="true"
 mode="{= ${device}>/system/phone} ? 'None' : 'singleSelectMaster'"
 selectionChange="onSelect">

 <items>
 <ObjectListItem
 title="{AirLineName}"
 intro="{AirLineID}"
 type="{= ${device}>/system/phone} ? 'Active' : 'Inactive'"
 press="onSelect">
 </ObjectListItem>
 </items>
 </List>
 </semantic:content>
</semantic:MasterPage>
```



Figure 133: Use the Device

The developer can bind properties of UI controls to the device model. In the above example, you can see how to bind the `sap.m.List` control to the device model to support the correct behavior on desktop and phones.

## Content Density

### Content Densities

To adapt to different interaction styles better, SAPUI5 apps need to support three different form factors:



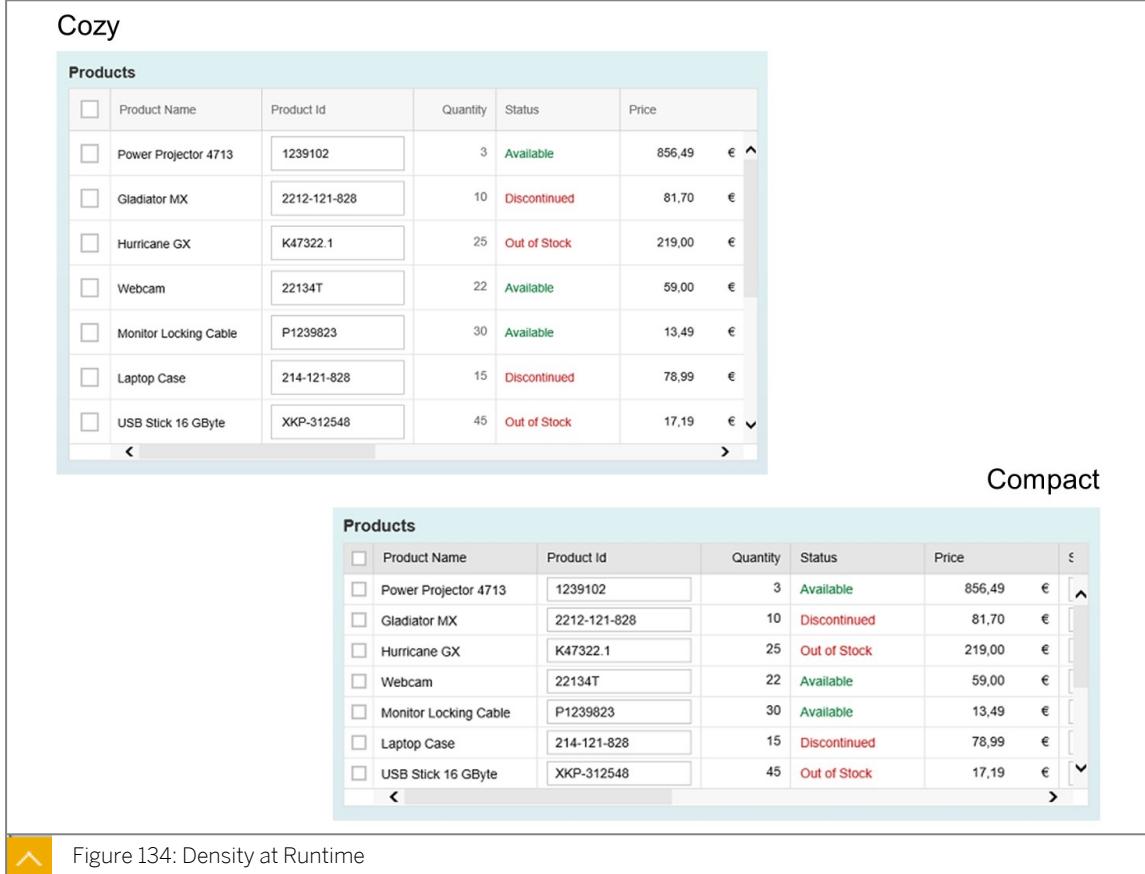
Content Density	CSS Class
Cozy	sapUiSizeCozy
Compact	sapUiSizeCompact
Condensed	sapUiSizeCondensed

The devices used to run apps that are developed with SAPUI5 run on various different operating systems and have very different screen sizes. SAPUI5 contains different content densities for certain controls that allow your app to adapt to the device in question, allowing you to display larger controls for touch-enabled devices and a smaller, more compact design for devices that are operated by mouse.

Please note:

- When running on touch devices, the app should display all controls with dimensions that are big enough for finger tips. This is called the cozy form factor. This is the default density for most controls, particularly those in the `sap.m library`.
- When running on mouse and keyboard enabled (non-touch) devices, the information density on the screen can be increased by reducing control dimensions. This is called the compact form factor. In the compact mode, the default control area of 3 rem is reduced to 2 rem.
- When it is necessary to show the UI control in a smaller size compared to the Compact form factor. For example, this density can be used for all tables of the `sap.ui.table library`.

### Density at Runtime



The screenshot displays two versions of a SAP UI5 table component side-by-side, illustrating the difference in density at runtime. The top section is labeled "Cozy" and the bottom section is labeled "Compact". Both sections have a header "Products" and a table body with columns: Product Name, Product Id, Quantity, Status, and Price. The data rows include items like "Power Projector 4713", "Gladiator MX", "Hurricane GX", etc., with their respective details. The "Cozy" version uses larger fonts and has more vertical space between rows, while the "Compact" version uses smaller fonts and has less vertical space between rows, demonstrating how the same data is presented differently based on the density setting.

Products					
	Product Name	Product Id	Quantity	Status	Price
<input type="checkbox"/>	Power Projector 4713	1239102	3	Available	856,49 € ^
<input type="checkbox"/>	Gladiator MX	2212-121-828	10	Discontinued	81,70 €
<input type="checkbox"/>	Hurricane GX	K47322.1	25	Out of Stock	219,00 €
<input type="checkbox"/>	Webcam	22134T	22	Available	59,00 €
<input type="checkbox"/>	Monitor Locking Cable	P1239823	30	Available	13,49 €
<input type="checkbox"/>	Laptop Case	214-121-828	15	Discontinued	78,99 €
<input type="checkbox"/>	USB Stick 16 GByte	XKP-312548	45	Out of Stock	17,19 € v

Products						
	Product Name	Product Id	Quantity	Status	Price	€
<input type="checkbox"/>	Power Projector 4713	1239102	3	Available	856,49	€ ^
<input type="checkbox"/>	Gladiator MX	2212-121-828	10	Discontinued	81,70	€
<input type="checkbox"/>	Hurricane GX	K47322.1	25	Out of Stock	219,00	€
<input type="checkbox"/>	Webcam	22134T	22	Available	59,00	€
<input type="checkbox"/>	Monitor Locking Cable	P1239823	30	Available	13,49	€
<input type="checkbox"/>	Laptop Case	214-121-828	15	Discontinued	78,99	€
<input type="checkbox"/>	USB Stick 16 GByte	XKP-312548	45	Out of Stock	17,19	€ v

Figure 134: Density at Runtime

The above two screenshots show the difference between the Cozy and Compact densities, using a simple `sap.ui.table`.`Table` example.

## Supported Content Densities



To check which density is supported for a control the best place take a look to the samples section in the demo kit.

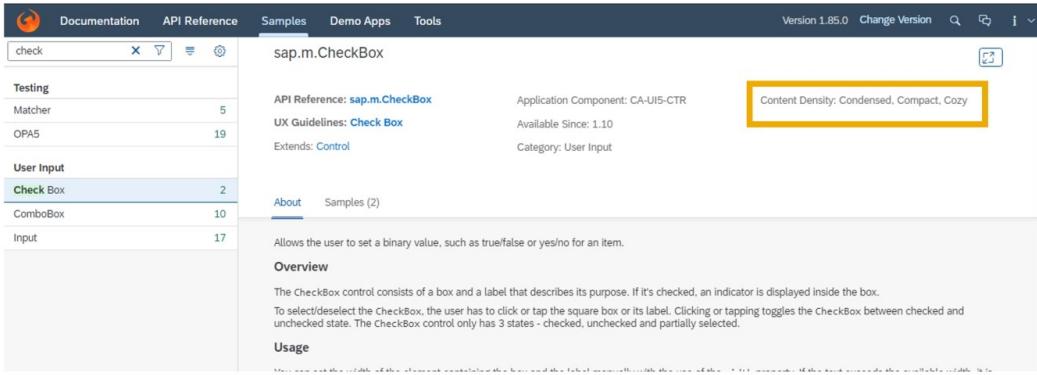


Figure 135: Checking Which Content Densities are Supported

If you need to know which content densities are supported for a particular control, the best place to look is the *Samples* section in the Demo Kit. After choosing a control from the list, look at the details in the *Object Header* area to see which density is supported. In the example shown in the slide, the control supports both the Compact and Cozy content densities.

## Densities, Usage



**XML view definition**

```
<mvc:View controllerName="com.sap.ux402.basicUX402_BasicProject.controller.Main" xmlns:html="http://www.w3.org/1999/xhtml"
 xmlns:mvc="sap.ui.core.mvc" displayBlock="true" xmlns="sap.m"
 class="sapUiSizeCompact"
>
```

**Dialog creation definition**

```
openDialog : function() {
 var oDialog = this._createDialogFragment();
 oDialog.addStyleClass("sapUiSizeCompact");
 this.getView().addDependent(oDialog);
 oDialog.open();
}
```

**Applying density based on device**



Figure 136: Using Densities

A density is triggered by the related CSS class, for example, *sapUiSizeCompact* for the Compact density, set on a parent element of the UI region for which you want to use the controls. This means that some parts of the UI or different apps inside a *sap.m.Shell* can use the standard density of the *sap.m* controls, while other parts can use a different density at the

same time. However, sub-parts of the UI part that is set to Compact density cannot use the Cozy density because the CSS class affects the entire HTML subtree.

As dialogs and other popups are located at the root of the HTML document, you also have to set the CSS class for those elements to the respective density. The CSS class only affects child controls. You cannot make a control itself compact or cozy by adding the CSS class to it. Instead, set the CSS class on the parent container, for example a view or a component.

### Synchronizing a Density for a Dialog



- As dialogs are rendered in a different part of the HTML tree, they do **not** automatically inherit the density.
- `sap.ui.core.syncStyleClass` can be used to synchronize the density.

```
Overview.controller.js ×
dynamicpage > webapp > controller > Overview.controller.js > sap.ui.define() callback > ...
1 sap.ui.define([
2 "sap/ui/core/mvc/Controller",
3 "sap/ui/model/Filter",
4 "sap/ui/model/FilterOperator",
5 "sap/ui/model/Sorter",
6 "sap/ui/core/Fragment",
7 "sap/ui/core/syncStyleClass"
8],
9 /**
10 * @param {typeof sap.ui.core.mvc.Controller} Controller
11 */
12 function (Controller, Filter, FilterOperator, Sorter, Fragment, syncStyleClass) {
13
14 if (!this._oViewSettingsDialog) {
15 Fragment.load({
16 name: "student00.sap.training.dynamicpage.view.ViewSettingsDialog",
17 controller: this
18 }).then(function(oFragment) {
19 this._oViewSettingsDialog = oFragment;
20 this.getView().addDependent(this._oViewSettingsDialog);
21 syncStyleClass(this.getOwnerComponent().getContentDensityClass(), this.getView(), this._oViewSettingsDialog);
22 this._oViewSettingsDialog.open(sDialogTab);
23 }.bind(this));
24 } else {
25 this._oViewSettingsDialog.open(sDialogTab);
26 }
27 }
}
```



Figure 137: Synchronizing a Density for a Dialog

As dialogs are rendered in a different part of the HTML tree, they do not automatically inherit the density. To decide if you set the relevant density for a dialog, either perform the same check as for the view or use the convenience function `sap.ui.core.syncStyleClass`. This convenience function synchronizes a style class between elements. The function accepts the following parameters: CSS class name, source element, target element.

### Hiding/Displaying Controls Depending on the Device

To determine a control's visibility in a device-dependent way, you can use the following CSS classes:



- `sapUiVisibleOnlyOnDesktop`
- `sapUiHideOnDesktop`
- `sapUiVisibleOnlyOnTablet`
- `sapUiHideOnTablet`
- `sapUiVisibleOnlyOnPhone`
- `sapUiHideOnPhone`

To determine a control's visibility in a device-dependent way, the above SAPUI5 CSS classes can be used. It is recommended to use these classes to specify the visibility.

It is important to understand that when using the classes, the control will still be part of the app but hidden by CSS only. For managing visibility on a generic level, consider controlling the visible property with the device API instead, as this means the controls will not be added to the DOM at all but just treated as invisible by SAPUI5.

## Responsive Design

### What is Responsive Web Design (RWD)?



- Responsive web design (RWD) is a concept of developing a website in a manner that helps the layout seamlessly get changed depending on the user's computer screen resolution.
- RWD is a web design approach that makes it possible to maintain one web application for all different types of screens:
  - No need to have a separate mobile app.
- Common adaptations in RWD:
  - Text size smaller for smaller devices.
  - Menus are turned into dropdowns.
  - Pictures are resized, moved or even removed when used on a smaller device.

### RWD and SAPUI5



- The sap.ui.layout namespace provides various layouts helping to build responsive UIs

The screenshot shows the SAPUI5 API Reference interface. The top navigation bar includes links for Documentation, API Reference, Samples, Demo Apps, and Tools, along with version information (Version 1.85.0) and a Change Version button. The main content area is focused on the `sap.ui.layout` namespace. On the left, there is a tree-based navigation pane where the `sap.ui.layout` node is expanded, revealing sub-nodes such as `BackgroundDesign`, `BlockLayout`, `Grid`, and `Flex`. The right side of the interface displays the `Overview` page for the `sap.ui.layout` namespace. This page contains sections for `SAPUI5 library with layout controls`, `Namespaces & Classes`, and `Properties`. Each section lists specific controls with their descriptions and examples. For instance, under `Namespaces & Classes`, it lists `sap.ui.layout.BackgroundDesign`, `sap.ui.layout.BlockLayout`, and `sap.ui.layout.Grid`.

Figure 138: RWD and SAPUI5

SAPUI5 provides a various range of layout controls. The layout controls are defined in a namespace called `sap.ui.layout`.

Depended on the purpose of your application you can choose the appropriate layout control. Not all controls of the `sap.ui.layout` namespace are responsive. You have to check the documentation before you use the control.

## RWD and SAPUI5



- The `sap.ui.layout.form` namespace provides various layouts building responsive form based applications

Namespaces & Classes	Description
<code>sap.ui.layout.form.ColumnCells</code>	An int type that defines how many cells a control inside of a column of a Form control using the ColumnLayout control as layout can use.
<code>sap.ui.layout.form.ColumnContainerData</code>	The ColumnLayout-specific layout data for the FormContainer element.
<code>sap.ui.layout.form.ColumnLayout</code>	The ColumnLayout-specific layout data for the FormElement content fields.
<code>sap.ui.layout.form.ColumnsL</code>	The ColumnLayout control renders a Form control in a column-based responsive way.
<code>sap.ui.layout.form.ColumnsM</code>	An int type that defines how many columns a Form control using the ColumnLayout as layout can have if it has large size Allowed values are numbers from 1 to 3.
<code>sap.ui.layout.form.ColumnsXL</code>	An int type that defines how many columns a Form control using the ColumnLayout as layout can have if it has medium size Allowed values are numbers from 1 to 2.
<code>sap.ui.layout.form.ColumnsXL</code>	An int type that defines how many columns a Form control using the ColumnLayout as layout can have if it has extra-large size Allowed values are numbers from 1 to 4.

Figure 139: RWD and SAPUI5

To implement form based UIs the SAPUI5 framework provides the `sap.ui.layout.form` namespace.

## SimpleForm



Address

Office

Name:	<input type="text" value="Red Point Stores"/>
Street/No.:	<input type="text" value="Main St"/> 1618
ZIP Code/City:	<input type="text" value="31415"/> Maintown
Country:	<input type="text" value="Germany"/>

Online

Web:	<input type="text" value="http://www.sap.com"/>
Twitter:	<input type="text" value="@sap"/>

More

Contact data

Email:	<input type="text"/>
Tel.:	<input type="text" value="+49 6227 7747474"/>
SMS:	<input type="text" value="+49 173 123456"/>

Figure 140: SimpleForm

The SimpleForm provides an easy-to-use API to create simple forms. Inside a SimpleForm, a Form control is created along with its FormContainers and FormElements, but the complexity in the API is removed.

## SimpleForm on Smaller Mobile Device



▪ The SimpleForm will fit the controls on a smaller device

**Address**

Name:

Street/No.:

ZIP Code/City:

Country:

**Online**

Web:

Twitter:

**More**

Contact data

Email:

Figure 141: SimpleForm on Smaller Mobile Device

The SimpleForm UI control will arrange the elements to fit the screen size.

### Implementing Responsive Tables

The following are basics about implementing responsive tables:



- It is a big challenge to implement responsive tables.
- Large tables with many columns don't fit on small screens:
  - CSS will not help.
- However, SAPUI5 does offer two vehicles to help:
  - Column-based solution – column hiding.
  - Row-based solution – pop-ins.

## Responsive Table Example

**Order items (4)**

Model	Name	Qty	Unit Price	Final Price	Status
2967...	Brilliance Monitor Samsung	20	50.00	1,000.00	In Process <span style="color: orange;">⚠️</span> <a href="#">↗</a>
2967...	Apple Keyboard	10	10.00	100.00	Shipped <span style="color: green;">✓</span> <a href="#">↗</a>
2967...	Brilliance Monitor Samsung 2	20	10.00	200.00	Delivered <a href="#">↗</a>
2967...	Brilliance Monitor Samsung 3	2	10.00	20.00	Shipped <a href="#">↗</a>

**Mobile Version**

Name	Status
Brilliance Monitor Samsung	In Process <span style="color: orange;">⚠️</span> <a href="#">↗</a>
Qty: 20 Unit Price: 50.00 Final Price: 1,000.00	
Apple Keyboard	Shipped <span style="color: green;">✓</span> <a href="#">↗</a>
Qty: 10 Unit Price: 10.00 Final Price: 100.00	

Figure 142: Responsive Table Example

The figure shows two examples of the responsive design: the desktop version and the mobile version.

## sap.m.Column Properties for RWD

- **minScreenWidth** defines the break point for the column visibility
  - For example, an iPhone5 has 568px x 320px resolution (dip/device-width)
  - If we assign 400px (or 25em based on 16px) then this column will not be visible for portrait mode (width 320px) but will be visible for landscape mode (width 568px)
  - Instead of px or em you can assign one of the predefined sap.m.ScreenSize types like **Tablet** (for 600px) or **Desktop** (for 1024px)
  - Default value of this property is empty string this means this column will be visible always

**Constructor Detail**

```
new sap.m.Column(sId?, mSettings?)
```

Constructor for a new Column.

Accepts an object literal mSettings that defines initial property values, aggregated and associations for the new instance.

If the name of a setting is ambiguous (e.g. a property has the same name as an event), automatic resolution, one of the prefixes "aggregation:", "association:" or "event:" can be used.

The supported settings are:

- Properties
  - width : sap.ui.core.CSSSize
  - hAlign : sap.ui.core.TextAlign (default: sap.ui.core.TextAlign.Begin)
  - vAlign : sap.ui.core.VerticalAlign (default: sap.ui.core.VerticalAlign.Inherit)
  - styleClass : string
  - visible : boolean (default: true)
  - minScreenWidth : string
  - demandPopin : boolean (default: false)
  - popinHAlign : sap.ui.core.TextAlign (default: sap.ui.core.TextAlign.Begin)
  - popinDisplay : sap.m.PopinDisplay (default: sap.m.PopinDisplay.Block)
  - mergeDuplicates : boolean (default: false)
  - mergeFunctionName : string (default: 'getText')
- Aggregations
  - header (**default aggregation**) : sap.ui.core.Control
  - footer : sap.ui.core.Control
- Associations
- Events

Figure 143: sap.m.Column Properties for RWD - minScreenWidth

By default column is always shown. If you set this property, control checks the minimum width of the screen to show or hide this column. As you can give specific CSS sizes(e.g: "480px" or "40em"), you can also use sap.m.ScreenSize enumeration (e.g: "Phone", "Tablet", "Desktop", "Small", "Medium", "Large", ....).

`sap.m.Column.MediaQuery1->Range1 = 199.`

This property can be used for responsive design. e.g: "40em"(or "640px" or "Tablet") setting shows this column in iPad(and Desktop) but hides in iPhone. Please also see "demandPopin" property.

### **sap.m.Column Properties for RWD - demandPopin**

The following are basics about the demandPopin:



- Depending on your minScreenWidth the column can be hidden in different screen sizes.
- Setting this property to true, shows this column as pop-in instead of hiding it.
- Default value is false.
- Referring back to the RWD table example shown earlier:
  - Name and status are always visible so use default minScreenWidth and demandPopin.
  - Model number can be hidden for small devices, so set minScreenWidth to Small.
  - Quantity, Unit Price and Final Price should pop in, so set minScreenWidth to Small and demandPopin to true.

According to your minScreenWidth settings, the column can be hidden in different screen sizes. Setting this property to true, shows this column as pop-in instead of hiding it.

### **Using Predefined CSS Margin Classes**

The following are predefined CSS Margin Classes:



#### **Single-Sided Margins**

`sapUiTinyMargin<Direction>`

#### **Two-Sided Margins**

`sapUiTinyMargin<Direction><Direction>`

#### **Responsive Margins**

`sapUiResponsiveMargin`

#### **Clear Margin- reset the margin to standard**

`sapUiTinyMargin, sapUiSmallMargin, sapUiMediumMargin, sapUiLargeMargin`

#### **Remove Margin**

`sapUiNoMargin<Direction>`

**Full Margins** If you would like to clear an area all around your control.

#### **Single-sided Margins**

For single-sided margins, choose a size (Tiny, Small, Medium or Large, which stand for 8, 16, 32 or 48px respectively) and a direction (Begin, End, Top or Bottom, where Begin is left and End is right and vice versa in RTL mode). For example, if you need to clear a 32px space to the left of your control (or to the right in RTL mode), you would add the class `sapUiMediumBegin`. You can also add several classes at once, as long as they point to

different directions. For example, you would add classes `sapUiLargeEnd` and `sapUiLargeBottom` to clear a 48px space to the bottom and to the right of a control (or to the left in RTL mode).

### Two-sided Margins

If you'd like to clear the space to the left and right or top and bottom of your control, we've provided several two-sided margin classes for you to use. Again, just choose the size and orientation that you need (BeginEnd, TopBottom). For example, if you need to clear a 32px space both to the left and right of a control, you would add the class `sapUiMediumBeginEnd`.

### Responsive Margins

If your application is supposed to run on smartphone, tablet and desktop, it may be useful to choose your margins depending on the screen width that is available. SAPUI5 now comes with CSS class `sapUiResponsiveMargin`, which does just that.



### LESSON SUMMARY

You should now be able to:

- Describe Key Responsive Design Controls

# Extending Standard Controls



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe how to leverage the features of the standard controls in the SAPUI5 framework

### Extension of Standard Controls

#### Extending a Button, Example



```
sap.ui.define(["sap/m/Button"], function(Button) {
 "use strict";
 return Button.extend("controls.HoverButton", {
 metadata: {
 properties: {
 "allowHover": {
 type: "boolean",
 defaultValue: false
 },
 "hoverText": {
 type: "string"
 }
 },
 events: {
 "hover": {}
 }
 },
 onmouseover: function(evt) {
 if (this.getAllowHover()) {
 this.fireHover();
 }
 },
 renderer: {}
 });
});
```



Figure 144: Code-Example of Extending a Button

This is a sample of extending an existing control:

- Details of this code will be explained throughout this topic discussion.
- The purpose of putting the code up front is to help you visualize the upcoming topics.

The statement on line 1 is an example of extending the Button class, creating a new class named *HoverButton*.

## Control Metadata - Properties



```
sap.ui.define([
 "sap/ui/core/Control"
], function(Control) {
 "use strict";

 return Control.extend("custom.NewCtrl", {
 metadata: {
 properties: {
 ...
 "prodId": "string",
 "prodName": "string",
 "quantity": "int",
 "unitPrice": "float",
 "width": { type: "sap.ui.core.CSSSize", defaultValue: "100px" },
 "height": { type: "sap.ui.core.CSSSize", defaultValue: "100px" }
 ...
 },
 });
});
```

Figure 145: Control Metadata - Properties

The control metadata defines:

- Properties:
  - Made up of a name and a type and optionally a default value.
  - Valid types are string (default), int or float for numeric properties, *int[]* for arrays and *sap.ui.core.CSSSize* for a custom property.

## Control Metadata - Events



- The control metadata defines:

```
return Control.extend("custom.NewCtrl", {
 metadata: {
 properties: {
 ...
 "allowHover": { type: "boolean", defaultValue: false }
 },
 events: {
 ...
 "hover": {}
 }
 },
});
```

Figure 146: Control Metadata - Events

The control metadata defines:

- Events:
  - Defined by their name only.
  - Methods for registering, de-registering and firing the event are created by the framework:
    - For example, *attachHover*, *detachHover* and *fireHover*.
  - Can be interrupted by the application:

- If `enablePreventDefault` is set to `true`.
  - `events : { "hover" : {enablePreventDefault : true} }`.

### Control Metadata - Aggregations and Associations

The difference between Aggregations and Associations are:



#### Aggregation:

- A relationship between two controls.
- A parent/child relationship.
- Example: table rows (parent) and cells (children).



#### Association:

- A relationship between two controls.
- Not a parent/child relationship.
- Represent a loose coupling.
- Example: a label is associated with a text field.

SAPUI5 also knows composite UI-controls. To implement a composite UI-control the control developer has to define aggregations or associations to other controls.



- Aggregations are defined by their name and a configuration object
- The configuration object contains:
  - type – subclass of the control
  - multiple – defines if is 0..1 or 0..n (default). True is 0..n
  - singularName – defines names of methods to be created by the framework
    - For example, `addWorksetItem`
- If type is all you want to set, you can do so as a string instead of the configuration object

```
aggregations: {
 "acceptButton" : "sap.m.Button",
 "content" : {singularName: "content"},
 "worksetItems" : {type : "sap.ui.core.Item", multiple : true, singularName : "worksetItem"}
```

Figure 147: Control Metadata - Aggregations and Associations

The above slide shows how to define an aggregation.

### Rules for Control Method Implementations

Rules for Control Method Implementations are:



- After adding the metadata, you add method implementations to the control.
- Method naming restrictions:

- Don't use names of methods provided in superclass.
  - Would result in overriding the superclass methods.
- Don't use names starting with get..., set..., insert..., add..., remove... or indexOf... as they could collide with automatically generated methods of the properties or aggregations.
- Method names with special meanings:
  - on....: are used for event handlers.
  - init: are used to initialize the control, after its instantiation:
    - This is a private method only called by the SAPUI5 core.
  - renderer: used for the function that creates the control's HTML.



- Private methods are identified by a name starting with an underscore
  - \_checkForZero : function(y) {...}
- All other methods are considered public
- The init method is used to provide default values to internal variables or subcontrols in the case of a composite control
  - init : function( ) { ... }
- Event handler methods
  - Called automatically
  - Have access to getters and setters of properties defined by the control

```
onmouseover: function(evt) {
 if (this.getAllowHover()) {
 this.fireHover();
 }
},
```



Figure 148: Control Method Implementations

Property getters and setters are automatically created by the framework.

Be sure to use the original example code for extending a control (shown earlier) and that you recall that the control had an *allowHover* property added. Because of that, you should understand the framework creates the *getAllowHover()* seen in the code above.



- Recall that earlier, you saw the definition of a new event named hover
  - And that the framework appends on... in front of events
- Here is an example of application code using the new event:

```

<mvc:View xmlns:core="sap.ui.core" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:f="sap.ui.layout.form"
 xmlns:l="sap.ui.layout" xmlns:cust="custom"
 controllerName="controller.MyController" xmlns:html="http://www.w3.org/1999/xhtml">
 <Page navButtonPress="onNavBack" showNavButton="true"
 title="Page title">
 <content>
 <cust:NewCtrl text="OK" allowHover="true"
 hoverText="My hover text"
 hover="onHover"/>
 </content>
 </Page>
</mvc:View>

onHover : function(evt) {
 MessageToast.show(evt.getSource().getHoverText(),
 {duration:1000});
},

```

Figure 149: Handling an Event at Runtime

Namespace

Control usage

Controller

Go back to the code at the start of this topic. See the event named "hover" that was added. Should now understand that the framework puts "on" in front of your event names and that coincides with the code shown above.

### Control Renderer

The following are information about Control Renderer:



- The renderer is responsible for creating the HTML structure for the control.
- A renderer is assigned to the render-method of the control:
- Static, so cannot use "this":
  - *Control* instance and *RenderManager* instance are given to the method,
- *RenderManager* collects and concatenates string fragments and places them in the DOM at the appropriate position by using its methods such as:
  - *openStart* - Opens the start tag of an HTML element,
  - *openEnd* - Ends an open tag started with *openStart*,
  - *style* - Adds a style name-value pair to the style collection of the last open HTML element. This is only valid when called between *openStart/voidStart* and *openEnd/voidEnd*.
- The renderer method has access to the control's property getters and setters.
- The renderer method also has access to any aggregations and associations defined by the control.

## Referencing the Control Renderer

```

 sap.ui.define([
 "sap/ui/core/Control",
 "controls/NewCtrlRenderer"
], function(Control,NewCtrlRenderer) {
 "use strict";
 });
 return Control.extend("controls.NewCtrl", {
 metadata: {
 properties: {
 "prodId": "string",
 "prodName": "string",
 "quantity": "int",
 "unitPrice": "float",
 "width": {type: "sap.ui.core.CSSSize", defaultValue:"100px"},
 "height": {type: "sap.ui.core.CSSSize", defaultValue:"100px"}
 }
 },
 renderer: NewCtrlRenderer
 });
}

```

Figure 150: Referencing Control Renderer

The UI-control definition takes a reference to the renderer class.

### Sample: Control Renderer

```

sap.ui.define([
 "sap/ui/core/Renderer"
], function(Renderer) {
 "use strict";

 var NewCtrlRenderer = Renderer.extend("controls.NewCtrlRenderer");
 NewCtrlRenderer.apiVersion = 2;
 NewCtrlRenderer.render = function(oRm, oControl) {
 ...
 };
 return NewCtrlRenderer;
})

```

Figure 151: Sample: Control Renderer

The above slide shows you in a sample implementation the anatomy of a renderer implementation.

As of SAPUI5 1.67 the RenderManager provides a set of new APIs to describe the structure of the DOM that can be used by the control renderers. Assign the property `apiVersion` with value 2 to your `Renderer`-implementation to use the new API.

Contract for `Renderer.apiVersion` 2.

To allow a more efficient in-place DOM patching and to ensure the compatibility of the control, the following prerequisites must be fulfilled for the controls using the new rendering technology.

Legacy control renderers must be migrated to the new semantic renderer API:

- `openStart`,

- *voidStart*,
- *style*,
- *class*,
- *attr*,
- *openEnd*,
- *voidEnd*,
- *text*,
- *unsafeHtml*,
- *icon*,
- *accessibilityState*,
- *renderControl*,
- *cleanupControlWithoutRendering*

During the migration, restrictions that are defined in the API documentation of those methods must be taken into account, e.g. tag and attribute names must be set in their canonical form. Fault tolerance of HTML5 markup is not applicable for the new semantic rendering API, e.g. except void tags, all tags must be closed; duplicate attributes within one HTML element must not exist.

Existing control DOM structure will not be removed from the DOM tree; therefore all custom events, including the ones that are registered with jQuery, must be de-registered correctly at the *onBeforeRendering* and exit hooks. Classes and attribute names must not be escaped.

Styles should be validated via types (e.g. `sap.ui.core.CSSSize`). But this might not be sufficient in all cases, e.g. validated URL values can contain harmful content; in this case `encodeCSS` can be used.

To allow a more efficient DOM update, second parameter of the `openStart` or `voidStart` methods must be used to identify elements, e.g. use `rm.openStart("div", oControl.getId() + "-suffix")`; instead of `rm.openStart("div").attr("id", oControl.getId() + "-suffix")`; Controls that listen to the *focusin* event must double-check their focus handling. Since DOM nodes are not removed and only reused, the *focusin* event might not be fired during re-rendering.

## Render Function Implementation



```

NewCtrlRenderer.render = function(oRm, oControl) {
 oRm.openStart("div", oControl);
 oRm.openEnd();
 oRm.style("width",oControl.getWidth());
 oRm.style("height",oControl.getHeight());
 oRm.class("NewCtrlClass");

 oRm.openStart("table");
 oRm.attr("align", "center");
 oRm.openEnd();
 oRm.openStart("tr");
 oRm.openEnd();
 oRm.openStart("td");
 oRm.openEnd();

 oRm.text(" " + oControl.getProdId());
 oRm.close("td");
 oRm.close("tr");
 oRm.openStart("tr");
 oRm.openEnd();
 oRm.openStart("td");
 oRm.openEnd();

 oRm.text(" " + oControl.getProdName());
 oRm.close("td");
 oRm.close("tr");
 oRm.openStart("tr");
 oRm.openEnd();
 oRm.openStart("td");
 oRm.openEnd();

 oRm.text(" " + oControl.getQuantity());
 oRm.close("td");
 oRm.close("tr");
 oRm.openStart("tr");
 oRm.openEnd();
 oRm.openStart("td");
 oRm.openEnd();

 oRm.text(" " + oControl.getUnitPrice());
 oRm.close("td");
 oRm.close("tr");
 oRm.close("table");

 oRm.close("div");
};

```



Figure 152: Render function implementation

The above slide shows a sample implementation of the render method. The method has access to the RenderManager instance and to the control that should be rendered.



## LESSON SUMMARY

You should now be able to:

- Describe how to leverage the features of the standard controls in the SAPUI5 framework

# Describing Custom Controls



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe how to implement custom controls using the SAPUI5 framework

## Custom Controls

**Control, Definition: What is a Control?**



**Conceptually:**

A UI component (usually a visible one) that can react to user activities and expose properties, methods and events to the application (and other controls) via JS API.

**Technically, a UI5 control at design time consists of:**

The **Control API** definition which defines the properties, events, methods and sometimes the associations and aggregations.

The **Control Renderer** which is responsible for creating the HTML string defining the structure of the control.

The **Control Behavior** which is the JavaScript code taking care of the control interactivity, reacting on user events, firing control events, handling method calls and property changes.

The **Control Style** defining the visuals of the control (usually a control has different visual designs, bundled as "Themes").

## Same Rules Apply to Custom Controls

**Rules, which apply to Custom Controls are:**



- The Control consists of the following elements:
  - Interface:
    - Properties
    - Events
    - Aggregations
    - Associations
    - Methods
  - Behavior

- Renderer
- CSS Theme Data
- You can define properties
- Aggregations and associations
- You can define events
- You can define normal methods
- Can have a renderer method
- As stated, there is not much difference in extending a control and creating a custom control

### Custom Similar To Extending



- Once you can extend a control, creating a custom control is not really that different
- When you create a custom control, you simply change who the control is extending
- In this example, it is said that you are creating a custom control because you are extending the generic Control class

```
sap.ui.define([
 "sap/ui/core/Control"
], function(Control) {
 "use strict";

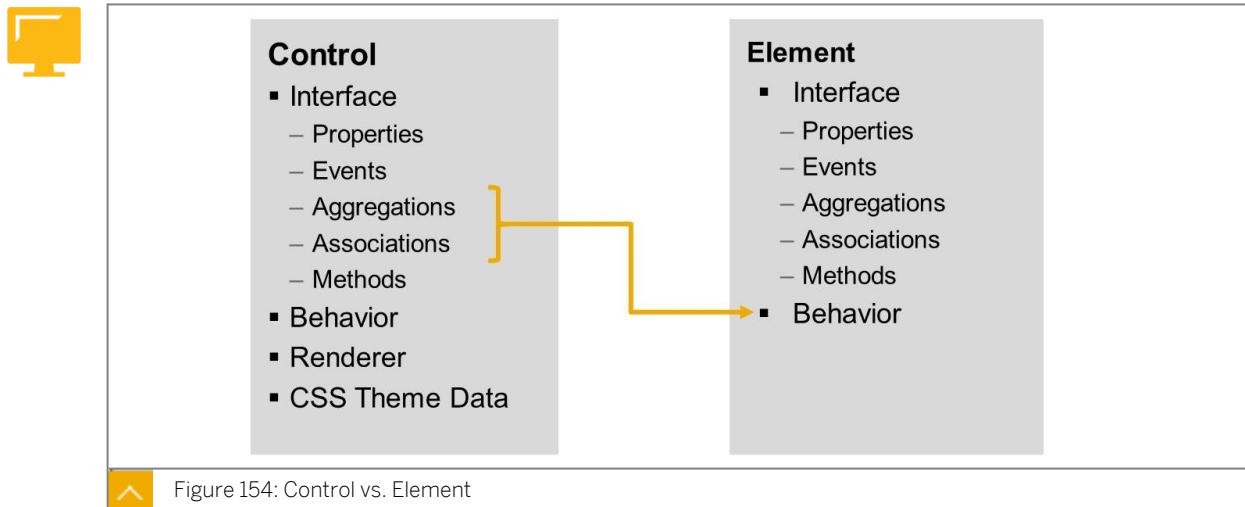
 return Control.extend("custom.NewCtrl", {
 metadata: {
 properties: {
 ...
 "prodId": "string",
 "prodName": "string",
 "quantity": "int",
 "unitPrice": "float",
 "width": { type: "sap.ui.core.CSSSize", defaultValue: "100px" },
 "height": { type: "sap.ui.core.CSSSize", defaultValue: "100px" }
 }
 }
 });
});
```



Figure 153: Custom Similar To Extending

Compare this declaration to that of the *HoverButton* seen earlier.

## Control vs. Element

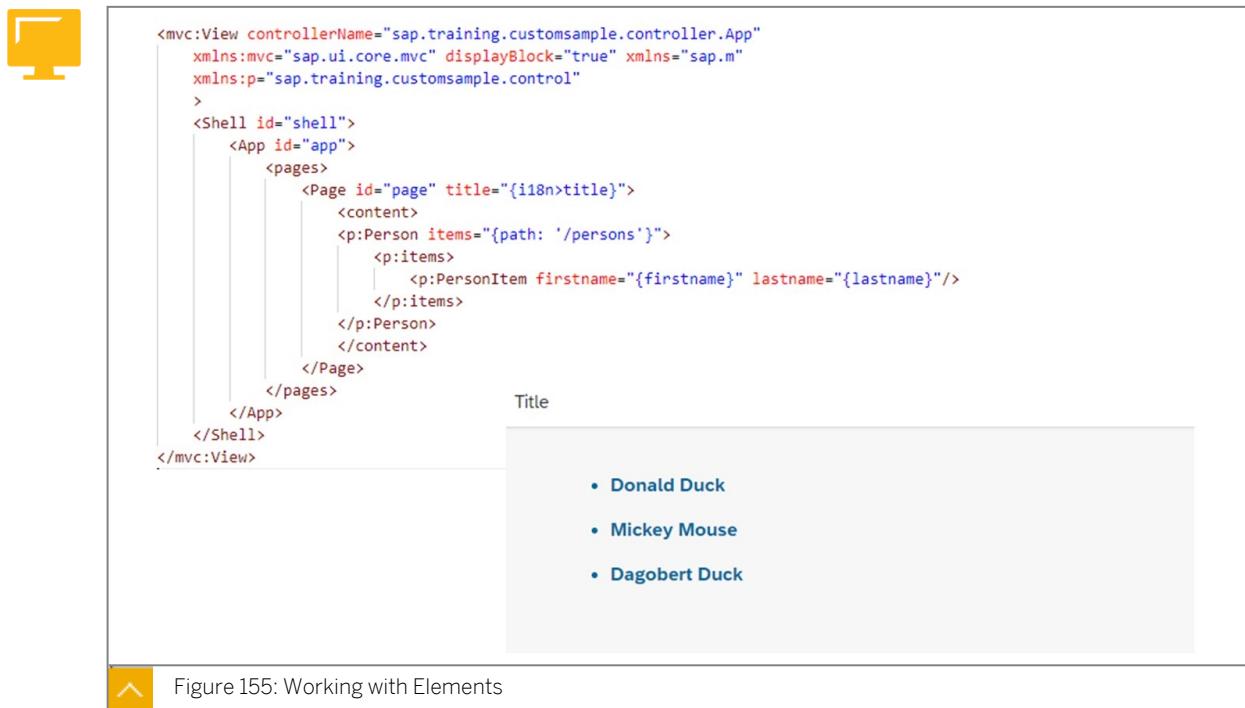


SAPUI5 distinguishes between a Control and an Element. An Element is a visual aspect that is not able to live without a parent/parent control.

### What is an "Element"?

`sap.ui.core.Element` is base class of `sap.ui.core.Control`

- 
- Does have an API but usually not a renderer.
  - Not meant to be used standalone.
  - Typically aggregated by a control (e.g. MenuItems in a Menu).
  - The control does the rendering and uses the Element as data container.



The above slide shows you how to use elements during aggregation binding.



```
Person.js x
1 1 √ sap.ui.define([
2 | "sap/ui/core/Control"
3 3 √], function(Control) {
4 | "use strict"
5
6 √ return Control.extend("sap.training.customsample.control.PersonControl",{
7 √ metadata: {
8 √ aggregations : {
9 √ "items" : {
10 | type : "sap.training.customsample.control.PersonItem",
11 | multiple : true,
12 | singularName : "item"
13 }
14 },
15 defaultAggregation : "items"
16 },
17
18 renderer : "sap.training.customsample.control.PersonRenderer"
19 })
20 })
```



```
1 1 √ sap.ui.define([
2 | "sap/ui/core/Element"
3 3 √], function(Element) {
4 | "use strict"
5
6 √ return Element.extend("sap.training.customsample.control.PersonItem",{
7 √ metadata: {
8 √ properties : {
9 √ "firstname" : { type : "string"},
10 "lastname" : { type : "string"}
11 }
12 }
13 })
14 })
```

Figure 156: Implementation Relationship Between Control and Element

The above code shows you the implementation and relationship of a control and an element.



```
PersonRenderer.js x

1 sap.ui.define([
2 "sap/ui/core/Renderer"
3], function(Renderer) {
4 "use strict";
5
6 var PersonRenderer = Renderer.extend("sap.training.customsample.control.PersonRenderer");
7 PersonRenderer.apiVersion = 2;
8 PersonRenderer.render = function(oRm, oControl) {
9 let aItems = oControl.getItems();
10
11 oRm.openStart("ul");
12 oRm.style("margin", "20px 50px");
13 oRm.openEnd();
14 for(var i=0; i < aItems.length; i++) {
15 oRm.openStart("li");
16 oRm.style("padding-top", "20px");
17 oRm.style("font-size", "16px");
18 oRm.style("color", "#006699");
19 oRm.style("font-weight", "bold");
20 oRm.openEnd();
21 oRm.text(aItems[i].getFirstname());
22 oRm.text(" ");
23 oRm.text(aItems[i].getLastname());
24 oRm.close();
25 }
26 oRm.close();
27 };
28 return PersonRenderer;
29 });
});
```

Figure 157: Accessing the Elements During Rendering

The code above shows you how to access the list of PersonItem-elements.



## LESSON SUMMARY

You should now be able to:

- Describe how to implement custom controls using the SAPUI5 framework



## Unit 2

### Lesson 11

# Creating Control and Component Libraries



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create a control or component library

### Implementing Custom Libraries

In this unit, we want to move the previously created UI controls into a UI library. The figure shows the two UI controls.



**Controls encapsulated in a UI library**

The screenshot shows a UI library interface. At the top, it says "American Airlines" and "AA". Below that is a "Flights" section with a table:

Name	Flight-No	Flight date	Plane Infos	Book flight
AA	0017	Mar 22, 2018	747-400 385 375	Book flight
AA	0017		747-400 385 372	Book flight

A yellow callout box with the text "10 seats are available" points to the second row of the table. Two arrows point from the "Plane Infos" and "Book flight" buttons in the first row to their respective counterparts in the second row, indicating they are reused.

Figure 158: Reusing UI Controls

A custom library provides reuse functionality and UI controls without copy and paste.

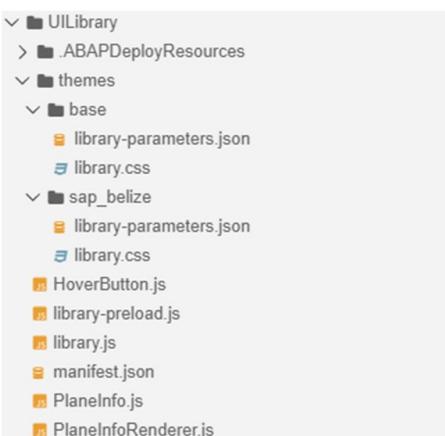
The following figure shows the basic advantages of control libraries.



- Use case: Reuse controls / functionality in different applications
- Control libraries provide automatic support for theming
- Control libraries provide automatic support for right-to-left languages
- Can be installed on the ABAP-platform
- Can be installed in the SAP Cloud Platform

Figure 159: Control Library Basics

The following figure shows the structure of a custom library.

**library.js**

Initializes the library

Defines the name and version

Defines dependencies

Defines types, interfaces, controls, and elements

**manifest.json**

Defines the application as a library

Figure 160: Structure of a Custom Library

**Encapsulate Function in a Custom Library - library.js**

The *library.js* file contains JavaScript code for all enumeration types.

It contains dependencies to other namespaces and library-specific initialization code that is independent from the controls in the library.

The file calls the `sap.ui.getCore().initLibrary()` method with an object that describes the content of the library which includes:

- Types
- Interfaces
- Controls
- Elements

The following figure is an example of a *library.js* file.



```

library.js x
 1 sap.ui.define(["jquery.sap.global","sap/ui/core/library"],
 2 function(jQuery,library) {
 3 "use strict";
 4
 5 sap.ui.getCore().initLibrary({
 6 name:"student00.com.sap.training.ux402.controls",
 7 version:"1.0.0",
 8 dependencies: ["sap.ui.core"],
 9 types:[],
 10 interfaces: [],
 11 controls: [
 12 "student00.com.sap.training.ux402.controls.PlaneInfo",
 13 "student00.com.sap.training.ux402.controls.HoverButton"
 14],
 15 elements: [
 16],
 17 noLibraryCSS: true
 18 });
 19
 20 return student00.com.sap.training.ux402.controls;
 21 },/*bExport*/ false);
 22
```

Figure 161: library.js

The following figure describes the functionality of the `sap.ui.getCore().initLibrary()` method.



- Provides the framework with information about a library
- Intended to be called exactly once while the main module of a library (its `library.js` module) is executing, typically at its begin
- The single parameter `oLibInfo` is an info object that describes the content of the library
- A normalized version of the `oLibInfo` will be kept and will be returned as library information in later calls to `getLoadedLibraries()`

Figure 162: `library.js` – `sap.ui.getcore().initlibrary()` Method

The following figure compares the purpose of the `library.js` and `manifest.json` files.



- The information contained in the `sap.ui.getCore().initLibrary(oLibInfo)` method is partially redundant to the content of the library descriptor (`manifest.json`).
- Future versions of UI5 may ignore the information provided in the `sap.ui.getCore().initLibrary(oLibInfo)` method and may evaluate the descriptor file instead.
- Library developers should keep the information in both files in sync.

Figure 163: `library.js` Vs `manifest.json`

### Encapsulate Function in a Custom Library – `manifest.json`

The following figure describes the purpose of the `manifest.json` file.



- The descriptor for applications, components, and libraries.
- Inspired by the Web Application Manifest concept introduced by the W3C.
- Provides a central, machine-readable and easy-to-access location for storing metadata associated with:
  - Application
  - application component
  - library

Figure 164: `Manifest.json`

The structure of the `manifest.json` file is shown in the following figure.



#### `sap.app`

Defines general application attributes.

#### `sap.ui`

Defines general UI attributes.

#### `sap.ui5`

Defines SAPUI5 specific attributes.

#### `manifest.json`

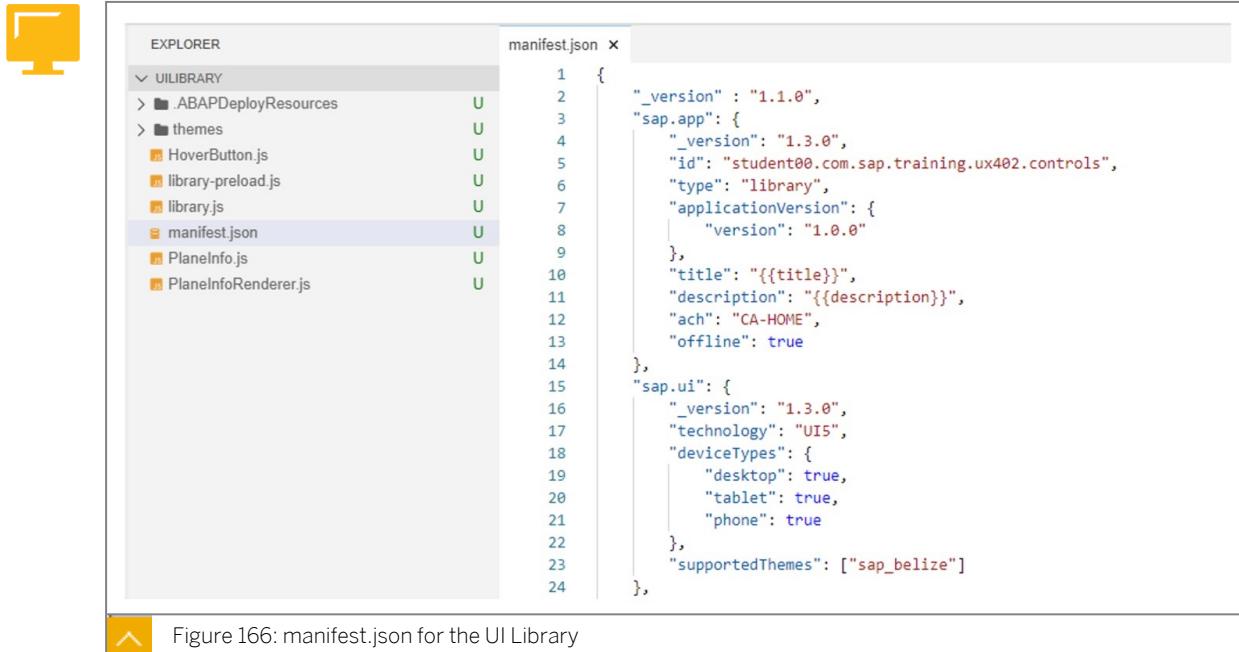
```

1 {
2 "_version" : "1.1.0",
3 "sap.app": { },
15 "sap.ui": { },
25 "sap.ui5": { }
42 }
```

Figure 165: Structure of the `manifest.json` File

In the *manifest.json* file the developer must provide the following attributes:

- *id*  
Unique app identifier
- *type*  
Must be set to library
- *applicationVersion*  
Version number including patch



The screenshot shows the SAP IDE's Explorer view on the left, displaying a folder structure for a UI Library. The manifest.json file is selected in the Explorer. To the right is the code editor window showing the manifest.json file content. The manifest.json file contains the following JSON code:

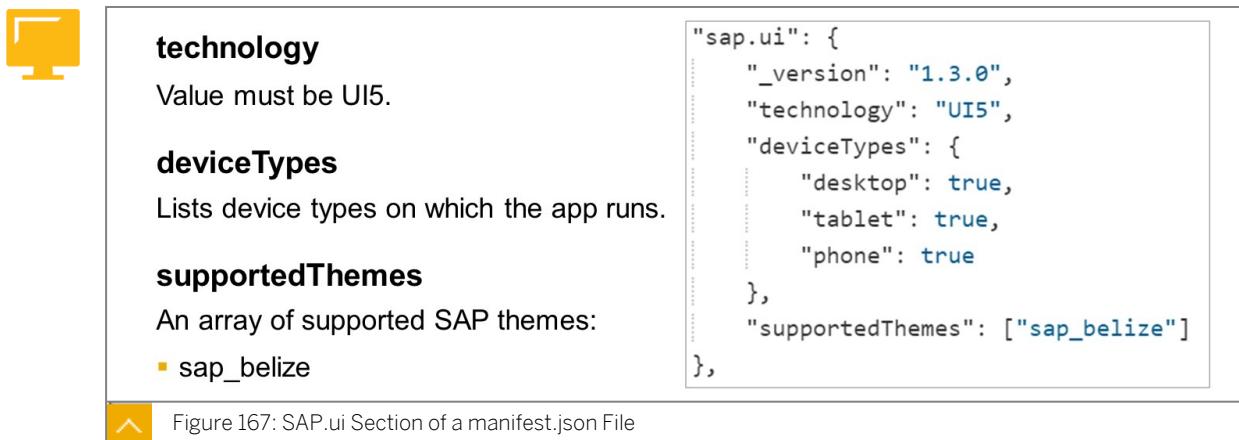
```

1 {
2 "_version": "1.1.0",
3 "sap.app": {
4 "_version": "1.3.0",
5 "id": "student00.com.sap.training.ux402.controls",
6 "type": "library",
7 "applicationVersion": {
8 "version": "1.0.0"
9 },
10 "title": "{{title}}",
11 "description": "{{description}}",
12 "ach": "CA-HOME",
13 "offline": true
14 },
15 "sap.ui": {
16 "_version": "1.3.0",
17 "technology": "UI5",
18 "deviceTypes": {
19 "desktop": true,
20 "tablet": true,
21 "phone": true
22 },
23 "supportedThemes": ["sap_belize"]
24 }
},

```

Figure 166: manifest.json for the UI Library

The following figure gives more detail about the *sap.ui* section of the *manifest.json* file.



The screenshot shows the SAP IDE's Explorer view on the left, displaying a folder structure. The manifest.json file is selected in the Explorer. To the right is the code editor window showing the SAP.ui section of the manifest.json file. The SAP.ui section contains the following JSON code:

```

"sap.ui": {
 "_version": "1.3.0",
 "technology": "UI5",
 "deviceTypes": {
 "desktop": true,
 "tablet": true,
 "phone": true
 },
 "supportedThemes": ["sap_belize"]
},

```

Figure 167: SAP.ui Section of a manifest.json File

The following figure gives more detail about the *sap.ui5* section of the *manifest.json* file.



### dependencies

specifies the external dependencies that are loaded by the SAPUI5 core during the initialization phase of the component minUI5Version

- Minimum version of SAPUI5 that your component requires

### contentDensities

Contains the content density modes that the app supports

- compact
- cozy

### manifest.json

```

25 "sap.ui5": {
26 "_version": "1.2.0",
27 "dependencies": {
28 "minUI5Version": "1.30.0",
29 "libs": {
30 "sap.ui.core": {
31 "minVersion": "1.30.0"
32 }
33 }
34 },
35 "contentDensities": {
36 "compact": true,
37 "cozy": true
38 }
39 }
40 }
41 }
```



Figure 168: SAP.ui5 Section of the manifest.json File

To get the mapping of SAPUI5 version to the \_version name in the *manifest.json* file take a look into the documentation.

<https://sapui5.hana.ondemand.com/#/topic/be0cf40f61184b358b5faedaec98b2da.html>

### Encapsulate Function in a Custom Library - The Control

The following figure shows the characteristics of controls in a custom library.



- Multiple Controls can be added to a library.
- The full qualified classname must match the one defined in library.js.
- Standard JavaScript can be used.
- Defined as a JavaScript module with name, dependencies, and factory function

```
sap.ui.define([...], function() {...});
```



Figure 169: The Control

At the end, you must implement the controls that are to be encapsulated in the UI library.



```

PlaneInfo.js x

1 sap.ui.define([
2 "sap/ui/core/Control",
3 "student00/com/sap/training/ux402/controls/PlaneInfoRenderer"
4], function(Control, PlaneInfoRenderer) {
5 "use strict";
6
7 return Control.extend("student00.com.sap.training.ux402.controls.PlaneInfo", {
8 metadata: {
9 properties: {
10 "seatsMax": {
11 type: "string"
12 },
13 "seatsOcc": {
14 type: "string"
15 },
16 "planeType": {
17 type: "string"
18 }
19 }
20 },
21 renderer: PlaneInfoRenderer
22 });
23 });

```

Figure 170: Implement The Controls For Reuse

After finished with the development you have to deploy the UI library project to your on premise or cloud system.

After deployment, the UI library shows up in transaction SCIF.



**Define Services**

Create Host/Service External Aliases System Monitor Active

Filter Details

Virtual Host	Service Path	
ServiceName	ZUX*LIBRARY*	
Description		
Lang.	English	Ref.Service:

**Virtual Hosts / Services**

Virtual Host	Documentation	Reference Service
default_host	VIRTUAL DEFAULT HOST	
sap	SAP NAMESPACE; SAP IS OBLIGED NOT T...	
bc	BASIS TREE (BASIS FUNCTIONS)	
bsp	BUSINESS SERVER PAGES (BSP) RUNTIME	
sap	NAMESPACE SAP	
ui5_u5	SAPUI5 Application Handler SAPUI5 Appl...	
sap	sap Namespace for SAPUI5 Applications	
zux402library00	UI-Library	

Figure 171: The UI library on the Front-end Server

## Custom Library Usage

In the *index.html* of the consuming UI5-application, we must specify a mapping between the namespaces of the UI5 controls of the library and the URI of the library project.

If it possible to use `jQuery.sap.registerModulePath` instead of using the `data-sap-ui-resourceroots` in the *index.html* file.



```

index.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>ux402_fullscreen</title>
7 <script id="sap-ui-bootstrap"
8 src=".resources/sap-ui-core.js"
9 data-sap-ui-theme="sap_belize"
10 data-sap-ui-resourceroots='{"student00.com.sap.training.ux402.fullscreen.ux402_fullscreen": "./",
11 "student00.com.sap.training.ux402.controls": "../../zux402library00/"}
12 data-sap-ui-compatVersion="edge"
13 data-sap-ui-oninit="module:sap/ui/core/ComponentSupport"
14 data-sap-ui-async="true"
15 data-sap-ui-frameOptions="trusted">
16 </script>
17 </head>
18 <body class="sapUiBody">
19 <div data-sap-ui-component data-name="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen"
20 data-id="container" data-settings='{"id": "ux402_fullscreen"}'>
21 </div>
22 </body>
23 </html>

```

Figure 172: Mapping of Namespace to URL

A dependency is defined in the *manifest.json* file of the application that uses the UI library. The dependency triggers the get response of the *library.js* file from the dependent UI library.



```

manifest.json x
43
44
45 "sap.ui5": {
46 "flexEnabled": false,
47 "rootView": {
48 "viewName": "student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.view.Main",
49 "type": "XML",
50 "async": true,
51 "id": "Main"
52 },
53 "dependencies": {
54 "minUI5Version": "1.60.1",
55 "libs": {
56 "sap.ui.core": {},
57 "sap.m": {},
58 "sap.ui.layout": {}
59 },
60 }
61 },
62 "contentDensities": {
63 "compact": true,
64 "cozy": true
65 },
66 "models": {
67 "i18n": {
68 "type": "sap.ui.model.resource.ResourceModel",
69 "settings": {
70 "bundleName": "student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.i18n.i18n"
71 }
72 },
73 "": {
74 "dataSource": "mainService",

```

Figure 173: Referencing the UI Library in the manifest.json File

The usage of the UI controls from the UI library is identical as already discussed. You must implement a reference (1) for example, in the controller of a view.

Thereafter, you can define (2) an XML namespace in the view and use the UI control in the view as any other standard UI5 control from SAP.



```

Flights.controller.js x
1 sap.ui.define([
2 "sap/ui/core/mvc/Controller",
3 "student00/com/sap/training/ux402/controls/HoverButton",
4 "sap/m/MessageToast",
5 "student00/com/sap/training/ux402/controls/PlaneInfo"
6],
7 function (Controller, HoverButton, MessageToast, PlaneInfo) {
8 "use strict";
9

```

```

Flights.view.xml x
1 <mvc:View controllerName="student00.com.sap.training.ux402.fullscreen.ux402_fullscreen.controller.Flights"
2 xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m" xmlns:l="sap.ui.layout"
3 xmlns:cust="student00.com.sap.training.ux402.controls">
4 <Page navButtonPress="onNavBack" showNavButton="true" title="{AirLineName}">
5 <content>
6 <l:VerticalLayout>
7 <ObjectHeader title="{AirLineName}" number="" numberUnit="{LocalCurrencyCode}" titleHref="{URL}">
8 <attributes>
9 <ObjectAttribute text="{AirLineID}"></ObjectAttribute>
10 <ObjectAttribute text="{URL}"></ObjectAttribute>

```




 Figure 174: Usage of UI Control from the UI Library

When development is finished, you will also deploy the UI5 application to your infrastructure.

The UI library and the UI5 application must be hosted on the same system as shown in the following figure.

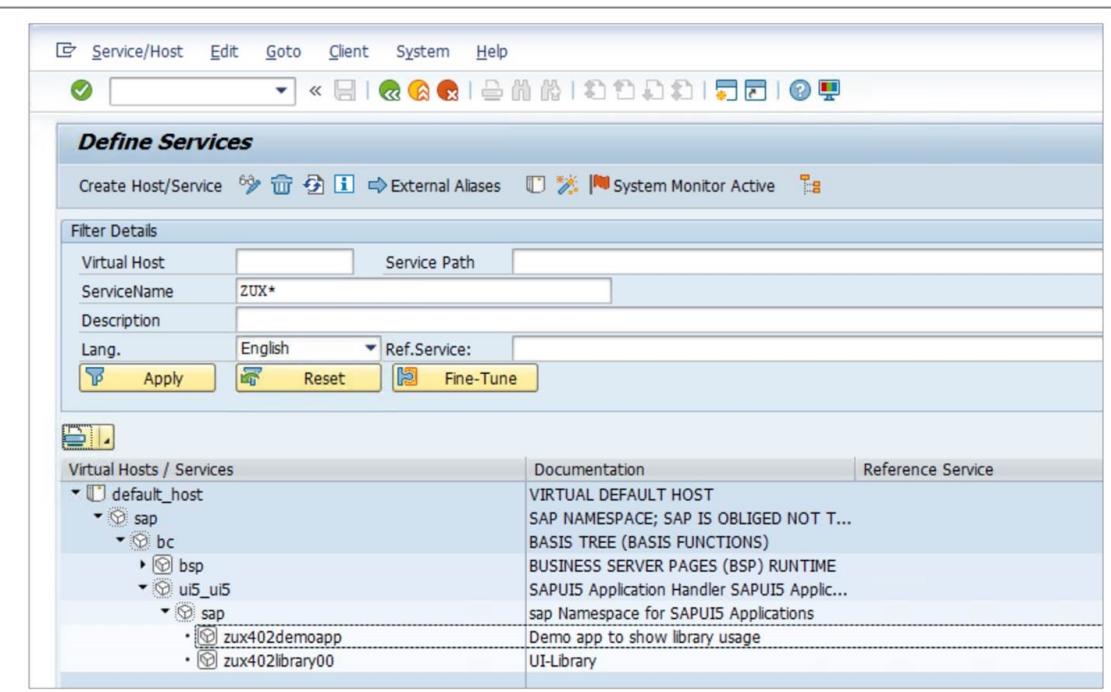



 Figure 175: UI Library and UI5 Application on the FES



## LESSON SUMMARY

You should now be able to:

- Create a control or component library

# Implementing Unit Tests with Qunit



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement unit tests

### Unit Test with QUnit

Before you start implementing your first test, you should think about how to test the different aspects of your application. QUnit testing is based on the Agile Testing Pyramid as shown in the following figure.

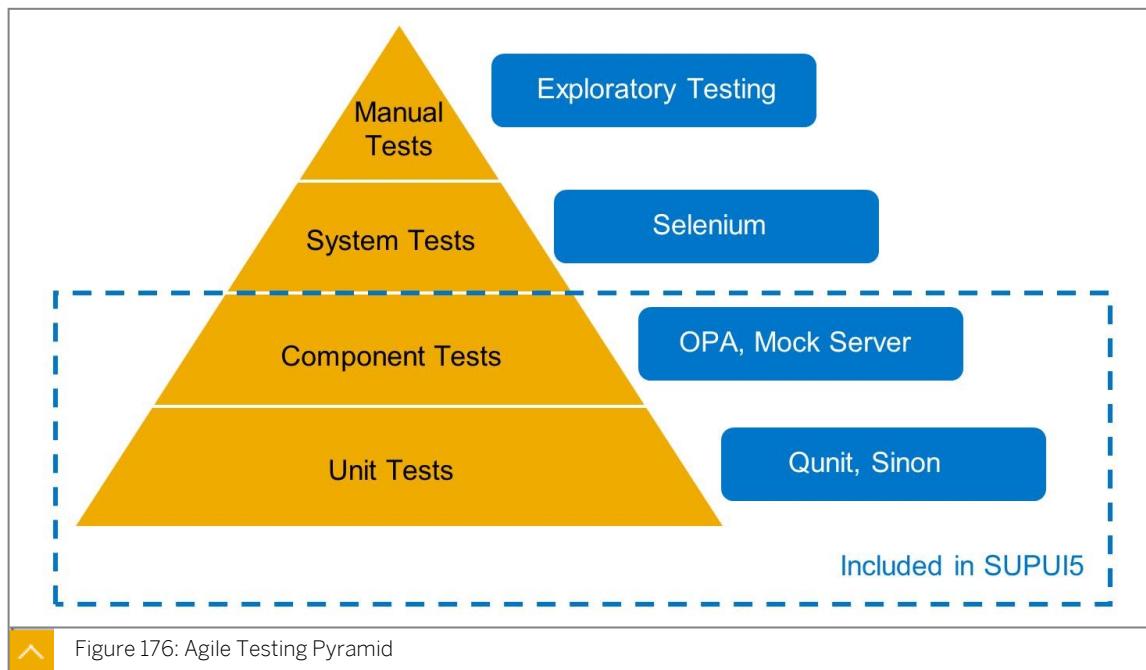


Figure 176: Agile Testing Pyramid

For a detailed overview of mockups, please visit: <http://martinfowler.com/articles/mocksArentStubs.html>.

A mock is an object that we can set expectations on, and which will verify that the expected actions have indeed occurred. A stub is an object that you use to pass to the code under test. You can set expectations on it, so it would act in certain ways, but those expectations will never be verified. A stub's properties automatically behave like normal properties, and you cannot set expectations on them.

If you want to verify the behavior of the code under test, you use a mock with the appropriate expectation, and verify that. If you want just to pass a value that may need to act in a certain way, but is not the focus of this test, you use a stub.



Note:  
A stub never causes a test to fail.

Stubs and mocks can be implemented manually.

**Problem:** A lot of additional test code is necessary and it is difficult to maintain.

**Solution:** Dynamically created mocks using a mocking framework, that is, `Sinon.JS`.

`Sinon.JS` is a mocking framework for JavaScript that has the following features:

- Provides support for spies, stubs and mocks.
- Supplies higher-level test doubles for timers and AJAX requests.
- Is included in the SAP UI5 libraries.

The following figure provides an overview of QUnit.

Q

• QUnit is a JavaScript unit and integration test framework  
 • QUnit is capable of testing any generic JavaScript code  
 • It supports asynchronous tests out-of-the-box  
 • It can be found at: <http://api.qunitjs.com/>



Figure 177: QUnit

## QUnit Test Example

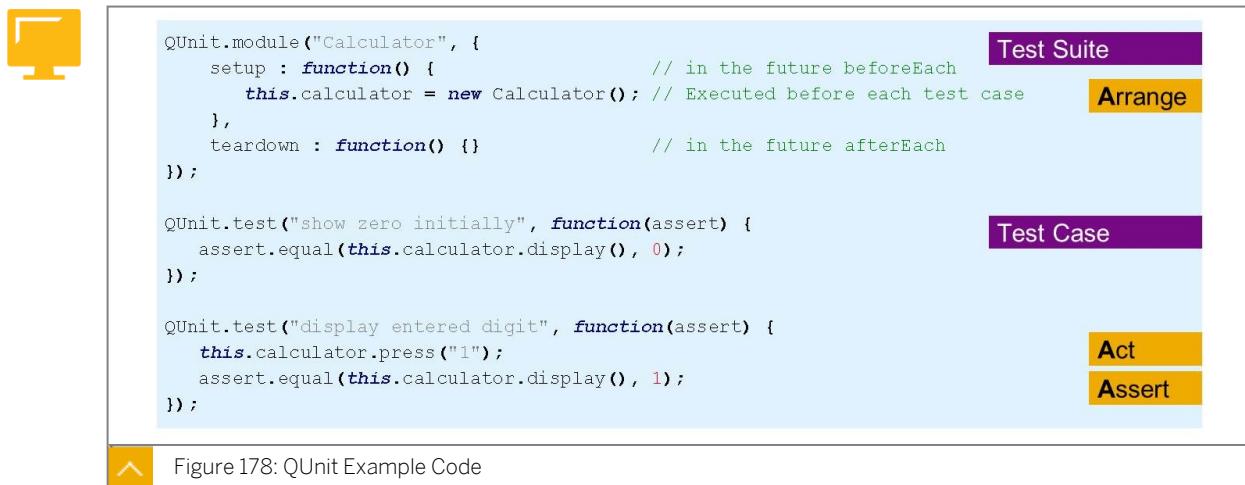
QUnit uses a set of top-level functions to provide semantic meaning in unit tests.

Just like most of unit test frameworks, it follows an arrange-act-assert pattern, a common test pattern employed when unit testing. Its purpose is to make a clear distinction between the set up for the test, the action that is to be tested, and the evaluation of the results.

Important constructs include:

- `module(string)` - defines a module.

- `test(string, function)` - defines a test.
- `ok(boolean, string)` - validates to true or false.
- `equal(value1, value2, message)` - compares two values, using the double-equal comparator.
- `deepEqual(value1, value2, message)` - compares two values based on their content, not just their identity.
- `strictEqual(value1, value2, message)` - strictly compares two values, using the triple-equal comparator.



```

QUnit.module("Calculator", {
 setup : function() { // in the future beforeEach
 this.calculator = new Calculator(); // Executed before each test case
 },
 teardown : function() {} // in the future afterEach
});

QUnit.test("show zero initially", function(assert) {
 assert.equal(this.calculator.display(), 0);
});

QUnit.test("display entered digit", function(assert) {
 this.calculator.press("1");
 assert.equal(this.calculator.display(), 1);
});

```

Figure 178: QUnit Example Code

## Test Double

For an introduction to mocks and stubs, refer to the “Unit Test with QUnit” section at the beginning of this lesson.



### Note:

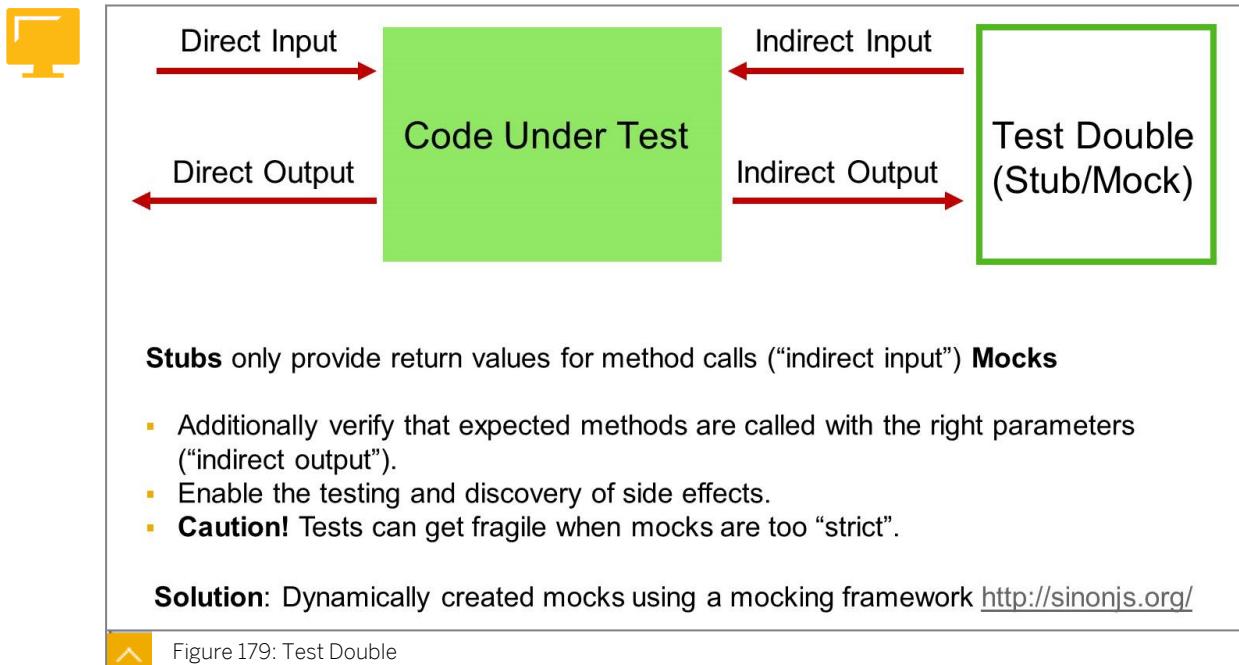
A stub never causes a test to fail.

Stubs and mocks can be implemented manually.

**Problem:** A lot of additional test code is necessary and it is difficult to maintain.

**Solution:** Dynamically created mocks using a mocking framework, that is, Sinon.JS.Sinon.JS is a mocking framework for JavaScript that has the following features:

- Provides support for spies, stubs and mocks.
- Supplies higher-level test doubles for timers and AJAX requests.
- Is included in the SAP UI5 libraries.



## Test Page

The following figure shows the code for an app to be tested.

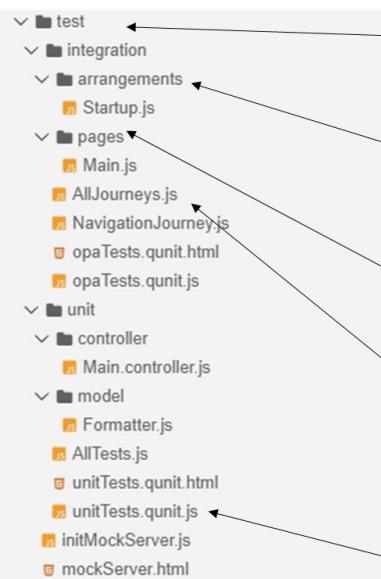
```

Main.controller.js x
1 sap.ui.define([
2 "sap/ui/core/mvc/Controller"
3], function (Controller) {
4 "use strict";
5
6 return Controller.extend("com.sap.training.ux402.qunit.UX402_QUnit.controller.Main", {
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 });

```

Figure 180: App Code Example – Prettydate.js

The following figure describes the content of a test page.



By default, the SAPUI5-project template contains a test folder.

By convention, the arrangements for the tests like startup methods etc. It has to be referenced in the AllJourneys.js file.

Contains the page objects which represents the UI under test. Contains locators, actions and assertions.

The test suites are collected in the AllJourneys.js file.

The unitTests file sets up the SAPUI5 environment and triggers the AllTests file to start the tests.

Figure 181: Required Test Page Content

The qunit-test suite of a project is separated into different files with different responsibility.



```

unitTests.qunit.html ×
1 !DOCTYPE html
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Unit tests for student00.com.sap.training.ux402.qunit/ux402_qunit</title>
6
7 <script id="sap-ui-bootstrap"
8 src="../../resources/sap/ui/core.js"
9 data-sap-ui-resourceroots='{
10 "student00/com/sap/training/ux402/qunit/ux402_qunit": "../../"
11 }'
12 </script>
13
14 <link rel="stylesheet" type="text/css" href="../../resources/sap/ui/thirdparty/qunit-2.css">
15
16 <script src="../../resources/sap/ui/thirdparty/qunit-2.js"></script>
17 <script src="../../resources/sap/ui/qunit/qunit.js"></script>
18 <script src="../../resources/sap/ui/qunit/qunit-coverage.js"></script>
19
20 <script src="unitTests.qunit.js"></script>
21 </head>
22 <body>
23 <div id="qunit"></div>
24 <div id="qunit-fixture"></div>
25 </body>
26 </html>
27

```

```

unitTests.qunit.js ×
1 /* global QUnit */
2 QUnit.config.autostart = false;
3
4 sap.ui.getCore().attachInit(function () {
5 "use strict";
6
7 sap.ui.require([
8 "student00/com/sap/training/ux402/qunit/ux402_qunit/test/unit/AllTests"
9], function () {
10 QUnit.start();
11 });
12 });

```

Figure 182: Test Page Example (1 of 2)



```
AllTests.js x
1 sap.ui.define([
2 "student00/com/sap/training/ux402/qunit/ux402_qunit/test/unit/controller/Main.controller",
3 "student00/com/sap/training/ux402/qunit/ux402_qunit/test/unit/model/Formatter"
4], function () {
5 "use strict";
6 });

Formatter.js x
1 sap.ui.define([
2 "student00/com/sap/training/ux402/qunit/ux402_qunit/model/Formatter"
3], function (Formatter) {
4 "use strict";
5
6 QUnit.module("Formatter", {
7 beforeEach: function () {
8 this._formatter = Formatter;
9 }
10);
11
12 QUnit.test("Test the pretty date implementation", function (assert) {
13 var now = "2008/01/28 22:25:00";
14 assert.equal(this._formatter.prettyDate(now, "2008/01/28 22:25:00"), "just now");
15 assert.equal(this._formatter.prettyDate(now, "2008/01/28 22:24:00"), "1 minute ago");
16 assert.equal(this._formatter.prettyDate(now, "2008/01/28 22:22:00"), "3 minutes ago");
17 assert.equal(this._formatter.prettyDate(now, "2008/01/28 21:25:00"), "1 hour ago");
18 assert.equal(this._formatter.prettyDate(now, "2008/01/27 22:25:00"), "Yesterday");
19 assert.equal(this._formatter.prettyDate(now, "2008/01/26 22:25:00"), "2 days ago");
20 assert.equal(this._formatter.prettyDate(now, "2007/01/26 22:25:00"), undefined);
21 });
22 });

```

Figure 183: Test Page Example (2 of 2)



```
Main.controller.js x
1 /*global QUnit*/
2
3 sap.ui.define([
4 "student00/com/sap/training/ux402/qunit/ux402_qunit/controller/Main.controller"
5], function (Controller) {
6 "use strict";
7
8 QUnit.module("Main Controller");
9
10 QUnit.test("I should test the Main controller", function (assert) {
11 var oAppController = new Controller();
12 oAppController.onInit();
13 assert.ok(oAppController);
14 });
15
16 });

```

Figure 184: Test Page Example Continued

## Unit Test Run Configuration

The following figure shows the project settings for configuring a unit test run in SAP Business Application Studio.



Select a npm script

```
start fiori run --open 'test/flpSandbox.html?sap-client=100#student00comsaptrainingux402masterdetailu:
start-local fiori run --config ./ui5-local.yaml --open 'test/flpSandbox.html?sap-client=100#student00coms:
start-noflp fiori run --open 'index.html?sap-client=100'
start-mock fiori run --open 'test/flpSandboxMockServer.html?sap-client=100#student00comsaptrainingu:
start-variants-management fiori run --open "preview.html?fiori-tools-rta-mode=true&sap-ui-rta-skip-flex-
unit-tests fiori run --open test/unit/unitTests.qunit.html
int-tests fiori run --open test/integration/opaTests.qunit.html
```

Figure 185: SAP Business Application Studio – Run Unit Tests

To execute the unit test the script *unit-test* is used. To execute the script select your project and choose *preview application*.

## Unit Test Run

When the unit test completes, a results screen is displayed. The following figure shows the results of the unit test run.



Unit tests for student00.com.sap.training.ux402.qunit/ux402\_qunit

QUnit 2.3.2; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36

2 tests completed in 11 milliseconds, with 1 failed, 0 skipped, and 0 todo.

7 assertions of 8 passed, 1 failed.

1. Main Controller: I should test the Main controller (1) ok

2. Formatter: Test the pretty date implementation (1, 1, 7) Run

1. okay	0 ms
2. okay	0 ms
3. failed	1 ms
Expected: "3 minutes ago"	
Result: "0 minutes ago"	
Diff: "38 minutes ago"	
Source: at Object.eval (https://aa23683ftrial-workspaces-ws-gtthk- app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingux402qunitux402_qunit/test/unit/model/Formatter.js?eval:16:10)	
4. okay	4 ms
5. okay	4 ms
6. okay	5 ms
7. okay	5 ms

Source at eval (https://aa23683ftrial-workspaces-ws-gtthk app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingux402qunitux402\_qunit/test/unit/model/Formatter.js?eval:12:11) at  
<https://sapui5.hana.ondemand.com/resources/sap-ui-core.js:106:1139> at requireAll (<https://sapui5.hana.ondemand.com/resources/sap-ui-core.js:105:657>) at executeModuleDefinition (<https://sapui5.hana.ondemand.com/resources/sap-ui-core.js:106:831>) at Object.  
 uSDefine [as define] (<https://sapui5.hana.ondemand.com/resources/sap-ui-core.js:107:1094>) at eval ([https://aa23683ftrial-workspaces-ws-gtthk-  
app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingux402qunitux402\\_qunit/test/unit/model/Formatter.js?eval:1:8](https://aa23683ftrial-workspaces-ws-gtthk-<br/>app1.eu10.trial.applicationstudio.cloud.sap/student00comsaptrainingux402qunitux402_qunit/test/unit/model/Formatter.js?eval:1:8)) at eval ()

Figure 186: Test Results



## LESSON SUMMARY

You should now be able to:

- Implement unit tests



## Implementing One-Page Acceptance (OPA) Tests



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Implement OPA tests

### Integration Tests with OPA5

The following figure defines OPA5 and describes its usage for the development of integration tests.



- OPA5 is the abbreviation for One-Page Acceptance Test.
- It is a JavaScript-based test framework for SAPUI5 that allows the development of:
  - User interaction tests
  - SAPUI5 integration tests
  - Navigation tests
  - Databinding
- OPA5 is fully supported for execution on any mobile device.
- The developer writes tests during development.
- Test-Driven-Development (TDD) results in less fragile code.
- Using the TDD approach results in sustainable code.



Figure 187: Integration Tests with OPA5

### OPA5 Advantages and Limitations

The following figure shows the advantages of using OPA5 for integration testing.



- Advantages of OPA5:
  - \_ Quick and easy access to JavaScript functions.
  - \_ Easy ramp-up as it can be used with any JavaScript unit test framework, such as QUnit or Jasmine.
  - \_ Use of the same runtime enables debugging.
  - \_ Good SAPUI5 integration.
  - \_ Feedback within seconds makes it possible to execute tests directly following a change.
  - \_ Asynchronicity is handled with polling instead of timeouts, which makes it faster.
  - \_ It enables Test-Driven Development (TDD).



Figure 188: Advantages of OPA5

The following figure shows some of the limitations when using OPA5.



- Screen capturing.
- Testing across more than one page.
- Remote test execution.
- End-to-end test are not recommended with OPA5 due to authentication issues and the fragility of test data.



Figure 189: OPA5 Limitations

## OPA5 and Bootstrapping

The following figure shows the bootstrapping.

```

opaTests.qunit.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Integration tests for Basic Template</title>
6
7 <script id="sap-ui-bootstrap"
8 src="../../resources/sap/ui/core.js"
9 data-sap-ui-theme='sap_belize'
10 data-sap-ui-resourceroots='{
11 "student00.com.sap.training.ux402.opa": "../../"
12 }'
13 data-sap-ui-animation="false"
14 data-sap-ui-compatVersion="edge"
15 data-sap-ui-async="true">
16 </script>
17
18 <link rel="stylesheet" type="text/css" href="../../resources/sap/ui/thirdparty/qunit-2.css">
19
20 <script src="../../resources/sap/ui/thirdparty/qunit-2.js"></script>
21 <script src="../../resources/sap/ui/qunit/qunit-junit.js"></script>
22
23 <script src="opaTests.qunit.js"></script>
24 </head>
25 <body>
26 <div id="qunit"></div>
27 <div id="qunit-fixture"></div>
28 </body>
29 </html>

```

```

opaTests.qunit.js x
1 /* global QUnit */
2 QUnit.config.autostart = false;
3
4 sap.ui.getCore().attachInit(function () {
5 "use strict";
6
7 sap.ui.require([
8 "student00.com.sap.training.ux402.opa/ux402_opa/test/integration/AllJourneys"
9], function () {
10 QUnit.start();
11 });
12 });

```

Figure 190: The Bootstrapping

## OPA5 Library Loading and Test Initiation

The following figure shows the loading of the required libraries and the `QUnit.start()` call to start the test.

```

AllJourneys.js x
1 sap.ui.define([
2 "sap/ui/test/Opa5",
3 "./arrangements/Startup",
4 "./NavigationJourney"
5], function (Opa5, Startup) {
6 "use strict";
7
8 Opa5.extendConfig({
9 arrangements: new Startup(),
10 viewNamespace: "student00.com.sap.training.ux402.opa.ux402_opa.view.",
11 autoWait: true
12 });
13 });

```

```

Startup.js x
1 sap.ui.define([
2 "sap/ui/test/Opa5"
3], function (Opa5) {
4 "use strict";
5
6 return Opa5.extend("student00.com.sap.training.ux402.opa.ux402_opa.test.integration.arrangements.Startup", {
7 iStartMyApp: function (oOptionsParameter) {
8 var oOptions = oOptionsParameter || {};
9
10 // start the app with a minimal delay to make tests fast but still async to discover basic timing issues
11 oOptions.delay = oOptions.delay || 50;
12
13 // start the app UI component
14 this.iStartMyUIComponent({
15 componentConfig: {
16 name: "student00.com.sap.training.ux402.opa.ux402_opa",
17 async: true
18 },
19 hash: oOptions.hash,
20 autoWait: oOptions.autoWait
21 });
22 }
23 });
24 });
25 });

```

Figure 191: Load Required Libraries and Start the Test

The *AllJourneys.js* file is responsible for load the test resources. Before UI-tests are possible the application under test has to be loaded. The component that wraps the application is loaded in the *Startup.js* file. After the arrangement is done the *NavigationJourney* will be executed.



```

NavigationJourney.js ×
 1 /*global QUnit*/
 2
 3 sap.ui.define([
 4 "sap/ui/test/opaQunit",
 5 "./pages/Main"
 6], function (opaTest) {
 7 "use strict";
 8
 9 QUnit.module("Navigation Journey");
 10
 11 opaTest("Should see the initial page of the app", function (Given, When, Then) {
 12 // Arrangements
 13 Given.iStartMyApp();
 14
 15 // Assertions
 16 Then.onTheAppPage.iShouldSeeTheApp();
 17 Then.onTheAppPage.iShouldFindAButton();
 18
 19 // Cleanup
 20 Then.iTeardownMyApp();
 21 });
 22});
 23

```

Figure 192: Navigation Journey

The *NavigationJourney.js* file contains the first test case. It will test if the application was started, using a Given. After that the assertions are processed.

## OPA5 Anatomy of a Test Case

The following two figures describe the anatomy of an OPA5 test case.



```

Matcher_TestCase.js ×
 1 sap.ui.require([
 2 "sap/ui/test/Opa5",
 3 "sap/ui/test/opaQunit",
 4 "sap/ui/test/matchers/PropertyStrictEquals",
 5 "sap/ui/test/matchers/Properties",
 6 "sap/ui/test/matchers/Ancestor"
 7], function(Opa5, opaTest, PropertyStrictEquals, Properties, Ancestor) {
 8 QUnit.module("Matchers");
 9
 10
 11
 12
 13 });

```

Figure 193: OPA5 Anatomy of a Test Case(1 of 2)



OPA gives you the following three objects:

**Given = arrangements**

**When = actions**

**Then = assertions**

```
opaTest("ComponentTest", function (Given, When, Then) {
 Given.iStartMyUIComponent({
 componentConfig:{name: "sap.training.opatest.integration.button"}
 }).done(
 function() {
 Opa5.assert.ok(jQuery(".sapUiOpaComponent").length,"The component was loaded");
 }).done(function() {
 Then.iTeardownMyUIComponent().done(function() {
 Opa5.assert.ok(!jQuery(".sapUiOpaComponent").length,"The component was removed");
 });
 });
 Opa5.emptyQueue();
});
//QUnit.start();
```

Figure 194: OPA5 Anatomy of a Test Case (2 of 2)

## OPA5 Control Retrieval by ID

The following figure shows the key aspects involved in the retrieval of a control by ID.



ViewName and UI-control id

Trigger an event of UI-Control

```
When.waitFor({
 viewName:"sap.training.opatest.view.Main",
 id:"helloButton",
 success : function(oButton) {
 * oButton.$().trigger("press");
 Opa5.assert.ok(oButton.getId(),"Button with the given ID found");
 },
 errorMessage: "Did not find the hello-Button"
});
```

Assert when found

Figure 195: Retrieving a Control by ID

## OPA5 Control Retrieval without ID

The follow figures show Matchers retrieving a control without and ID.



```

opaTest("Should find a Button with a matching property", function(Given, When, Then) {
 Given.iStartMyAppInAFrame("../index.html");

 When.waitFor({
 viewName: "sap.training.opatest.view.Main",
 controlType: "sap.m.Button",
 matchers: new PropertyStrictEquals({
 name: "text",
 value: "Press me"
 }),
 success: function(aButtons) {
 Opa5.assert.ok(true, "Found the button: " + aButtons[0]);
 },
 errorMessage: "Did not find the button with the property Text equal to Changed text"
 });

 Then.waitFor({
 id: new RegExp("helloButton"),
 success: function(aButtons) {
 Opa5.assert.ok(true, "Found the button: " + aButtons[0]);
 },
 errorMessage: "Did not find the button with corresponding ID"
 });

 Then.iTeardownMyAppFrame();
});

```



Figure 196: Matchers – Retrieving a Control without an ID (1 of 3)



```

opaTest("Should find a Button with a matching property", function(Given, When, Then) {
 Given.iStartMyAppInAFrame("../index.html");

 When.waitFor({
 viewName: "sap.training.opatest.view.Main",
 controlType: "sap.m.Button",
 matchers: new PropertyStrictEquals({
 name: "text",
 value: "Press me"
 }),
 success: function(aButtons) {
 Opa5.assert.ok(true, "Found the button: " + aButtons[0]);
 },
 errorMessage: "Did not find the button with the property Text equal to Changed text"
 });

```



Figure 197: Matchers – Retrieving a Control without an ID (2 of 3)



```

Then.waitFor({
 id: new RegExp("helloButton"),
 success: function(aButtons) {
 Opa5.assert.ok(true, "Found the button: " + aButtons[0]);
 },
 errorMessage: "Did not find the button with corresponding ID"
});

Then.iTeardownMyAppFrame();
});

```



Figure 198: Matchers – Retrieving a Control without an ID (3 of 3)



## LESSON SUMMARY

You should now be able to:

- Implement OPA tests



# Learning Assessment

1. What is the relation between the model and the controller in the standard MVC implementation?

*Choose the correct answers.*

- A The controller modifies the model.
- B The model notifies the controller about changes.
- C The model updates the controller.
- D The controller sets the model visibility.

2. Which model types are supported by SAPUI5?

*Choose the correct answers.*

- A JSON model
- B Resource model
- C Translation model
- D XML Model
- E OData model

3. Which OData versions are currently supported by SAPUI5?

*Choose the correct answers.*

- A OData V2
- B OData V5
- C OData V4
- D OData V7

4. Which binding modes are supported by SAPUI5?

*Choose the correct answers.*

- A One-time
- B One-way
- C Single-time-only
- D Two-way
- E Once

5. In which base class implementation is the `setModel` function implemented?

*Choose the correct answer.*

- A `sap.ui.base.ManagedObject`
- B `sap.ui.base.Object`
- C `sap.ui.base.Interface`
- D `sap.ui.model.base.BaseModel`

6. What is the data binding used for?

---

---

---

7. Which of the following are best practices when developing a SAPUI5 app?

*Choose the correct answers.*

- A Describe your app using a set of metadata.
- B Use the Synchronous Model Definition (SMD) syntax.
- C Minimize the code in the `index.html` file.
- D Make use of patterns.
- E Use an asynchronous model definition in your JavaScript code.

8. What approach gives you the most flexibility for your SAPUI5 app?

*Choose the correct answer.*

- A Using a controller-based approach.
- B Using a view-only based approach.
- C Implement complex UIs in the `index.html` file.
- D Using a component-based approach.

9. Which control is used in the `index.html` file to support letterboxing if required?

*Choose the correct answer.*

- A `sap.ui.core.ComponentContainer`
- B `sap.ui.core.Component`
- C `sap.m.Shell`
- D `sap.ui.core.View`

10. What configuration steps are necessary to define a navigation route?

*Choose the correct answers.*

- A Configure a route in the `manifest.json` file.
- B Configure a target.
- C Assign at least one target to the route.
- D Activate routing in the `sap.app`.

11. Why does it make sense to use the navigation API of SAPUI5 and concepts such as `eventbus` or the `navcontainer` functions of the base application?

*Choose the correct answers.*

- A Using the navigation API, it is possible to use bookmarks.
- B The `eventbus` concept is deprecated and should no longer be used.
- C Using the navigation API, the configuration of routes and targets are clearly separated from the application implementation.
- D The `NavContainer` of the `App` object is not accessible inside a component-based app.

12. From which control does the `sap.m.App` control inherit navigation capabilities?

*Choose the correct answer.*

- A `sap.m.NavigationContainer`
- B `sap.m.NavContainer`
- C `sap.ui.core.NavContainer`

13. Is it true to say that the `sap.m.App` control does not provide responsive behavior?

*Choose the correct answer.*

- A That is true, the `sap.m.App` control does not provide responsive behavior.
- B It depends on the SAPUI5 version. From version 1.30 and later, the `sap.m.App` control provides responsive behavior.
- C No, that is not true. The `sap.m.App` control provides responsive behavior.

14. What is the name of the aggregation of the `sap.m.App` control, or more precisely the name of the `sap.m.NavContainer` control, in which UI controls are aggregated?

*Choose the correct answer.*

- A `fullPages`
- B `pages`
- C `masterPages`
- D `detailPages`

15. When is the use of a master-detail pattern not recommended?

*Choose the correct answers.*

- A You need to offer complex filters for the master list of items.
- B You want to display a single object.
- C When you want to display different facets of the same object, data, or both.

16. Which modes are provided by the `sap.m.SplitAppMode` enumeration?

*Choose the correct answers.*

- A** ShowHideMode
- B** HideShowMode
- C** StretchCompressMode
- D** PopoverMode
- E** ShrinkCozyMode

17. What are the two aggregations provided by `sap.m.SplitApp` control to add page implementations?

*Choose the correct answers.*

- A** mainPages
- B** masterPages
- C** infoPages
- D** detailPages
- E** detail

18. What message types are known to SAPUI5?

*Choose the correct answers.*

- A** Control message
- B** UI message
- C** Server message
- D** Log message

19. How do you access the `MessageManager`?

*Choose the correct answer.*

- A** It is a singleton and can be accessed by the `getMessageManager` function on the core object.
- B** You have to instantiate the `MessageManager` using the constructor function.
- C** Each UI control provides a function to access the `MessageManager`.

20. In what configuration area of the `manifest.json` file can you activate automatic message creation?

*Choose the correct answer.*

- A `sap.app`
- B `sap.ui`
- C `sap.ui5`

21. What layout control is used to achieve flexible and responsive layouts?

*Choose the correct answer.*

- A `VerticalLayout`
- B `FlexBox`
- C `Grid`
- D `Splitter`

22. What aspects of the runtime environment can be accessed by the `Device API` of SAPUI5?

*Choose the correct answers.*

- A Operating system
- B Screen size
- C Orientation change
- D Language
- E Touch-specific features

23. What are the different content densities provided by SAPUI5?

*Choose the correct answers.*

- A `cozy`
- B `large`
- C `condensed`
- D `strict`
- E `compact`

24. What is the best approach to showing a UI control on a desktop only, and not on a mobile device?

*Choose the correct answer.*

- A Use the `Device` API to check the environment and call `setVisible` on the UI control.
- B Use the UI control from `sap.ui.commons`. These UI controls can handle this automatically.
- C Use the standard CSS class `sapUiVisibleOnlyOnDesktop`.

25. You want to define a property with the name `width` to enhance a standard UI5 control.

The property should hold the current width of the UI control. What is the best approach to defining the type of such a property?

*Choose the correct answer.*

- A Define the property `width` of the type `string`.
- B Define the property `width` of the type `sap.ui.core.Integer`.
- C Define the property `width` of the type `sap.ui.core.CSSSize`.
- D Define the property `width` of the type `sap.ui.core.type.CSSSize`.

26. Which function must be called inside a control renderer to add the control ID to the DOM tree and support eventing?

*Choose the correct answer.*

- A `writeClasses`
- B `writeIcon`
- C `writeControlData`
- D `write`

27. Which of the following statements are true with regard to implementing your own renderer?

*Choose the correct answers.*

- A Implement the `render` function inside the control.
- B Implement a `renderer` class, derived from `sap.ui.core.Renderer`, in a separate file.
- C Implement the renderer using AMD syntax.
- D Assign a reference to the `renderer` property of the UI control.

28. Which of the following aspects are true for a SAPUI5 UI element?

*Choose the correct answers.*

- A A UI element has an API.
- B A UI element does not have a renderer.
- C A UI element has a renderer.
- D A UI element can have events.

29. What is the base class for all UI controls in SAPUI5?

*Choose the correct answer.*

- A sap.ui.Control
- B sap.ui.core.Control
- C sap.ui.base.Control
- D sap.ui.Element

30. How can a renderer access the associated elements?

*Choose the correct answer.*

- A The developer must implement an appropriate function to access the elements.
- B SAPUI5 provides functions to access all properties, associations, and aggregations.
- C The developer must define a property method in the control metadata and declare the access control.

31. Which file contains the initialization code for the UI library?

*Choose the correct answer.*

- A library.load.js
- B library.js
- C loadlibrary.js
- D lib.dll

32. What method is called inside the `library.js` file?

*Choose the correct answer.*

- A `sap.ui.getCore().registerLibrary`
- B `sap.ui.getCore().initLibrary`
- C `sap.ui.getCore().loadLibrary`
- D `sap.ui.getCore().runLibrary`

33. Which of the following statements are true with respect to QUnit?

*Choose the correct answers.*

- A Supports only synchronous testing out of the box.
- B QUnit is a JavaScript unit and integration test framework.
- C Supports asynchronous tests out-of-the-box.
- D Is capable of testing any generic JavaScript code.

34. Does QUnit support SAPUI5 view tests?

*Choose the correct answer.*

- A Yes, you can implement a test class to test UI aspects of SAPUI5.
- B No, for UI tests you must use OPA5.
- C You can use the QUnit-extensions, called Selenium, to test SAPUI5 controls.

35. Which of the following statements are true with regard to OPA5?

*Choose the correct answers.*

- A Can be used for user interaction tests.
- B Can be used for SAPUI5 integration tests.
- C Is a view controller test framework.
- D Provides the possibility to test navigation.

## Learning Assessment - Answers

1. What is the relation between the model and the controller in the standard MVC implementation?

*Choose the correct answers.*

- A The controller modifies the model.
- B The model notifies the controller about changes.
- C The model updates the controller.
- D The controller sets the model visibility.

Correct. The controller modifies the model and the model notifies the controller about changes.

2. Which model types are supported by SAPUI5?

*Choose the correct answers.*

- A JSON model
- B Resource model
- C Translation model
- D XML Model
- E OData model

Correct. SAPUI5 supports the JSONModel, XMLModel, and ODataModel as data models and the ResourceModel for internationalization (i18n) data.

3. Which OData versions are currently supported by SAPUI5?

*Choose the correct answers.*

- A OData V2
- B OData V5
- C OData V4
- D OData V7

Correct. SAPUI5 currently supports the OData Standard 2 and 4.

4. Which binding modes are supported by SAPUI5?

*Choose the correct answers.*

- A One-time
- B One-way
- C Single-time-only
- D Two-way
- E Once

Correct. SAPUI5 supports one-time, one-way, and two-way binding. In one-way binding, data are bound from the model to the UI aspect and changes in the UI do not update the model. In contrast, in two-way binding, changes in the UI do update the model data. SAPUI5 also supports the so called one-time binding where binding executes only once.

5. In which base class implementation is the `setModel` function implemented?

*Choose the correct answer.*

- A `sap.ui.base.ManagedObject`
- B `sap.ui.base.Object`
- C `sap.ui.base.Interface`
- D `sap.ui.model.base.BaseModel`

Correct. The `setModel` function is implemented in the `ManagedObject` class. `ManagedObject` is one of the base classes of many other classes in SAPUI5, such as, `Component`, `Control`, `Element`, `View`, `Fragment`, and many more.

6. What is the data binding used for?

Data Binding is used to bind two data sources together and keeping them in sync.

---

7. Which of the following are best practices when developing a SAPUI5 app?

*Choose the correct answers.*

- A Describe your app using a set of metadata.
- B Use the Synchronous Model Definition (SMD) syntax.
- C Minimize the code in the `index.html` file.
- D Make use of patterns.
- E Use an asynchronous model definition in your JavaScript code.

Correct. It is good practice to use standard patterns while implementing SAPUI5 applications to strengthen the user experience of your apps. An application should be described by a set of metadata using the `manifest.json` file and minimize the complexity of code in the `index.html`. This makes your application much more reusable and easier to integrate into the SAP Fiori launchpad. Also, it is a common practice to use Asynchronous Model Definition (AMD) syntax to describe and resolve dependencies between different JavaScript resources.

8. What approach gives you the most flexibility for your SAPUI5 app?

*Choose the correct answer.*

- A Using a controller-based approach.
- B Using a view-only based approach.
- C Implement complex UIs in the `index.html` file.
- D Using a component-based approach.

Correct. The encapsulation of your application as a component provides you the most flexibility. Using the component approach makes it very easy to integrate the application into other applications and into the SAP Fiori launchpad.

9. Which control is used in the `index.html` file to support letterboxing if required?

*Choose the correct answer.*

- A `sap.ui.core.ComponentContainer`
- B `sap.ui.core.Component`
- C `sap.m.Shell`
- D `sap.ui.core.View`

Correct. The `sap.m.Shell` control handles visual adaptation.

10. What configuration steps are necessary to define a navigation route?

*Choose the correct answers.*

- A Configure a route in the `manifest.json` file.
- B Configure a target.
- C Assign at least one target to the route.
- D Activate routing in the `sap.app`.

Correct. The routing aspects are configured inside the `manifest.json` file. During the routing configuration, you must configure targets. A navigation route must be assigned to at least one target.

11. Why does it make sense to use the navigation API of SAPUI5 and concepts such as `eventbus` or the `navcontainer` functions of the base application?

*Choose the correct answers.*

- A Using the navigation API, it is possible to use bookmarks.
- B The `eventbus` concept is deprecated and should no longer be used.
- C Using the navigation API, the configuration of routes and targets are clearly separated from the application implementation.
- D The `NavContainer` of the `App` object is not accessible inside a component-based app.

Correct. It is recommended to use the navigation API because using the API ensures that the user of the app is able to store bookmarks for navigation targets. Also, using the API separates the navigation routes from the application implementation. This is because the routing configuration is done inside the `manifest.json` file.

12. From which control does the `sap.m.App` control inherit navigation capabilities?

*Choose the correct answer.*

- A `sap.m.NavigationContainer`
- B `sap.m.NavContainer`
- C `sap.ui.core.NavContainer`

Correct. The `sap.m.App` control inherits navigation capabilities from the `sap.m.NavContainer`. The `NavContainer` object provides functions to navigate to a target page and return from a target page.

13. Is it true to say that the `sap.m.App` control does not provide responsive behavior?

*Choose the correct answer.*

- A That is true, the `sap.m.App` control does not provide responsive behavior.
- B It depends on the SAPUI5 version. From version 1.30 and later, the `sap.m.App` control provides responsive behavior.
- C No, that is not true. The `sap.m.App` control provides responsive behavior.

Correct. The `sap.m.App` control provides responsive behavior.

14. What is the name of the aggregation of the `sap.m.App` control, or more precisely the name of the `sap.m.NavContainer` control, in which UI controls are aggregated?

*Choose the correct answer.*

- A `fullPages`
- B `pages`
- C `masterPages`
- D `detailPages`

Correct. The `pages` aggregation works as the container for visual aspects, such as, views, pages, or carousels.

15. When is the use of a master-detail pattern not recommended?

*Choose the correct answers.*

- A You need to offer complex filters for the master list of items.
- B You want to display a single object.
- C When you want to display different facets of the same object, data, or both.

Correct. The master-detail pattern is not recommended when you have to implement an application where the user is able to apply complex filters on the data displayed in the master list. It is also not recommended when the user is able to display the data in different facets, such as charts and forms.

16. Which modes are provided by the `sap.m.SplitAppMode` enumeration?

*Choose the correct answers.*

- A ShowHideMode
- B HideShowMode
- C StretchCompressMode
- D PopoverMode
- E ShrinkCozyMode

Correct. The `SplitAppMode` enumeration provides 1) `ShowHideMode`, where the master is automatically hidden in portrait mode, 2) `StretchCompressMode`, where the master always shows but in a compressed version when in portrait mode, and 3) `PopoverMode`, where the master is shown inside a popover when in portrait mode.

17. What are the two aggregations provided by `sap.m.SplitApp` control to add page implementations?

*Choose the correct answers.*

- A mainPages
- B masterPages
- C infoPages
- D detailPages
- E detail

Correct. The `sap.m.SplitApp` control provides the `masterPages` and `detailPages` aggregations, which are borrowed from the `sap.m.SplitContainer` control.

18. What message types are known to SAPUI5?

*Choose the correct answers.*

- A Control message
- B UI message
- C Server message
- D Log message

Correct. SAPUI5 knows the control message and server message types.

19. How do you access the `MessageManager`?

*Choose the correct answer.*

- A It is a singleton and can be accessed by the `getMessageManager` function on the `core` object.
- B You have to instantiate the `MessageManager` using the constructor function.
- C Each UI control provides a function to access the `MessageManager`.

Correct. You can access the `MessageManager` from the `core` object using the `getMessageManager` function. The `getMessageManager` function returns the singleton instance of the `MessageManager`.

20. In what configuration area of the `manifest.json` file can you activate automatic message creation?

*Choose the correct answer.*

- A `sap.app`
- B `sap.ui`
- C `sap.ui5`

Correct. You can activate automatic message generation in the `sap.ui5` section of the `manifest.json` file. Here, you must set the `handleValidation` property to `true`.

21. What layout control is used to achieve flexible and responsive layouts?

*Choose the correct answer.*

- A `VerticalLayout`
- B `FlexBox`
- C `Grid`
- D `Splitter`

Correct. The `Grid` control is used to achieve flexible and responsive layouts.

22. What aspects of the runtime environment can be accessed by the Device API of SAPUI5?

*Choose the correct answers.*

- A Operating system
- B Screen size
- C Orientation change
- D Language
- E Touch-specific features

Correct. You can retrieve information about the operating system, screen size, orientation change, and touch-specific features using the Device API.

23. What are the different content densities provided by SAPUI5?

*Choose the correct answers.*

- A cozy
- B large
- C condensed
- D strict
- E compact

Correct. SAPUI5 provides a **content density** factor that allows the size of the controls to be adjusted depending on the interaction style. The `cozy` factor displays controls with dimensions large enough to enable fingertip interaction. This factor is ideal for devices operated by touch. The `compact` factor reduces the dimensions of the controls, allowing more information to be displayed on the UI. This factor is ideal for devices operated by mouse and keyboard.

24. What is the best approach to showing a UI control on a desktop only, and not on a mobile device?

*Choose the correct answer.*

- A Use the Device API to check the environment and call `setVisible` on the UI control.
- B Use the UI control from `sap.ui.commons`. These UI controls can handle this automatically.
- C Use the standard CSS class `sapUiVisibleOnlyOnDesktop`.

Correct. The best way to implement UI adaption is to use standard CSS classes such as `sapUiVisibleOnlyOnDesktop` to ensure that the UI control is only shown on a desktop and not on a mobile device.

25. You want to define a property with the name `width` to enhance a standard UI5 control. The property should hold the current width of the UI control. What is the best approach to defining the type of such a property?

*Choose the correct answer.*

- A Define the property `width` of the type `string`.
- B Define the property `width` of the type `sap.ui.core.Integer`.
- C Define the property `width` of the type `sap.ui.core.CSSSize`.
- D Define the property `width` of the type `sap.ui.core.type.CSSSize`.

Correct. You should use the `sap.ui.core.type.CSSSize` type.

26. Which function must be called inside a control renderer to add the control ID to the DOM tree and support eventing?

*Choose the correct answer.*

- A `writeClasses`
- B `writeIcon`
- C `writeControlData`
- D `write`

Correct. You must invoke the `writeControlData` function to add the control ID to the DOM tree. It is also necessary to call the function to ensure that eventing is supported.

27. Which of the following statements are true with regard to implementing your own renderer?

*Choose the correct answers.*

- A Implement the `render` function inside the control.
- B Implement a `renderer` class, derived from `sap.ui.core.Renderer`, in a separate file.
- C Implement the renderer using AMD syntax.
- D Assign a reference to the `renderer` property of the UI control.

Correct. To implement a control renderer, you must implement a class that is derived from `sap.ui.core.Renderer`. The implementation should use the AMD syntax and it is also necessary to assign the renderer implementation to the `renderer` property of your UI control implementation.

28. Which of the following aspects are true for a SAPUI5 UI element?

*Choose the correct answers.*

- A** A UI element has an API.
- B** A UI element does not have a renderer.
- C** A UI element has a renderer.
- D** A UI element can have events.

Correct. A SAPUI5 element provides an API and supports events. An element does not provide its own renderer implementation. The UI element is rendered by the renderer of the parent control.

29. What is the base class for all UI controls in SAPUI5?

*Choose the correct answer.*

- A** sap.ui.Control
- B** sap.ui.core.Control
- C** sap.ui.base.Control
- D** sap.ui.Element

Correct. The base class of all UI5 controls is the sap.ui.core.Control implementation.

30. How can a renderer access the associated elements?

*Choose the correct answer.*

- A** The developer must implement an appropriate function to access the elements.
- B** SAPUI5 provides functions to access all properties, associations, and aggregations.
- C** The developer must define a property method in the control metadata and declare the access control.

Correct. The SAPUI5 framework ensures that the renderer has access to all properties, associations, and aggregations that are defined in a SAPUI5 control.

31. Which file contains the initialization code for the UI library?

*Choose the correct answer.*

- A library.load.js
- B library.js
- C loadlibrary.js
- D lib.dll

Correct. The `library.js` file contains the initialization code for the UI library. It also contains information about the name, version, and dependencies.

32. What method is called inside the `library.js` file?

*Choose the correct answer.*

- A `sap.ui.getCore().registerLibrary`
- B `sap.ui.getCore().initLibrary`
- C `sap.ui.getCore().loadLibrary`
- D `sap.ui.getCore().runLibrary`

Correct. The developer must invoke the `sap.ui.getCore().loadLibrary` function.

33. Which of the following statements are true with respect to QUnit?

*Choose the correct answers.*

- A Supports only synchronous testing out of the box.
- B QUnit is a JavaScript unit and integration test framework.
- C Supports asynchronous tests out-of-the-box.
- D Is capable of testing any generic JavaScript code.

Correct. Qunit is a JavaScript unit and integration test framework that supports synchronous and asynchronous tests out-of-the-box. It has the capability of testing any generic JavaScript code.

34. Does QUnit support SAPUI5 view tests?

*Choose the correct answer.*

- A Yes, you can implement a test class to test UI aspects of SAPUI5.
- B No, for UI tests you must use OPA5.
- C You can use the QUnit-extensions, called Selenium, to test SAPUI5 controls.

Correct. QUnit is a JavaScript unit test framework. Is it not intended for the implementation of UI tests.

35. Which of the following statements are true with regard to OPA5?

*Choose the correct answers.*

- A Can be used for user interaction tests.
- B Can be used for SAPUI5 integration tests.
- C Is a view controller test framework.
- D Provides the possibility to test navigation.

Correct. The OPA5 framework is intended to implement user interaction tests, integration tests, and it provides the possibility to test navigation.



## UNIT 3

# Advanced Data Handling

### Lesson 1

Describing Remote vs. Local OData Services

199

### Lesson 2

Working with the MockServer

207

### Lesson 3

Working with the ODataModel

211

### Lesson 4

Describing OData Deep Inserts

221

### Lesson 5

Introducing SAPUI5 Smart Controls

225

### Lesson 6

Working with SAPUI5 Smart Controls

231

### Lesson 7

Introducing SAP Fiori Elements

237

### UNIT OBJECTIVES

- Describe how to access back-end data using the OData-specific data binding feature
- Work with the Mock Server
- Work with the OData Model
- Describe OData Deep Inserts
- Understand SAPUI5 Smart Controls
- Use SAPUI5 Smart Controls
- Get an Introduction to SAP Fiori Elements



# Describing Remote vs. Local OData Services



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe how to access back-end data using the OData-specific data binding feature

## Remote vs. Local OData Services

### Scenario

As a Business Process owner, you are requested to permanently improve the user experience in your area of responsibility.

Creating user interfaces is a critical aspect in the user experience. In this training, you will learn how to develop SAPUI5 user interfaces to induce a great user experience.

### Remote Data Use Case

The following are some basics about remote data use cases:



- Most applications will be built using remote data:
  - In other words, the model will access data that resides on a server such as an SAP system and accessed via an SAP Gateway OData service.
- Someone will create an OData service(s) consisting of:
  - Collections
  - Properties
  - Associations
- Code you write in the SAPUI5 application will make use of these entities.

### Local Data Use Case

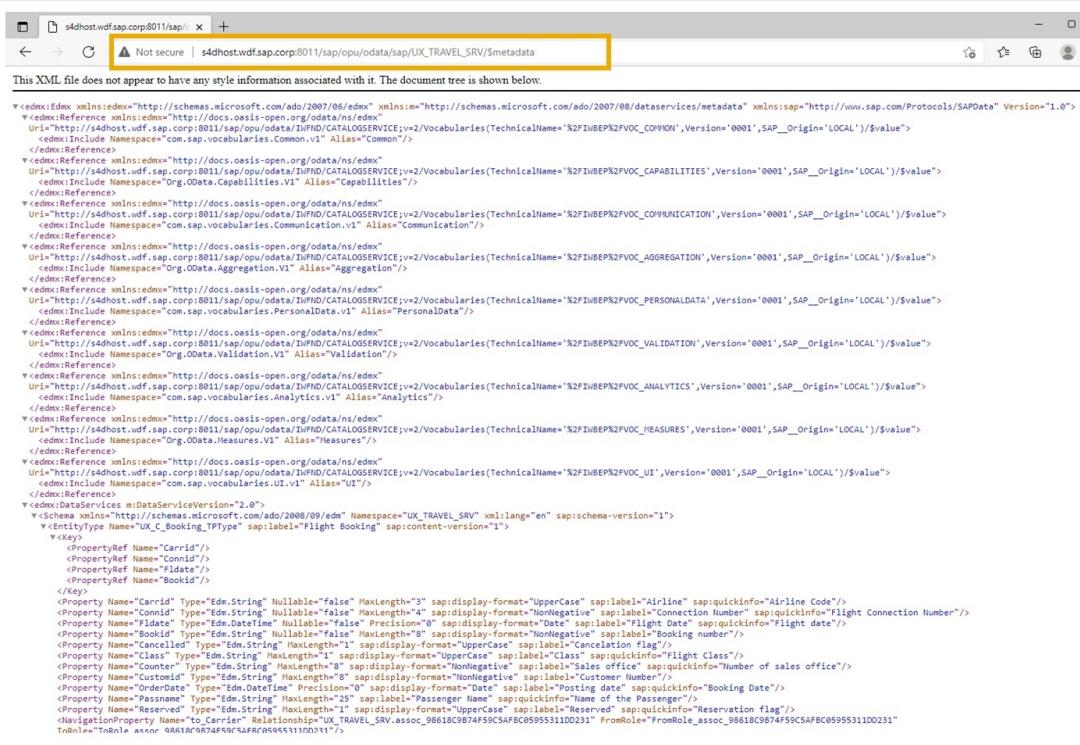
There are a variety of reasons a developer may want to create and use local data during the development cycle:



- A developer sometimes has to work offline.
- Poor development server performance.
- Developers want to perform a quick test without creating the live entities needed on the back-end server (Gateway).
- Technically, what is being shown here could be used in a production application as well.

- Production applications could fetch some of their data from local files instead of enduring the overhead of making remote requests:
  - For example, local data could be used for lookup tables/lists.

## Accessing the Metadata



The screenshot shows a browser window with the URL [http://s4host.wdf.sap.corp:8011/sap/opu/odata/sap/UX\\_TRAVEL\\_SRV/\\$metadata](http://s4host.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/$metadata). The page displays the XML structure of the OData metadata for the UX\_TRAVEL\_SRV service. The XML includes definitions for various entities, their properties, and relationships, such as \$metadata, \$entityType, \$entitySet, \$key, \$property, \$relationship, and \$navigationProperty. It also includes sections for \$ref, \$annotation, \$content, and \$schema.

Figure 199: Access the Metadata

To get the metadata of an existing service, you have to add the option \$metadata to the URL of the OData-service.

## Adding metadata.xml to Project



- The metadata.xml file is added to the SAP Business Application Studio project when an OData-Service is added to the project.
- The file is stored at folder localService.

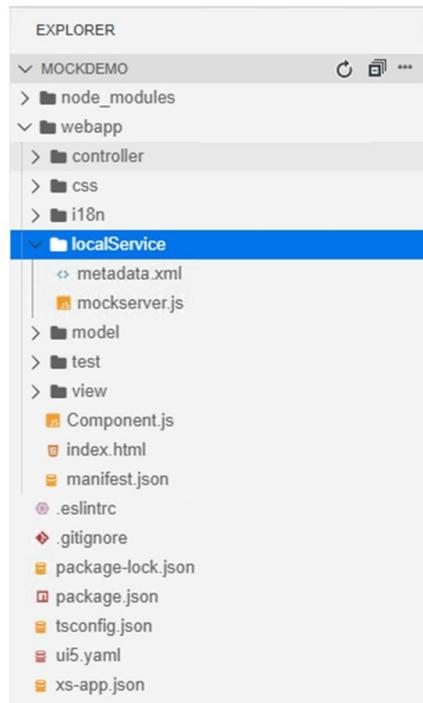


Figure 200: Adding metadata.xml to Project

When working with the SAP Business Application Studio, it is not necessary to add the *metadata.xml* file manually to the SAPUI5 project.

If you work with another development tool you have to request the *metadata.xml* file as shown in the previous slide, download the file and store the file in your project.

The file will be added by the standard SAP Fiori project templates or through the "Consume SAP Services"-command of your SAP Business Application Studio. You can find the command in the command palette View → *Find Command* or CTRL+Shift+P.

# The OData Model Editor



SAP Business Application Studio provides a text editor with intellisense for creating or editing metadata files.

Figure 201: The OData Model Editor

Sometimes when implementing a OData-Service based application, the service implementation is not finished or even did start already. To create or to adapt an existing metadatafile. The SAP Business Application Studio provides a text-based editor a OData Model Editor. The editor does have intellisense.

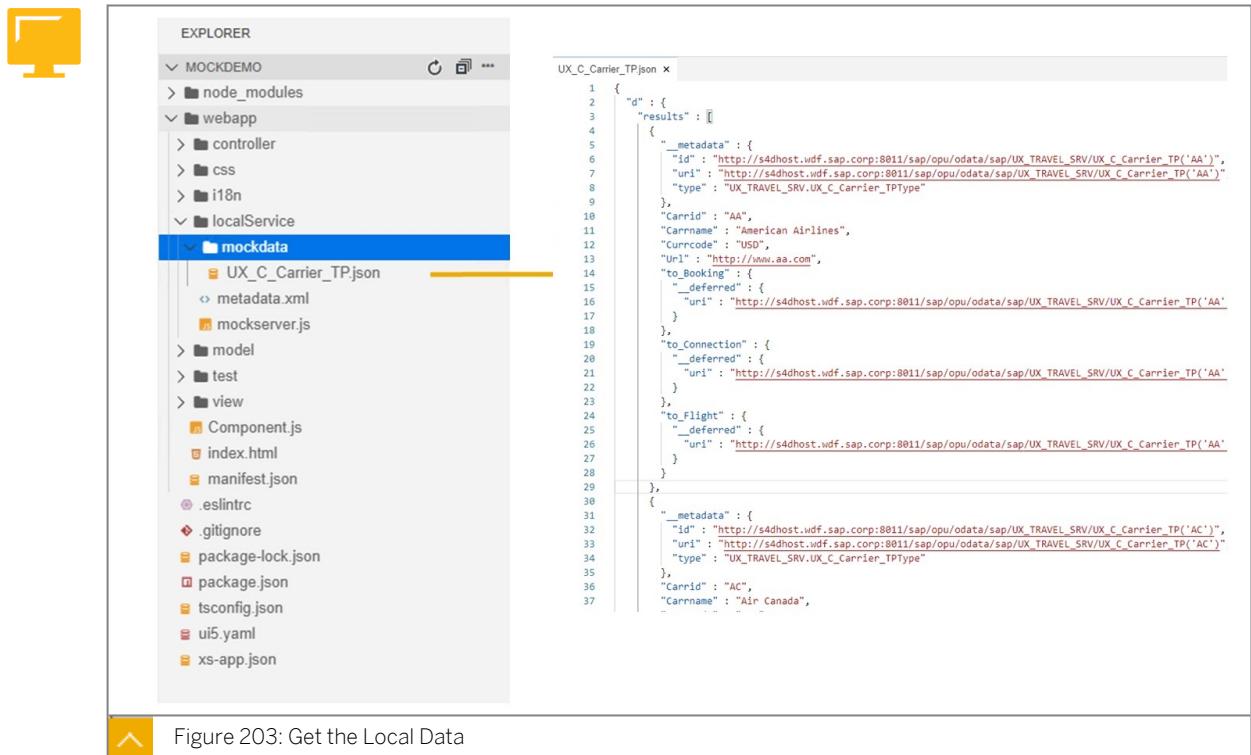
## Getting Local Data



- This JSON data could have been gotten via browser access to an OData service using the following URL and then saved to the file:

Figure 202: Get the Local Data

To request the data from the backend in the JSON-format, the developer can add the option `format=json` to the service URL.



The screenshot shows the SAP Studio interface. On the left, the Explorer view displays a project structure under 'MOCKDEMO'. A folder named 'mockdata' is selected, containing files like 'UX\_C\_Carrier\_TP.json', 'metadata.xml', and 'mockserver.js'. To the right, the code editor shows the contents of 'UX\_C\_Carrier\_TP.json'. The JSON file defines two entities: 'AA' (American Airlines) and 'AC' (Air Canada), each with properties like 'Carrid', 'Carrname', 'Currcode', and 'Url'.

```

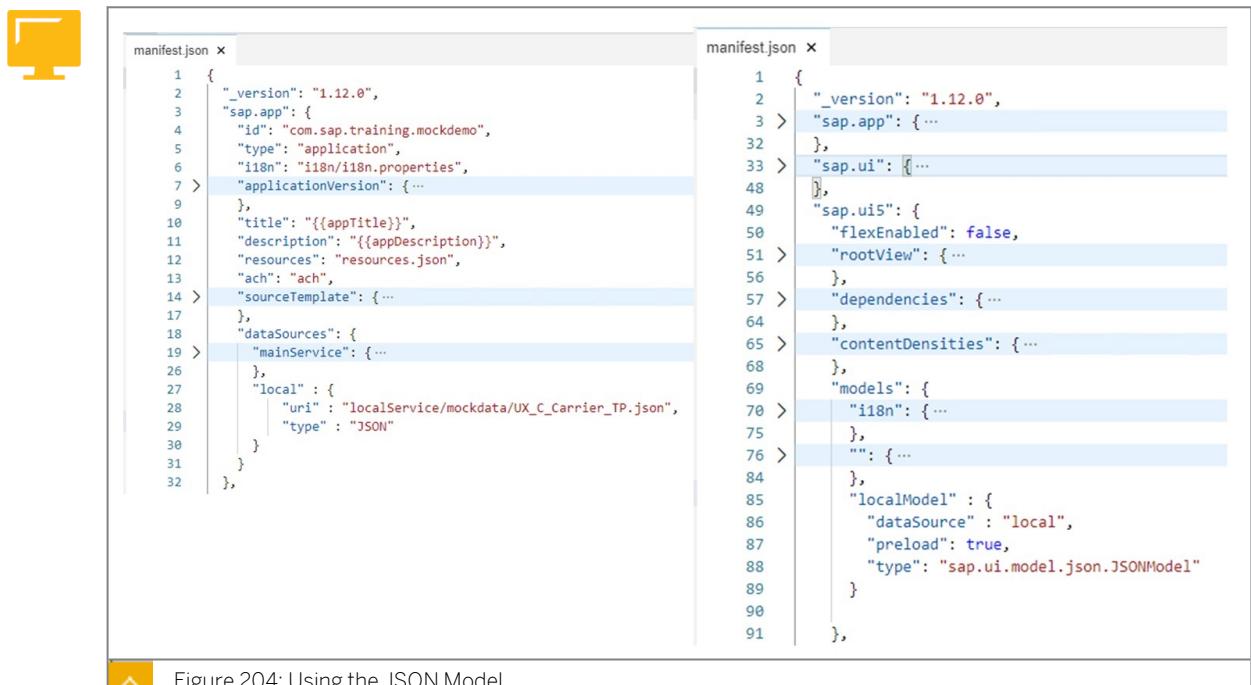
1 {
2 "d": {
3 "results": [
4 {
5 "_metadata": {
6 "id": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AA')",
7 "uri": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AA')",
8 "type": "UX_TRAVEL_SRV.UX_C_Carrier_TPType"
9 },
10 "Carrid": "AA",
11 "Carrname": "American Airlines",
12 "Currcode": "USD",
13 "Url": "http://www.ea.com",
14 "to_Booking": {
15 "_deferred": {
16 "uri": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AA')"
17 }
18 },
19 "to_Connection": {
20 "_deferred": {
21 "uri": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AA')"
22 }
23 },
24 "to_Flight": {
25 "_deferred": {
26 "uri": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AA')"
27 }
28 }
29 },
30 {
31 "_metadata": {
32 "id": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AC')",
33 "uri": "http://s4dhost.wdf.sap.corp:8011/sap/opu/odata/sap/UX_TRAVEL_SRV/UX_C_Carrier_TP('AC')",
34 "type": "UX_TRAVEL_SRV.UX_C_Carrier_TPType"
35 },
36 "Carrid": "AC",
37 "Carrname": "Air Canada",
38 "currCode": "CAD"
39 }
40]
41 }
42 }
```

Figure 203: Get the Local Data

A good place and a common practice is to store the requested business data in a folder called *mockdata* as a subfolder of the *localService* folder in the SAPUI5 project.

The file name has to reflect the name of the entity set and the file type needs to be json.

### Using the JSON Model



The screenshot shows two manifest.json files side-by-side. The left manifest.json file includes a 'mainService' section with a 'local' data source pointing to 'UX\_C\_Carrier\_TP.json'. The right manifest.json file shows the resulting configuration after the local model is processed, including the 'localModel' entry.

```

manifest.json x
1 {
2 "_version": "1.12.0",
3 "sap.app": {
4 "id": "com.sap.training.mockdemo",
5 "type": "application",
6 "i18n": "i18n/i18n.properties",
7 "applicationVersion": { ...
8 },
9 "title": "{{appTitle}}",
10 "description": "{{appDescription}}",
11 "resources": "resources.json",
12 "ach": "ach",
13 "sourceTemplate": { ...
14 },
15 "dataSources": {
16 "mainService": { ...
17 },
18 "local" : {
19 "uri" : "localService/mockdata/UX_C_Carrier_TP.json",
20 "type" : "JSON"
21 }
22 }
23 },
24 },
25 }
```

```

manifest.json x
1 {
2 "_version": "1.12.0",
3 "sap.app": { ...
4 },
5 "sap.ui": { ...
6 },
7 "sap.ui5": {
8 "flexEnabled": false,
9 "rootView": { ...
10 },
11 "dependencies": { ...
12 },
13 "contentDensities": { ...
14 },
15 "models": {
16 "i18n": { ...
17 },
18 "": { ...
19 },
20 "localModel" : {
21 "dataSource": "local",
22 "preload": true,
23 "type": "sap.ui.model.json.JSONModel"
24 }
25 },
26 }
27 },
28 },
29 "": { ...
30 },
31 },
32 },
33 }
```

Figure 204: Using the JSON Model

The most structured way using a local JSON-file is to implement a JavaScript function to the *model.js* file. The *model.js* file is located in the *model*-folder of the SAPUI5-project.

The function in the `model.js` file creates a new instance of the type `sap.ui.model.json.JSONModel`. The constructor function gets the path to the local JSON-file, the newly created JSONModel-instance is returned to the caller.



```

models.js x
 1 sap.ui.define([
 2 "sap/ui/model/json/JSONModel",
 3 "sap/ui/Device"
 4], function (JSONModel, Device) {
 5 "use strict";
 6
 7 return {
 8
 9 createDeviceModel: function () {
 10 var oModel = new JSONModel(Device);
 11 oModel.setDefaultBindingMode("OneWay");
 12 return oModel;
 13 },
 14
 15 createLocalModel : function() {
 16 let oModel = new JSONModel("./localService/mockdata/UX_C_Carrier_TP.json");
 17 return oModel;
 18 }
 19 };
 20 });
 21);

```

Figure 205: Using the JSON Model (2)

## Using the JSON Model

The implementation of the function from the `models.js` file is called by the `init` function of the Component. The returned JSONModel-object is stored to the model context of the component.

### Using the JSON Model



#### Bind the UI-control to the model

```

App.view.xml x
 1 <mvc:View controllerName="com.sap.training.mockdemo.controller.App" xmlns:mvc="sap.ui.core.mvc" displayBlock="true" xmlns="sap.m">
 2 <Shell id="shell">
 3 <App id="app">
 4 <pages>
 5 <Page id="page" title="{i18n>title}">
 6 <content>
 7 <Table id="idCarrierList"
 8 inset="false"
 9 items="{
 10 path: 'localModel>/UX_C_Carrier_TP'
 11 }"
 12 <columns>
 13 <Column width="12em">
 14 <Text text="{i18n>carrid}" />
 15 </Column>
 16 <Column minScreenWidth="Tablet" demandPopin="true">
 17 <Text text="{i18n>carriername}" />
 18 </Column>
 19 <Column minScreenWidth="Tablet" demandPopin="true" hAlign="End">
 20 <Text text="{i18n>carrierurl}" />
 21 </Column>
 22 </columns>
 23 <items>
 24 <ColumnListItem>
 25 <cells>
 26 <Text text="{localModel>Carrid}" />
 27 <Text text="{localModel>Carriername}" />
 28 </cells>
 29 </ColumnListItem>
 30 </items>
 31 </Table>
 32 </content>
 33 </Page>
 34 </pages>
 35 </App>
 36 </Shell>
 37 </mvc:View>

```

Figure 206: Using the JSON Model

The created model can now be used as a data provider inside the view.



## LESSON SUMMARY

You should now be able to:

- Describe how to access back-end data using the OData-specific data binding feature



# Working with the MockServer



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with the Mock Server

### The Mock Server

#### Using the MockServer for Back-end Simulation

**Basics about using a MockServer for backend simulation:**

- To simulate the backend system SAPUI5 provides a class `sap.ui.core.util.MockServer`.
- Using Mockserver:
  - Reference the MockServer class.
  - Prepare your metadata in xml file. You can use existing service, grab its metadata with `$metadata` command in gateway client like this `<gw-server>/<gw-service>/  
$metadata`,
  - You need to prepare JSON sample data of your entities, you can use `<gw-server>/  
<gw-service>/<entitySet>?$format=json` and delete metadata.
  - Request the necessary data from the backend using a browser and store the result locally in your project.

There would be an oData service modeled before the development phase, at least in an ideal world. We can then let our frontend UI5 developers use the mockserver instead of running oData service on SAP GW. The mockserver responds to all oData requests with some sample data, so our backend developers can feel free to experiment with SAP GW as much as they want and not to be afraid of stopping ui5 developers from work.

## How the Mock Server Works

## Understanding the MockServer - Simple Example



```
_startMockServer: function(sServiceUrl) {
 var oMockServer = new sap.ui.core.util.MockServer({
 rootUri: sServiceUrl
 });

 var iDelay = +(jQuery.sap.getUriParameters().get("responderDelay") || 0);
 sap.ui.core.util.MockServer.config({
 autoRespondAfter: iDelay
 });

 oMockServer.simulate("model/metadata.xml", "model/");
 oMockServer.start();

 sap.m.MessageToast.show("Running in demo mode with mock data.", {
 duration: 4000
 });
}.
```

Figure 207: Binding Mode Support by Model Type

The above code shows you how to implement the usage of the MockServer by your own.

While instantiating the MockServer object you have to pass the URI to the ODataService as a configuration parameter.

All requests to this URI are intercepted by the MockServer. The MockServer implementation defines a static method to configure all running MockServers-objects centrally, this method is called config.

The simulate method of the MockServer gets two parameter, the first parameter defines where to find the metadata of your ODataService and the second parameter where to find the mock data. The mock data files are defined as JSON-files and the name of the file has to be the name of the EntitySet that should be returned.

To start the MockServer, you have to invoke the `start-function`.

## Understanding the MockServer - Complex Example



```
mockserver.js x
 1 sap.ui.define([
 2 "sap/ui/core/util/MockServer",
 3 "sap/ui/model/json/JSONModel",
 4 "sap/base/util/UriParameters",
 5 "sap/base/log"
 6], function (MockServer, JSONModel, UriParameters, log) {
 7 "use strict";
 8
 9
 10 var oMockServer,
 11 _appPath = "com/sap/training/mockdemo",
 12 _jsonfilePath = _appPath + "/localService/mockdata";
 13
 14 var oMockServerInterface = {
 15
 16 /**
 17 * Initialize the mock server asynchronously.
 18 * You can configure the delay with the URL parameter "serverDelay".
 19 * The local mock data in this folder is returned instead of the real data for testing.
 20 * @protected
 21 * @param {object} [optionsParameter] init parameters for the mockserver
 22 * @returns{Promise} a promise that is resolved when the mock server has been started
 23 */
 24 init : function (optionsParameter) {
 25 var options = optionsParameter || {};
 26
 27 return new Promise(function (fnResolve, fnReject) {
 28 var $manifestUrl = sap.ui.require.toUrl(_appPath + "manifest.json"),
 29 $manifestModel = new JSONModel($manifestUrl);
 30
 31 $manifestModel.attachRequestCompleted(function () {
 32 var oUriParameters = new UriParameters(window.location.href),
 33 sJsonfilePath = _jsonfilePath.replace(/\$/g, UriParameters.getSegment(oUriParameters));
 34
 35 oManifestModel = oManifestModel.getEntry("mainService");
 36 oManifestModel.setDataSource(oManifestModel.getEntry("mainService").$dataSources.mainService);
 37 oManifestModel.setMetadataUrl(sJsonfilePath);
 38 oManifestModel.setMetadataPath(sJsonfilePath);
 39
 40 // ensure there is a trailing slash
 41 oManifestModel.setAbsoluteUri(oManifestModel.getEntry("mainService").$absoluteUri);
 42 oManifestModel.setAbsolutePath(oManifestModel.getEntry("mainService").$absolutePath);
 43
 44 // create a mock server instance or stop the existing one to reinitialize
 45 MockServer.create(oManifestModel, oManifestModel.getEntry("mainService").$absolutePath);
 46 });
 47 });
 48 }
 49 });
 50
 51
 52 // create a mock server instance or stop the existing one to reinitialize
 53 MockServer.create(oManifestModel, oManifestModel.getEntry("mainService").$absolutePath);
 54
```

Figure 208: Understanding the MockServer - Complex Example

The SAPUI5 project template used in SAP Business Application Studio provides by default a mockserver implementation.



```

var aRequests = oMockServer.getRequests();

// compose an error response for each request
var fnResponse = function (iErrCode, sMessage, aRequest) {
 aRequest.response = function(oXhr){
 oXhr.respond(iErrCode, {"Content-Type": "text/plain; charset=utf-8"}, sMessage);
 };
};

// simulate metadata errors
if (oOptions.metadataError || oUriParameters.get("metadataError")) {
 aRequests.forEach(function (aEntry) {
 if (aEntry.path.toString().indexOf("$metadata") > -1) {
 fnResponse(500, "metadata Error", aEntry);
 }
 });
}

// simulate request errors
var sErrorParam = oOptions.errorType || oUriParameters.get("errorType"),
 iErrorCode = sErrorParam === "badRequest" ? 400 : 500;
if (sErrorParam) {
 aRequests.forEach(function (aEntry) {
 fnResponse(iErrorCode, sErrorParam, aEntry);
 });
}

// custom mock behaviour may be added here

// set requests and start the server
oMockServer.setRequests(aRequests);

```

Figure 209: Understanding the MockServer - Complex Example I

Using the MockServer it is also possible to implement own request/response cycles. In the above code snippet you can see how to implement, for example, an error when during mock no metadata-file was found.

### Start the Application in Simulation Mode



```

<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>nn</title>
 <script id="sap-ui-bootstrap"
 src="../../resources/sap-ui-core.js"
 data-sap-ui-theme="sap_fiori_3"
 data-sap-ui-resourceroots='{
 "com.sap.training.sampleproject": "./"
 }'
 data-sap-ui-oninit="module:com.sap/training/sampleproject/test/initMockServer"
 data-sap-ui-compatVersion="edge"
 data-sap-ui-async="true"
 data-sap-ui-preload="async"
 data-sap-ui-frameOptions="trusted"
 ></script>
 </head>
 <body class="sapUiBody">
 <div data-sap-ui-component
 data-name="com.sap.training.sampleproject"
 data-id="container"
 data-settings='{"id": "sampleproject"}'
 ></div>
 </body>
</html>

```

Figure 210: Start the Application in Simulation Mode

To start the application in simulation mode, it is recommended to add a specific start HTML file to the project. A common practice put such a starter file into a folder called test. The above slide shows an implementation of such a starter file.

### The Mock Server in SAP Business Application Studio

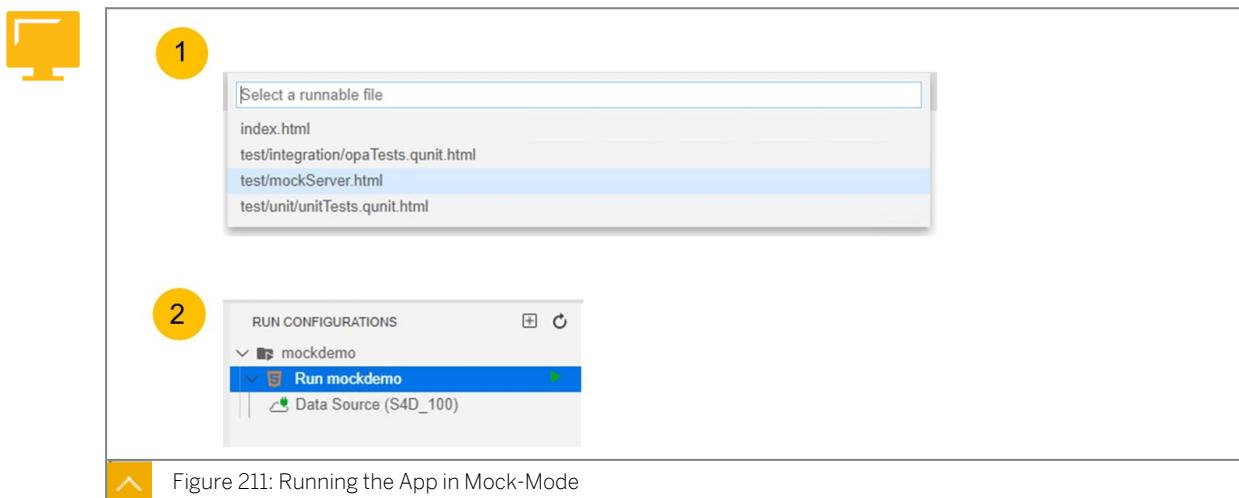
The SAP Business Application Studio offers a build in MockServer.

The developer has to configure:

- The URI to the OData service
- The path to the local *metadata.xml* file

The SAP Business Application Studio offers an editor to generate or modify existing mock data. To access the editor select the local *metadata.xml* file and choose from the context menu the menu item *Edit Mock Data*. With the editor it is possible to generate data and store them in JSON-files inside the SAPUI5 project.

After leaving the mock data editor with OK new json files are stored inside the project.



To start the application make sure that you provide local json-files for simulation. Switch than into the Run Configuration view and create a new run configuration and choose the file *test/mockServer.html*. In the second step bind, if available, the data source to your backend system and start the run configuration by click the green arrowhead. The binding to a datasource is necessary, this is not necessary because the data were loaded from the backend, rather without the binding the application will not start.

After the start a MessageToast appears to inform the developer that the simulation has started.



### LESSON SUMMARY

You should now be able to:

- Work with the Mock Server

# Working with the OData Model



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with the OData Model

## The OData Model

### Write Support = CRUD

SAPUI5 supports the following operations for the ODataModel:



- create
- read
- update
- delete (implemented as a remove method)

Recall, that for the ODataModel, the default binding mode is One-way but can be set to Two-way to support write operations.

OData uses HTTP ETags for optimistic concurrency control. The service must be configured to provide them. The ETag can be passed within the parameters map for every CRUD request. If no ETag is passed, the ETag of the cached entity is used, if it is loaded already.

To address cross-site request forgery, an OData service may require XSRF tokens for change requests by the client application. In this case, the client has to fetch a token from the server and send it with each change request to the server. The OData model fetches the XSRF token when reading the metadata and then automatically sends it with each write request header. If the token is no longer valid, a new token can be fetched by calling the refreshSecurityToken() function on the OData model. The token is fetched with a request to the service document. To ensure getting a valid token, make sure that the service document is not cached.

### XSRF Tokens

The following are properties of XSRF Tokens:



- XSRF tokens address the challenge of cross site request forgery.
- Required for write requests (create, update, delete).
- The client has to fetch the XSRF token from the server and then send it with each write request.
- The ODataModel fetches the XSRF token when reading the metadata and automatically sends it in each write request header.

- ODataModel has a refresh method that can be called to obtain a new XSRF token if the prior one becomes invalid.

Had a problem getting the boolean argument on the constructor for ODataModel to work to fetch the XSRF token.

The examples in the exercises fetched the XSRF token using a custom request header. Should be working in later versions.

### Performing Create

- Create triggers a HTTP POST operation against the OData service.
- The application specifies which collection of the service is to be used.
- The data payload must also be specified.
- Deep creates are not supported.
- Syntax:
- create (sPath, oData, mParameters?) : object.
  - sPath – the OData collection that where the entry should be created.
  - oData – the data of the entry that should be created.
  - mParameters – includes the success and error callback functions.



### The API Documentation



- Below is the API documentation for create which appears to require three parameters:

#### create

Trigger a POST request to the OData service that was specified in the model constructor.

Please note that deep creates are not supported and may not work.

**Visibility:** public

1    create (sPath, oData, mParameters?, ) : object



Figure 212: About the API Documentation

**sPath:** A string containing the path to the collection where an entry should be created. The path is concatenated to the service URL which was specified in the model constructor.

**oData:** Data of the entry that should be created.

**mParameters:** Optional parameter map containing any of the following properties:

**context:** If specified , sPath has to be relative to the path given with the context.

**success:** A callback function which is called when the data has been successfully retrieved. The handler can have the following parameters: oData and response. The oData parameter contains the data of the newly created entry if it is provided by the backend.

The responseparameter contains information about the response of the request.

**error:** A callback function which is called when the request failed. The handler can have the parameter oError which contains additional error information.

**urlParameters:** A map containing the parameters that will be passed as query strings.

**headers:** A map of headers for this request.

**batchGroupId:** Deprecated - use groupId instead.

**groupId:** ID of a request group; requests belonging to the same group will be bundled in one batch request.

**changeSetId:** ID of the ChangeSet that this request should belong to.

**refreshAfterChange:** Defines whether to update all bindings after submitting this change operation. See setRefreshAfterChange If given, this overrules the model-wide refreshAfterChange flag for this operation only.



- Based on the API documentation the create function get three parameters

```

onSave : function(oEvent) {
 var oData = this.getView().getModel("newBP").getData();

 this._create("/BusinessPartnerSet", oData)
 .then(function(oData) {
 MessageBox.show(
 "New BusinessPartner with ID " + oData.BusinessPartnerID + " created",
 {
 icon: MessageBox.Icon.SUCCESS,
 title: "Info",
 onClose: function () {/*handle*/}});
 }.bind(this))
 .catch(function(oError) {
 this._handleErrorMessage(oError);
 }.bind(this));
 },

_create : function(sPath, oObject) {
 var oDataModel = this.getOwnerComponent().getModel();
 var oPromise = new Promise(function(resolve, reject) {
 var mParameters = {
 success: function(oData) {
 resolve(oData);
 },
 error: function(oError) {
 reject(oError.statusCode);
 }
 };
 oDataModel.create(sPath, oObject, mParameters);
 });
 return oPromise;
},

```



Figure 213: About the API Documentation - Continued

The slide shows a sample implementation of the usage of the create-function. As you can see a new entry is added to the EntitySet BusinessPartnerSet. The data for the new entry where fetch from a JSONModel (not shown in coding). In the OData-Service the create method of the specified entity will be called. As you can see the implementation is using a Promise.

As you can see in the success handler the data for the newly created entity are return from the backend system and the newly created Id can be accessed. The next figure shows you how to handle errors.

## Handle Server Response After Create



```
_handleErrorMessage : function(oError) {
 let oErr = JSON.parse(oError.responseText);
 MessageBox.show(
 oErr.error.message.value,
 MessageBox.Icon.ERROR,
 "Error occurred"
);
},
```

Figure 214: Handle Server Response After Create

To handle the response of the backend you can implement two event handlers.

As you can see in the success-event handler you will be able to access the newly created entity.

In the event handler for the error event you can access the error of the backend through the given object.

## Creating Entities



- To create Entities for a specified Entity Set, call the `createEntry()` method.
  - The method returns a `Context` object that points to the newly created Entity. The application can bind against these objects and change the data by means of two-way binding.

```
manifest.json x
webapp > manifest.json > {} sap.ui5 > ...
59 <-
60 "" : {
61 "dataSource": "mainService",
62 "preload": true,
63 "settings": {
64 "operationMode": "Server",
65 "defaultBindingMode": "TwoWay"
66 }
67 },
68 ...
69

createEntry : function() {
 let oModel = this.getOwnerComponent().getModel(),
 oContext = oModel.createEntry("/BusinessPartnerSet", {properties: {
 BusinessPartnerRole : "02",
 Address : {
 AddressType : "02"
 }
 }}),
 oSimpleForm = this.getView().byId("idBusinessPartnerForm");
 oSimpleForm.setBindingContext(oContext);
}.
```

Figure 215: Creating Entities

The application can choose the properties that shall be included in the created object and can pass its own default values for these properties. Per default, all property values are empty, that is, undefined. The Entity Set and the passed properties must exist in the metadata definition of the OData service.

## Performing Read



- The read method triggers a HTTP GET request to the OData service specified in the model constructor
- Data are stored inside the model and Data is returned with the response
- Syntax: `read(sPath, mParameters?) : object`

```

readData : function(sObjectId) {
 let oModel = this.getOwnerComponent().getModel(),
 sPath = this.getModel().createKey("BusinessPartnerSet", {
 BusinessPartnerID : sObjectId
 });

 return new Promise(function(resolve, reject) {
 var mParameters = {
 success: function(oData) {
 resolve(oData);
 },
 error: function(oError) {
 reject(oError);
 }
 };
 oModel.read(sPath,mParameters);
 });
},

```

Figure 216: OData Model - Sample Code Binding ODataModel to a Table

The following parameters can be passed as an argument to the read-function:

- **sPath**: A string containing the path to the data which should be retrieved. The path is concatenated to the service URL which was specified in the model constructor
- **mParameters**: Optional parameter map containing any of the following properties:
  - **context**: If specified, sPath has to be relative to the path given with the context.
  - **urlParameters**: A map containing the parameters that will be passed as query strings.
  - **filter**: An array of filters of the type `sap.ui.model.Filter` to be included in the request URL.
  - **sorter**: An array of sorters of type `sap.ui.model.Sorter` to be included in the request URL.
  - **success**: A callback function which is called when the data has been successfully retrieved. The handler can have the following parameters: oData and response.

The oData parameter contains the data of the retrieved data. The response parameter contains further information about the response of the request:

- **error**: A callback function which is called when the request failed. The handler can have the parameter: oError which contains additional error information.
- **batchGroupId**: Deprecated - use groupId instead.
- **groupId**: ID of a request group; requests belonging to the same group will be bundled in one batch request.

## Performing Update



- Update triggers a HTTP PUT operation against the OData service

```
updateData: function(oObject) {
 let oModel = this.getOwnerComponent().getModel(),
 sPath = oModel.createKey("BusinessPartnerSet", {
 BusinessPartnerID : oObject.BusinessPartnerID
 });
 return new Promise(function(resolve, reject) {
 let mParameters = {
 success: function(oData) {
 resolve(oData);
 },
 error: function(oError) {
 reject(oError);
 }
 },
 customerHeader = {"Content-Type": "application/json"};
 oModel.setHeaders(customerHeader);

 oModel.update(sPath, oObject, mParameters);
 });
},
```



Figure 217: Performing Update

The `update()` function triggers a PUT/MERGE request to the OData service that was specified in the model constructor. The update method used is defined by the global `defaultUpdateMethod` parameter which is `sap.ui.model.odata.UpdateMethod.Merge` by default.

To send update requests in a PUT request, you have to set the parameter `defaultUpdateMethod` to `sap.ui.model.odata.UpdateMethod.Put` when instantiating the OData model.

## Performing Delete



- Remove triggers a HTTP DELETE operation against the OData service
- The application specifies which collection of the service is to be used
- The parameters object can also contain the eTag
- Syntax:
  - Remove(sPath, mParameters?) : object
  - sPath – the OData collection that where the entry should be created
  - mParameters – includes the success and error callback functions

```

deleteData : function(sObjectId) {
 let oModel = this.getOwnerComponent().getModel(),
 sPath = oModel.createKey("BusinessPartnerSet", {
 BusinessPartnerID : sObjectId
 });
 return new Promise(function(resolve, reject) {
 let mParameters = {
 success: function(oData) {
 resolve(oData);
 },
 error: function(oError) {
 reject(oError);
 }
 };
 oModel.remove(sPath, mParameters);
 });
},

```

Figure 218: Performing Delete

To delete an entity from the backend system you can call the remove function on the *ODataModel-object*. This call will trigger an HTTP DELETE.

## Refreshing the Model

### Basics about refreshing the model:

- The model provides a mechanism to automatically refresh bindings that depend on changed Entities. If you carry out a create, update or delete function, the model identifies the bindings and triggers a refresh for these bindings.
- You can disable the auto refresh by calling `oModel.setRefreshAfterChange(false);`. This disables automatic updates of all bindings after change operations.
- If the auto refresh is disabled, the application has to take care of refreshing the respective bindings.
- The `refresh()` function refreshes all data within an OData model. Each binding reloads its data from the server. Data that has been imported via manual CRUD requests is not reloaded automatically.

## Considerations for SAP Gateway Services



eTags can be used for concurrency control if the OData service is configured to provide them

```

deleteData : function(sObjectId) {
 let oModel = this.getOwnerComponent().getModel(),
 sPath = oModel.createKey("BusinessPartnerSet", {
 | BusinessPartnerID : sObjectId
 });
 return new Promise(function(resolve, reject) {
 let mParameters = {
 success: function(oData) {
 | resolve(oData);
 },
 error: function(oError) {
 | reject(oError);
 }
 },
 customerHeader = {
 "X-Requested-With" : "XMLHttpRequest",
 "X-CSRF-Token" : "Fetch",
 "DataServiceVersion": "2.0",
 "IF-Match" : "*"
 };
 oModel.setHeaders(customerHeader);
 oModel.remove(sPath, mParameters);
 });
},

```



Figure 219: Considerations for SAP Gateway Services

An ETag (entity tag) is an HTTP response header returned by an HTTP/1.1 compliant web server used to determine change in content of a resource at a given URL.

Use of the eTag:

- If an eTag is specified as a parameter, it will be used in the If-Match header.
- If no eTag is specified, it will be retrieved from the entry's metadata.

## Two-way Binding



### ▪ To enable two-way-binding in the manifest.json.

```

manifest.json x
webapp > manifest.json > sap.ui5 > models > ...
59 "": {
60 "dataSource": "mainService",
61 "preload": true,
62 "settings": {
63 "operationMode": "Server",
64 "defaultCountMode": "Inline",
65 "defaultBindingMode": "TwoWay"
66 }
67 },
68 }

```



Figure 220: Two-way Binding |

Filtering and sorting is not possible if two-way changes are present as this would cause inconsistent data on the UI. Therefore, before you carry out sorting or filtering, you have to submit or reset the changes.



- Data changes are made on a data copy. To submit the changes, use `submitChanges()`.
- Use `hasPendingChanges()` to check if there exist pending changes in the model.

```
saveChanges : function() {
 let oModel = this.getOwnerComponent().getModel();
 if(oModel.hasPendingChanges()) {
 oModel.submitChanges({
 success: function(oData) {
 ...
 },
 error:function(oError) {
 ...
 }
 });
 }
},
```

- With `resetChanges()` you can reset all changes.

```
let oModel = this.getOwnerComponent().getModel();
oModel.resetChanges()
```



Figure 221: Two-way Binding II

You can also reset only specific Entities by calling `resetChanges()` with an array of entity paths. Two-way-binding can also be configured inside the `manifest.json` file.

### Using Function Imports

You can implement such additional service operations in the Service Builder by creating function imports within your data model. For example, you could create function imports for the following custom operations: Confirm Work Item.

Check Flight Availability.

While it is simple to create new function imports to invoke custom operations, if the operation you want to use can be invoked using a standard CRUD operation, you should not create a function import. That is, you should only create function imports for custom operations that cannot be invoked using a standard operation.

## Using Function Imports



- The SAP Gateway can provide function imports
- Function imports are called using the callFunction method of the OData-Model

```
_confirm : function(oEvent) {
 var Carrid = oEvent.getSource().data("Carrid");
 var Bookid = oEvent.getSource().data("Bookid");
 var oDataModel = this.getOwnerComponent().getModel();
 var oPromise = new Promise(function(resolve, reject) {
 var mParameters = {
 success: resolve,
 error: function(oError) {
 reject(oError.responseText);
 },
 method: "POST",
 urlParameters: {
 Carrid : Carrid,
 Bookid : Bookid
 }
 };
 oDataModel.callFunction("/ConfirmBooking", mParameters);
 });

 return oPromise;
},
```



Figure 222: Using Function Imports

To invoke a function import the ODataModel provides the method `callFunction`. The method invocation triggers a request to the function import OData service that was specified in the model constructor.

If the return type of the function import is either an entity type or a collection of an entity type, then the changes are reflected in the model. Otherwise they are ignored, and the response can be processed in the success callback.



## LESSON SUMMARY

You should now be able to:

- Work with the OData Model

# Describing OData Deep Inserts



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe OData Deep Inserts

## OData Deep Inserts

### OData Deep Inserts

Some information about OData Deep Inserts:



- Complex business data is often hierarchical. Sometimes data can only be created or updated with the full hierarchy.
- For example Sales Orders and Sales Order Line Items. They are often created in the backend system at the same time in a single transaction, not created independently.
- OData and SAPUI5 support the creation of this hierarchical data, known as Deep Insert.
- SAP NW Gateway systems support Deep Insert via the method `CREATE_DEEP_ENTITY`.

To provide a deep insert it is necessary to implement a method on the SAP NetWeaver Gateway with the name `CREATE_DEEP_ENTITY`.

### OData Deep Inserts - OData Structure



```
{
 "ID" : "500000001",
 "GrossAmount" : "886.55",
 "Items" : [
 {
 "ProductId" : "HT-1030",
 "SoItemPos" : "10",
 "SoId" : "500000001",
 "GrossAmount" : "547.40",
 },
 {
 "ProductId" : "HT-1031",
 "SoItemPos" : "20",
 "SoId" : "500000001",
 "GrossAmount" : "339.15",
 }
]
}
```



Figure 223: OData Deep Inserts - OData Structure, Example

The figure shows a code example for a deep insert.

### The following are basics about the Deep inserts:



- A Deep Insert is basically the opposite of a OData Query that uses \$expand to retrieve an entity and related associations in a single call.
- Sample GET call response to a SalesOrdercollection expanding Items that contains the line item data: see figure above.
- Notice ID and GrossAmount at the header level.
- “Items” contains an array of line item objects.

### OData Deep Inserts - OData Structure, Further Details

#### Further Details:



- A Deep Insert is the same thing in reverse, the nested structure is the body of the HTTP POST.
- Sample POST call body to a SalesOrder collection including **Items** that contains the line item data: see figure above.
- Structure is very similar to the response from the previous GET call.
- **Items** contains an array of line item objects.

### OData Deep Inserts - SAPUI5



#### ▪ The SAPUI5 OData Model object supports deep inserts

```
class sap.ui.model.odata.v2.ODataModel
```

Overview   Constructor   Events ▾   Methods ▾

**create**

Trigger a POST request to the OData service that was specified in the model constructor.  
Please note that deep creates are not supported and may not work.  
Visibility: public

```
1 create(sPath, oData, mParameters) : object
```

Param	Type	Description
sPath	string	A string containing the path to the collection where an entry should be created. The path is concatenated to the service URL which was specified in the model constructor.
oData	object	Data of the entry that should be created.
mParameters	map	Optional parameter map containing any of the following properties:

Figure 224: OData Deep Inserts - SAPUI5 1/2

The create method automatically handles deep inserts if the object structure passed in hierarchical. The backend OData producer must also support the deep insert.



#### Note:

Please note that some older browsers may not support the deep insert.



- The parameter map of the create function consists of various aspects to configure the data handling

class sap.ui.model.odata.v2.ODataModel		
Overview	Constructor	Events
Methods		
<b>mParameters</b>	map	Optional parameter map containing any of the following properties
context	object	If specified, sPath has to be relative to the path given with the context.
success	function	A callback function which is called when the data has been successfully retrieved. The handler can have the following parameters: oData and response. The oData parameter contains the data of the newly created entry if it is provided by the backend. The response parameter contains information about the response of the request.
error	function	A callback function which is called when the request failed. The handler can have the parameter oError which contains additional error information.
urlParameters	map	A map containing the parameters that will be passed as query strings
headers	map	A map of headers for this request
batchGroupId	string	Deprecated - use groupId instead
groupId	string	ID of a request group; requests belonging to the same group will be bundled in one batch request
changeSetId	string	ID of the ChangeSet that this request should belong to
refreshAfterChange	string	Defines whether to update all bindings after submitting this change operation. See setRefreshAfterChange If given, this overrules the model-wide refreshAfterChange flag for this operation only.

Figure 225: OData Deep Inserts - SAPUI5 2/2

**mParameters** Optional parameter map containing any of the following properties:

- context**: If specified, sPath has to be relative to the path given with the context.
- success**: A callback function which is called when the data has been successfully retrieved. The handler can have the following parameters: oData and response. The oData parameter contains the data of the newly created entry if it is provided by the backend. The response parameter contains information about the response of the request.
- error**: A callback function which is called when the request failed. The handler can have the parameter oError which contains additional error information.
- urlParameters**: A map containing the parameters that will be passed as query strings.
- Headers**: A map of headers for this request.
- batchGroupId**: Deprecated - use groupId instead.
- groupId ID of a request group**: requests belonging to the same group will be bundled in one batch request.
- changeSetId**: ID of the ChangeSet that this request should belong to.
- refreshAfterChange**: Defines whether to update all bindings after submitting this change operation. See setRefreshAfterChange If given, this overrules the model-wide refreshAfterChange flag for this operation only.



## LESSON SUMMARY

You should now be able to:

- Describe OData Deep Inserts



# Introducing SAPUI5 Smart Controls



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand SAPUI5 Smart Controls

### SmartField Controls

The following figure provides an overview of the SmartField control.



- Provides an efficient way to add input-enabled fields.
- A SmartField control automatically adjusts to the metadata of the underlying OData service, for example, by doing the following:
  - Hosting controls for editing and displaying the values of OData properties
  - Providing value help automatically
  - Performing input checks
- A SmartField control implements field control and supports message handling.



Figure 226: SmartField Control Overview



- As a standalone control, for example, in XML views
- In combination with a SmartForm control
- In combination with a SmartTable control
- Used in particular as cell editor in editing scenarios



Figure 227: SmartField Control Use Cases



- The SmartField control selects a control for displaying and a control for editing the OData property to which it is bound.
- Complex properties are supported, which are arbitrarily and deeply nested.
- The SmartField control allows for binding of navigation properties.
- The entity set to which the bindings are related is either specified in the `entitySet` attribute of the control or derived from the binding context at runtime.

Figure 228: SmartField Control Details

The main criterion for selecting nested controls is the EDM type of the OData property to which a SmartField control is bound.

When binding the *SmartField* control against the OData service property of type `Edm.Boolean`, and if the *SmartField* control is in read-only mode, static texts are used for visual representation. In addition, a configuration parameter in the *SmartField* control can define the properties of the static texts of the *CheckBox*, such as Yes/No or True/False. For the *SmartForm* control, the custom data can be used for this purpose.

The following figure shows information about configuring a SmartField control.



- The configuration aggregation of the SmartField control provides the option to overwrite the default behavior of the SmartField control.
- Using the `controlType` property, you can select the appropriate control for your use case from the following control types that are available:
  - Check box
  - Date picker
  - Drop-down list (combo box)
  - Input
  - Select drop-down list (`sap.m.Select`)

Figure 229: SmartField Control Configuration

The control types supported depends on the related data types, for example:

- If the relevant OData property is of type `Edm.String`, the SmartField control can be configured to render a combo box or a select drop-down list.
- If the relevant OData property is of type `Edm.Boolean`, the SmartField control can be configured to render a combo box.
- If the relevant OData property is of type `Edm.DateTime`, the SmartField control can be configured to render a date picker.

The following figure shows the available controls for a SmartField control.



Editing Use Cases		Display Use Cases	
EDM Type	Control	EDM Type	Control
Edm.Boolean	sap.m.CheckBox	Edm.Boolean	sap.m.CheckBox
Edm.Int16	sap.m.Input	Edm.Int16	sap.m.Text
Edm.Int32		Edm.Int32	
Edm.Int64		Edm.Int64	
Edm.SByte		Edm.SByte	
Edm.Byte		Edm.Byte	
Edm.Single		Edm.Single	
Edm.Float		Edm.Float	
Edm.Double		Edm.Double	
Edm.Decimal		Edm.Decimal	
Edm.String		Edm.String	
Edm.DateTime	sap.m.DateTimePicker	Edm.DateTime	
Edm.DateTimeOffset		Edm.DateTimeOffset	

Figure 230: Available Controls

The following figure shows the field control aspects of a SmartField control.



- The field control handles the visual representation of SmartField controls.
- The following attributes are available to implement field control:
  - Enabled  
Toggles from display to edit mode
  - Visible  
Hides the SmartField control
  - Mandatory  
Determines whether input is required

Figure 231: Field Control

The field control handles the visual representation of SmartField controls, such as:

- Whether input is mandatory
- Whether the controls are read-only
- Whether the controls are hidden as defined by the SAP Fiori user interface programming model

The following attributes are available to implement the field control:

- Enabled
- Toggles from display to edit mode
- Visible hides the SmartField control
- Mandatory determines whether input is required

Consumers of the SmartField control can further adapt the runtime behavior by binding these attributes.

The following figure shows some additional properties of the SmartField control.



- The `displayBehaviour` property, defines how an ID and a description or Boolean values are represented in read-only mode.
- The `preventInitialDataFetchInValueHelpDialog` property, enables the prevention of the query being fired immediately when the value help dialog is opened.



Figure 232: Further Properties

Using the `displayBehaviour` property, you can define how an ID and a description or Boolean values are represented in read-only mode.

You have the following options:

- `sap.ui.comp.smartfield.DisplayBehaviour.descriptionAndId`  
Shows the description and ID of the available values.
- `sap.ui.comp.smartfield.DisplayBehaviour.descriptionOnly`  
Shows only the description of the available values.
- `sap.ui.comp.smartfield.DisplayBehaviour.idAndDescription`  
Shows the ID and description for the available values.
- `sap.ui.comp.smartfield.DisplayBehaviour.idOnly`  
Shows the ID only.
- `sap.ui.comp.smartfield.DisplayBehaviour.OnOff`  
Shows the Boolean value as On/Off.
- `sap.ui.comp.smartfield.DisplayBehaviour.TrueFalse`  
Shows the Boolean value as True/False.
- `sap.ui.comp.smartfield.DisplayBehaviour.YesNo`  
Shows the Boolean value as Yes/No.

The following figure shows how a SmartField control is used in an XML view.



```
<mvc:View
 xmlns:mvc="sap.ui.core.mvc"
 controllerName="sap.ui.demo.smartControls.SmartField"
 xmlns:form="sap.ui.layout.form"
 xmlns:smartField="sap.ui.comp.smartfield">
 <form:SimpleForm
 minWidth="1024"
 maxContainerCols="2"
 editable="true"
 layout="ResponsiveGridLayout"
 labelSpanL="3"
 labelSpanM="3"
 emptySpanL="4"
 emptySpanM="4"
 columnsL="1"
 columnsM="1"
 class="editableForm">
 <form:content>
 <smartField:SmartLabel labelFor="idPrice"/>
 <smartField:SmartField value="{Price}" id="idPrice"/>
 </form:content>
 </form:SimpleForm>
</mvc:View>
```



Figure 233: SmartField Control Usage in an XML View



## LESSON SUMMARY

You should now be able to:

- Understand SAPUI5 Smart Controls



# Working with SAPUI5 Smart Controls



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use SAPUI5 Smart Controls

### SmartForm Controls

Controls in the `sap.ui.comp` library (smart controls) focus strongly on SAP Fiori elements.

We will continue to maintain the library and all of its controls in the future. However, there will be no development of new features unless specifically requested by SAP Fiori elements.



- The SmartForm control displays form content.
- If used in combination with the SmartField control and OData metadata annotations along with additional configuration, the control allows you to create a form with minimal effort.
- Depending on user authorizations, the form enables users to for example:
  - Switch from display to edit mode
  - Add and group fields
  - Rename field labels
  - Implement a user input check



Figure 234: SmartForm Control Overview

The following figure describes the features of the *SmartForm* control.



The SmartForm control supports the following features:

- Adaptation settings  
A key user can adapt the form for all users in one client.
- Display/Edit button  
This optional button allows the user to toggle from display to edit mode.
- Field labels  
For fields of type SmartField, the SmartForm control automatically creates a label based on the OData metadata annotations.
- Check button  
This optional button allows the user to check the current user input.



Figure 235: SmartForm Control Features

The SmartForm control supports the following features:

- Adaptation settings. A key user can adapt the form for all users in one client by doing the following:
  - Adding and hiding fields
  - Adding and hiding groups
  - Changing the order of fields and groups
  - Renaming field labels
- Display/Edit button. This optional button allows the user to toggle from display to edit mode.



Note:

Fields of type *SmartField* are automatically displayed with the appropriate control in the required mode, for example, texts on the user interface in display mode and user input in edit mode. If controls other than the *SmartField* control are used, the application in question must handle the switch between display and edit mode.

- Field labels. For fields of type *SmartField*, the *SmartForm* control automatically creates a label based on the OData metadata annotations.
- Check button. This optional button allows the user to check the current user input.



Note:

For fields of type *SmartField*, values are checked based on the OData metadata annotations. Depending on which theme is defined, the fields with errors are circled in red. When the user clicks on one of these fields, the relevant error message is displayed.

The following figure provides some details about the *SmartForm* control.



- A *SmartForm* control consists of:
  - groups (`sap.ui.comp.smartform.Group`)
  - group elements (`sap.ui.comp.smartform.GroupElement`)
- A group element is a collection of controls that are displayed along with a label. Typically, a group element consists of exactly one control and the respective label.
- The *SmartForm* control aggregates groups, and a group aggregates group elements. The group elements themselves aggregate elements of type `sap.ui.core.Control`.



Figure 236: SmartForm Control Details

The following figure describes the *SmartForm* control layout.



- The SmartForm control uses a ResponsiveGridLayout that can be adjusted.
- The following properties are exposed in the aggregation layout:
  - labelSpanXL, labelSpanL, labelSpanM, labelSpanS
  - emptySpanXL, emptySpanL emptySpanM, emptySpanS
  - columnsXL, columnsL, columnsM
  - breakpointXL, breakpointL, breakpointM
  - gridDataSpan



Figure 237: SmartForm Control Layout

To display fields next to each other with a label on top, you can use the `gridDataSpan` property of the layout element in combination with the `useHorizontalLayout` property.

The form is embedded in an `sap.m.Panel` if the `expandable` property is set. Using this property, the form can be collapsed and expanded.

The following figure describes the `SmartForm` control toolbar.



- **The SmartForm control uses a toolbar for displaying the title of the form and the following buttons (if configured):**
  - **Display/Edit** (`editToggable` property)
  - **Check** (`checkButton` property)
- **Alternatively, the custom toolbar can be used (customToolbar aggregation).**  
The SmartForm control then replaces the standard toolbar with the custom toolbar and adds the title and the buttons if requested.



Figure 238: SmartForm Control Toolbar

The following two figures show how to use a SmartForm control in an XML view.



```
<smartForm:SmartForm id="MainForm" title="General Data"
 entityType="Header, Tax" editToggable="true" expandable="true"
 expanded="true" ignoredFields="AccountingDocumentCategory"
 checkButton="true">
 <smartForm:customData>
 <core:CustomData key="suppressUnit" value="false" />
 <core:CustomData key="dateFormatSettings" value='\{"style":"short"\}' />
 <core:CustomData key="defaultDropDownDisplayBehaviour" value='descriptionAndId' />
 </smartForm:customData>
 <smartForm:customToolbar>
 <Toolbar height="3rem">
 <Text text="Custom Toolbar with a header text" />
 <ToolbarSpacer />
 <Button icon="sap-icon://settings" />
 <Button icon="sap-icon://drop-down-list" />
 </Toolbar>
 </smartForm:customToolbar>
</smartForm:SmartForm>
```



Figure 239: SmartForm Control Usage in an XML View



```
<smartForm:Group label="Dates" id="Dates">
 <smartForm:layout>
 <layout:GridData span="L3 M3 S3" />
 </smartForm:layout>
 <smartForm:GroupElement id="Dates.DocumentDate">
 <smartField:SmartField value="{DocumentDate}" />
 </smartForm:GroupElement>
 <smartForm:GroupElement id="Dates.PostingDate">
 <smartField:SmartField value="{PostingDate}" />
 </smartForm:GroupElement>
</smartForm:Group>
```

Figure 240: SmartForm Control Usage in an XML View

## SmartTable Controls

Controls in the `sap.ui.comp` library (smart controls) focus strongly on SAP Fiori elements.

SAP will continue to maintain the library and all of its controls in the future. However, there will be no development of new features unless specifically requested by SAP Fiori elements.

The following figure provides an overview of the *SmartTable* control.



- The `sap.ui.comp.smarttable.SmartTable` is a wrapper control around any SAPUI5 table.
- The control analyzes the `$metadata` document of an OData service and renders a table for a specific entitySet.
- The control allows the consuming application to build list patterns in an efficient and consistent way and therefore makes it easy for the user to create tables without much effort.

Figure 241: SmartTable Control Overview

The consuming application can overwrite the OData default information.

The *SmartTable* control offers you additional built-in features, such as a row count and the ability to export to a spreadsheet application.

The following figure provides details of the *SmartTable* control.



- When using *SmartTable* with an internal responsive table, you can set the `demandPopin` property to true. This property renders columns that exceed the space available on the screen by displaying popins.
- *SmartTable* checks the custom data section for the columns and reads the `columnIndex` attribute to determine when the columns that are defined in the XML view are rendered.
- If you want to show and follow `navigationProperty` fields for `EntityType`, the *SmartTable* control automatically performs a `$expand` operation.

Figure 242: SmartTable Control Details

**Note:**

If you perform `$expand` operations while doing an export to a spreadsheet, the `$expand` parameters are automatically removed (relevant only for a Gateway export type).

The following figure describes the integration of the *SmartTable* control with other controls.



- The *SmartTable* control is closely linked to the following other controls:
  - VariantManagement
  - SmartFilterBar
  - P13nDialog
- The control also supports the popover of the *SmartLink* control.



Figure 243: SmartTable Control Integration with Other Controls

The following figure shows the use of a *SmartTable* control in an XML view.



```
<smartForm:SmartForm id=" MainForm" title="General Data"
 entityType="Header, Tax" editToggable="true" expandable="true"
 expanded="true" ignoredFields="AccountingDocumentCategory"
 checkButton="true">
 <smartForm:customData>
 <core:CustomData key="suppressUnit" value="false" />
 <core:CustomData key="dateFormatSettings" value='\{"style":"short"\}' />
 <core:CustomData key="defaultDropDownDisplayBehaviour" value='descriptionAndId' />
 </smartForm:customData>
 <smartForm:customToolbar>
 <Toolbar height="3rem">
 <Text text="Custom Toolbar with a header text" />
 <ToolbarSpacer />
 <Button icon="sap-icon://settings" />
 <Button icon="sap-icon://drop-down-list" />
 </Toolbar>
 </smartForm:customToolbar>
</smartForm:SmartForm>
```



Figure 244: SmartTable Control Usage in an XML View

**LESSON SUMMARY**

You should now be able to:

- Use SAPUI5 Smart Controls



## Introducing SAP Fiori Elements



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Get an Introduction to SAP Fiori Elements

### SAP Fiori Elements, Introduction

#### Motivation

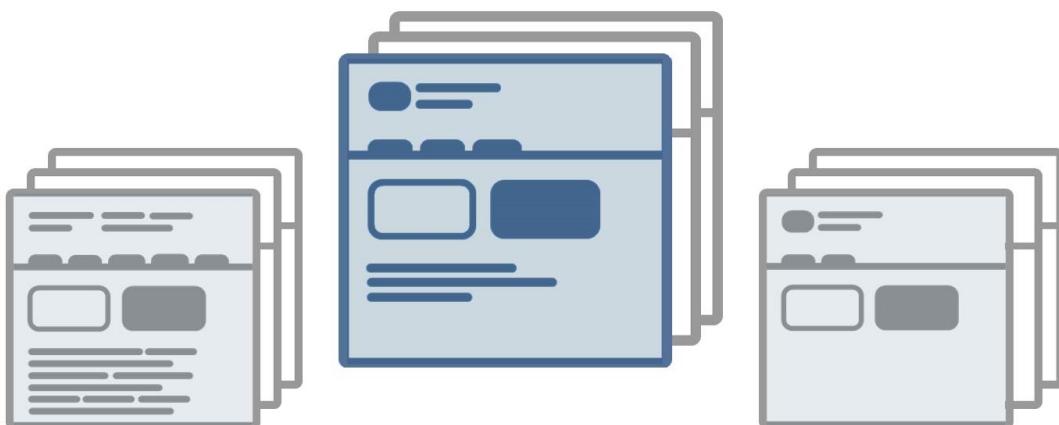
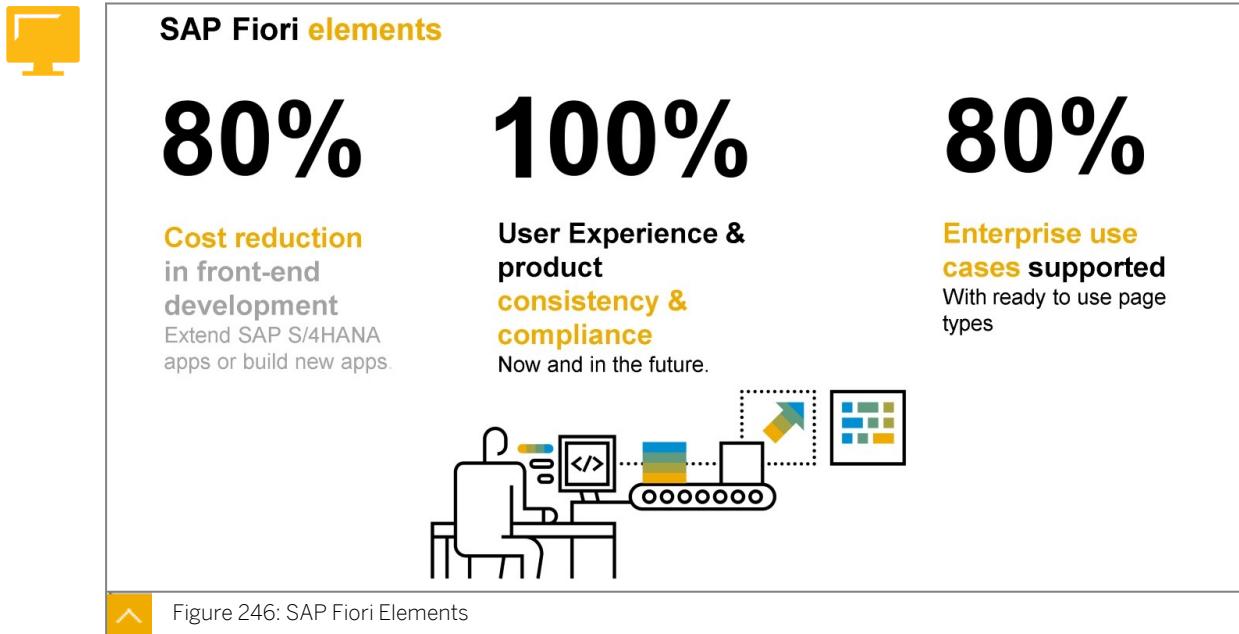


Figure 245: Motivation

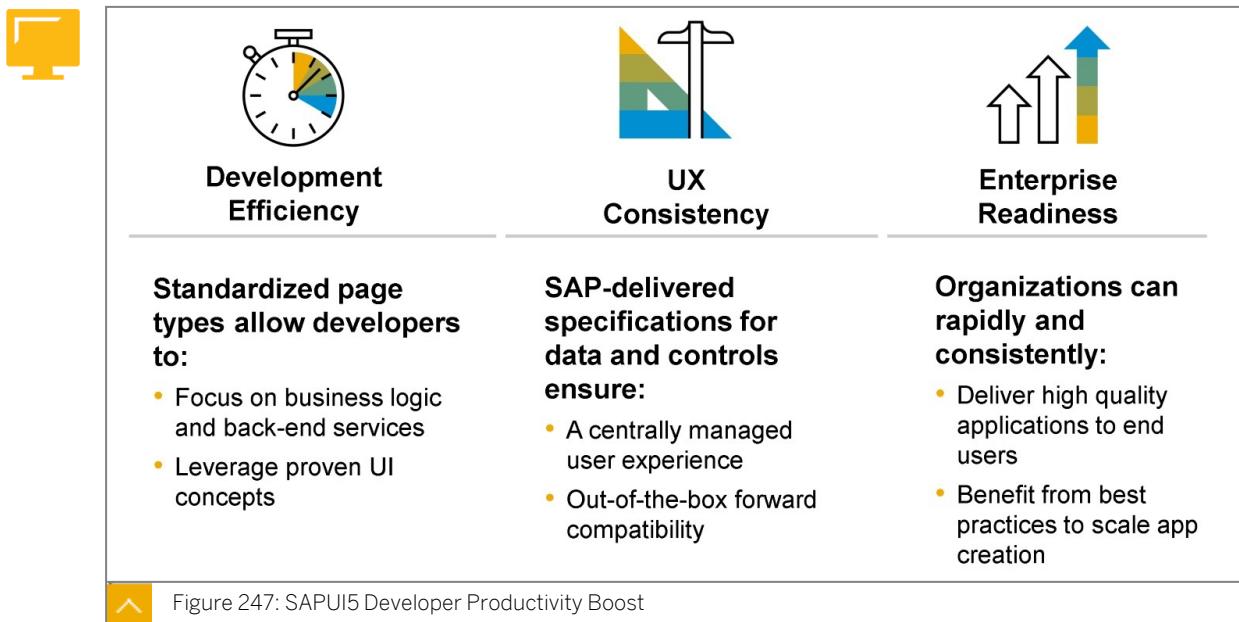
As already mentioned in this course a common goal when implementing SAPUI5 applications to implement UIs with a consistent look and feel over application boundaries.

Reusing the same UI patterns for the same use case is important. SAP Fiori Elements are helping you to achieve this goal.



SAP Fiori elements reduces the effort for front-end development by 80% compared with freestyle SAPUI5 development – and that is already actually quite efficient, since SAPUI5 is a powerful HTML5 / JavaScript development framework.

Fiori elements ensures that updates to the Fiori design are automatically incorporated into apps built with Fiori elements. 80% of typical enterprise use cases are supported by the Fiori elements page types.



Using this set of basic page types ensures that the apps you build are consistent in how they look and behave. Since the user experience is defined in the form of the page layout, navigation, and format of the controls, you can develop apps much more quickly than coding each of these UI elements manually. This means you can scale across your organization by building dozens or even hundreds of SAP Fiori apps that all look similar, even if they are built by different teams.

Like SAPUI5 development, on which SAP Fiori elements is based, you start with an OData service - we will discuss what this is next unit and again later in week 3. With SAP Fiori elements, you generate a standard app and use metadata annotations to modify it to suit your needs.

### Basic Characteristics of Fiori Elements

The following are basic characteristics of Fiori elements:



- SAP Fiori elements provides design for UI patterns and predefined templates for commonly used application patterns.
- Predefined views and controllers ensure UI design consistency across similar apps.
- Apps are based on OData services and annotations.
- No UI coding required.
- SAPUI5 interprets metadata and annotations of the underlying OData service.

### Benefits of Fiori Elements



#### Low-code, standardized UI development with SAPUI5

- Generates the UI at runtime from metadata, resulting in low-code development for standard apps.
- Developers can customize the UI by introducing additional controls and behaviors.
- Provides out-of-the-box UI functionality for:
  - Overview pages
  - List report pages
  - Object pages
  - Worklist pages
  - Analytical list pages
- Supports both desktop and mobile devices.

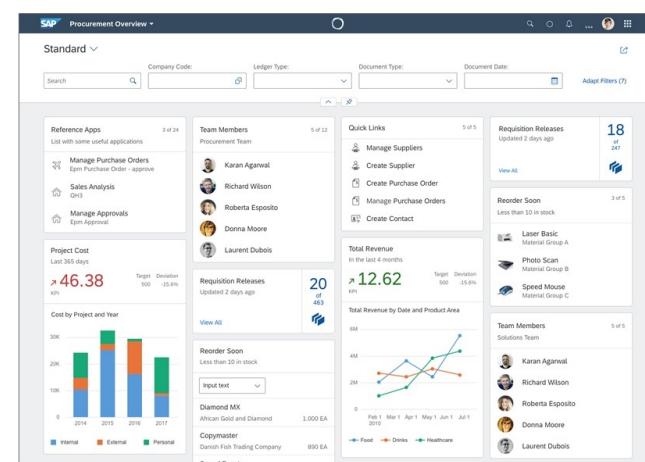


Figure 248: Benefits of Fiori Elements

The figure shows the main benefits of Fiori Elements.

### Reasons for Fiori Elements

The main reasons for Fiori elements: efficiency, consistency, semantic information.

**The main reasons for Fiori Elements are:**



- Efficiency
- Increased (UI) development efficiency
- Metadata-driven UIs enable central infrastructure investments and optimizations

- Consistency
- UX pattern developed just once and not in every single application
- Consistency of applications
- Semantic information
- Bringing semantics to the data leads to a better decoupling of UI and business logic
- Annotations describe semantic information related to data

### **Currently Available SAP Fiori Elements**

The following features are currently available:



- Reuse functionality that does not require specific programming
- A common look and feel and UI behavior for all apps
- **Edit** mode control, switching between display and edit, and submitting changes
- Message handling
- SAP Fiori launchpad integration
- SAP Business Application Studio wizard for app creation
- Control of the UI using OData annotations, which semantically enrich the OData metadata
- Multi-device support
- Status colors and icons to indicate criticality
- Header facets to define which information is displayed in the header
- Value help
- Handling of draft documents (draft saving is available)

Available SAP Fiori element based UIs are:



- List Report
- Object Page
- Overview Pages
- Analytical List Page

#### **List Report and Object Page**

SAP Fiori elements contain predefined templates for list reports and object pages. A list report allows users to view and work with items (objects) organized in list (table) format. The list report is typically used in conjunction with an object page. This object page allows users to work with objects, providing functionality to view, edit, and create objects.

#### **Overview Pages**

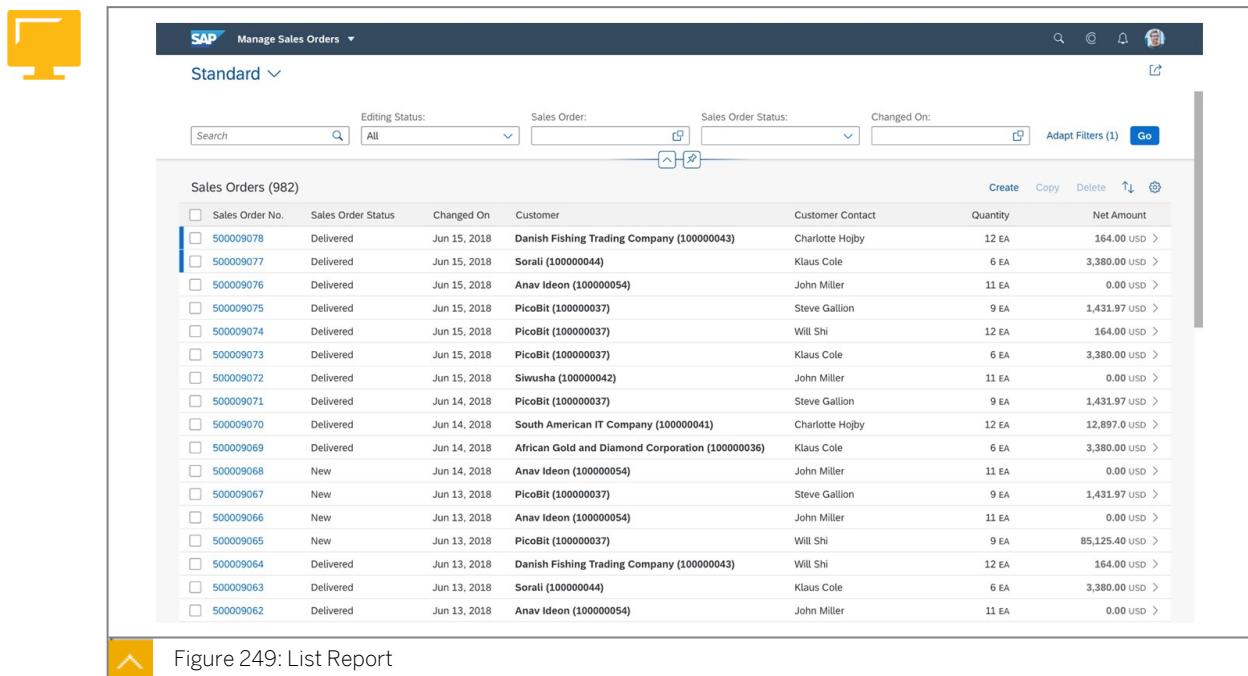
An overview page is a data-driven SAP Fiori app for organizing large amounts of information. Information is visualized in a card format, different cards for different types of content, in an attractive and efficient way. The user-friendly experience makes viewing, filtering, and acting upon data quick and simple. While simultaneously presenting the big picture at a glance,

business users can focus on the most important tasks enabling faster decision making as well as immediate action.

### Analytical List Page

Analytical list page is a SAP Fiori elements application for detailed analytics. It lets you analyze data from different perspectives, to investigate a root cause, and to act on transactional content. You can identify relevant areas within data sets or significant single instances using data visualization and business intelligence. All this can be done seamlessly within one page.

### List Report



The screenshot shows the SAP Fiori List Report application interface. At the top, there is a header bar with the SAP logo and the text "Manage Sales Orders". Below the header, there are several filter fields: "Editing Status:" (set to "All"), "Sales Order:" (empty), "Sales Order Status:" (empty), and "Changed On:" (empty). There are also buttons for "Adapt Filters (1)" and "Go". The main area displays a table titled "Sales Orders (982)". The table has columns: Sales Order No., Sales Order Status, Changed On, Customer, Customer Contact, Quantity, and Net Amount. The data in the table is as follows:

Sales Order No.	Sales Order Status	Changed On	Customer	Customer Contact	Quantity	Net Amount
500009078	Delivered	Jun 15, 2018	Danish Fishing Trading Company (100000043)	Charlotte Hojby	12 EA	164.00 USD >
500009077	Delivered	Jun 15, 2018	Sorali (100000044)	Klaus Cole	6 EA	3,380.00 USD >
500009076	Delivered	Jun 15, 2018	Anav Ideon (100000054)	John Miller	11 EA	0.00 USD >
500009075	Delivered	Jun 15, 2018	PicoBit (100000037)	Steve Gallion	9 EA	1,431.97 USD >
500009074	Delivered	Jun 15, 2018	PicoBit (100000037)	Will Shi	12 EA	164.00 USD >
500009073	Delivered	Jun 15, 2018	PicoBit (100000037)	Klaus Cole	6 EA	3,380.00 USD >
500009072	Delivered	Jun 15, 2018	Siwusha (100000042)	John Miller	11 EA	0.00 USD >
500009071	Delivered	Jun 14, 2018	PicoBit (100000037)	Steve Gallion	9 EA	1,431.97 USD >
500009070	Delivered	Jun 14, 2018	South American IT Company (100000041)	Charlotte Hojby	12 EA	12,897.00 USD >
500009069	Delivered	Jun 14, 2018	African Gold and Diamond Corporation (100000036)	Klaus Cole	6 EA	3,380.00 USD >
500009068	New	Jun 14, 2018	Anav Ideon (100000054)	John Miller	11 EA	0.00 USD >
500009067	New	Jun 13, 2018	PicoBit (100000037)	Steve Gallion	9 EA	1,431.97 USD >
500009066	New	Jun 13, 2018	Anav Ideon (100000054)	John Miller	11 EA	0.00 USD >
500009065	New	Jun 13, 2018	PicoBit (100000037)	Will Shi	9 EA	85,125.40 USD >
500009064	Delivered	Jun 13, 2018	Danish Fishing Trading Company (100000043)	Will Shi	12 EA	164.00 USD >
500009063	Delivered	Jun 13, 2018	Sorali (100000044)	Klaus Cole	6 EA	3,380.00 USD >
500009062	Delivered	Jun 13, 2018	Anav Ideon (100000054)	John Miller	11 EA	0.00 USD >

Figure 249: List Report

The List Report allows users to filter and work with large amounts of data:

- Easy to consume overview
- Choose from predefined variants
- Create own variants and ad-hoc queries
- Variable visualization
- Simple filtering for simple use cases, powerful filtering for complex use cases

## Object Page

The screenshot shows the SAP Object Page for a "Robot Arm Series 9" with Order ID PO-48865. Key details include:

- General Information:** Manufacturer: Robotech, Status: Delivery, Delivery Time: 12 days, Assembly Option: To Be Selected, Monthly Leasing Instalment: 379.99 USD.
- Order Details:** Order ID: 589946637, Expected Delivery Date: June 23, 2018; Contract: 10045876, Factory: Florida, OL; Transaction Date: May 6, 2018, Supplier: Robotech.
- Configuration Details:** Model: Robot Arm Series 9, Leasing Instalment: 379.99 USD per month; Color: White (default), Axis: 6 Axis; Socket: Default Socket 10.
- Assembly Options:** Work Items with Errors (23) table showing errors for various document numbers (e.g., 10223882001820) across companies like Jologa and DelBont Industries.

Figure 250: Object Page

The Object Page shows all facets of a single business object:

- Successor of the (read-only) Factsheet
- Easy to consume overview
- Create, edit and view business objects
- Type of data determines visualization
- Rich data visualization and embedded analytics

## Overview Pages

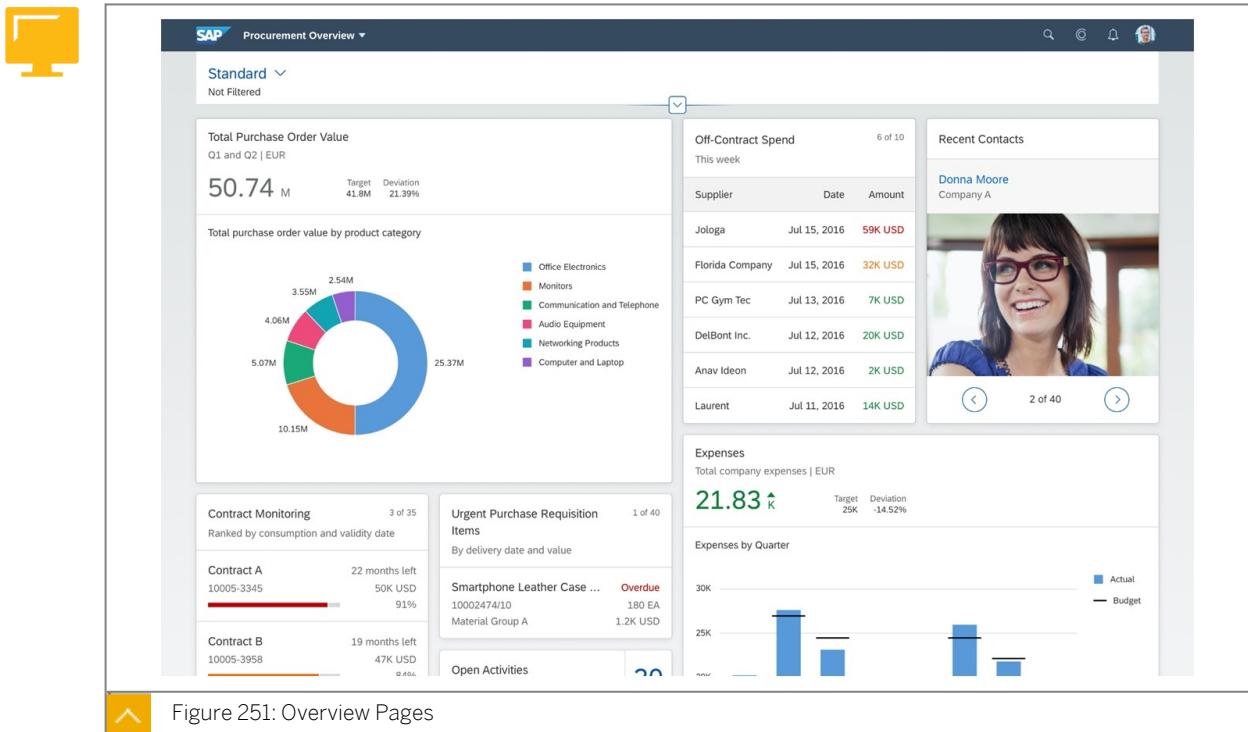


Figure 251: Overview Pages

The Overview Page aggregates domain-specific information:

- Immediate insight on what needs attention
- Access the most important applications in the current business context
- Trigger quick actions or drill down to next level of detail
- Cards efficiently combine information and different visualizations e.g. charts, tables, lists
- Set filters for the whole page and save them as variants, to only show relevant information - e.g. for an individual supplier

## Analytical List Page

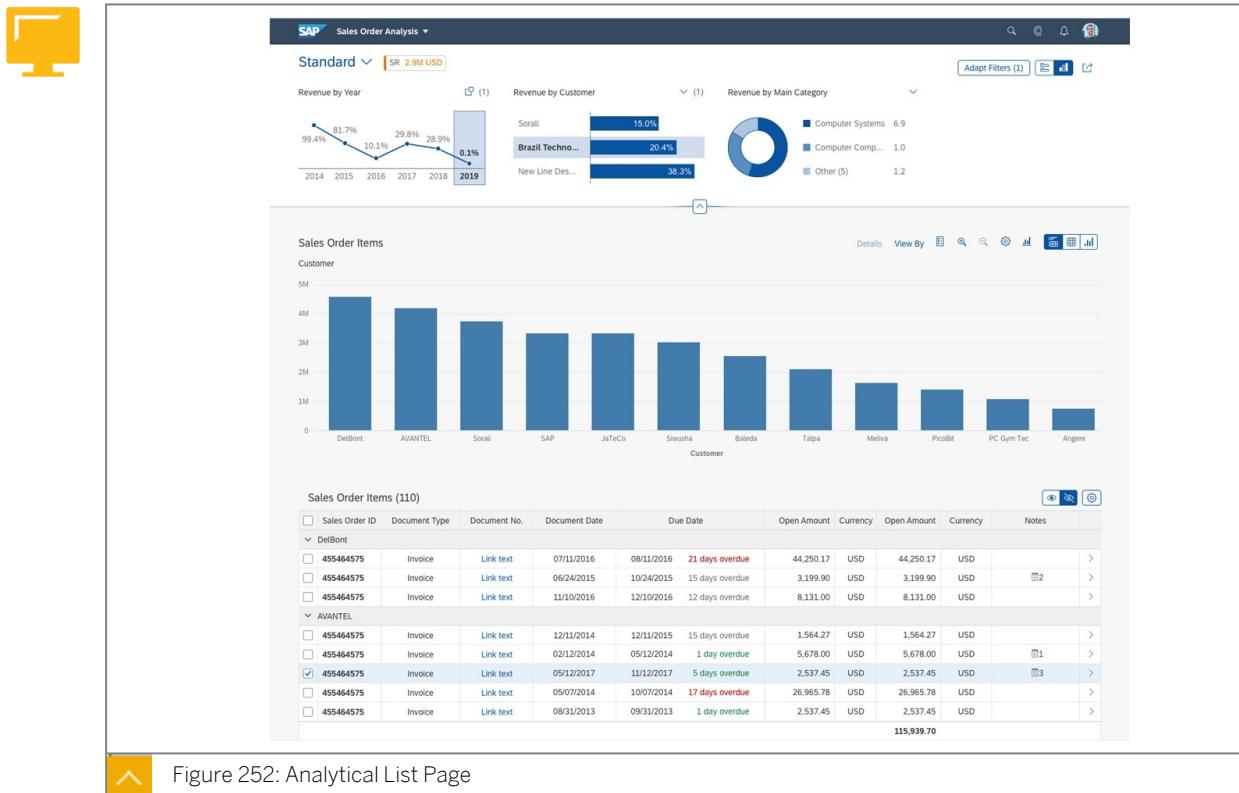


Figure 252: Analytical List Page

The analytical list page enhances the regular list report by applying KPI tags, a visual filter bar, and multiple visualization options.

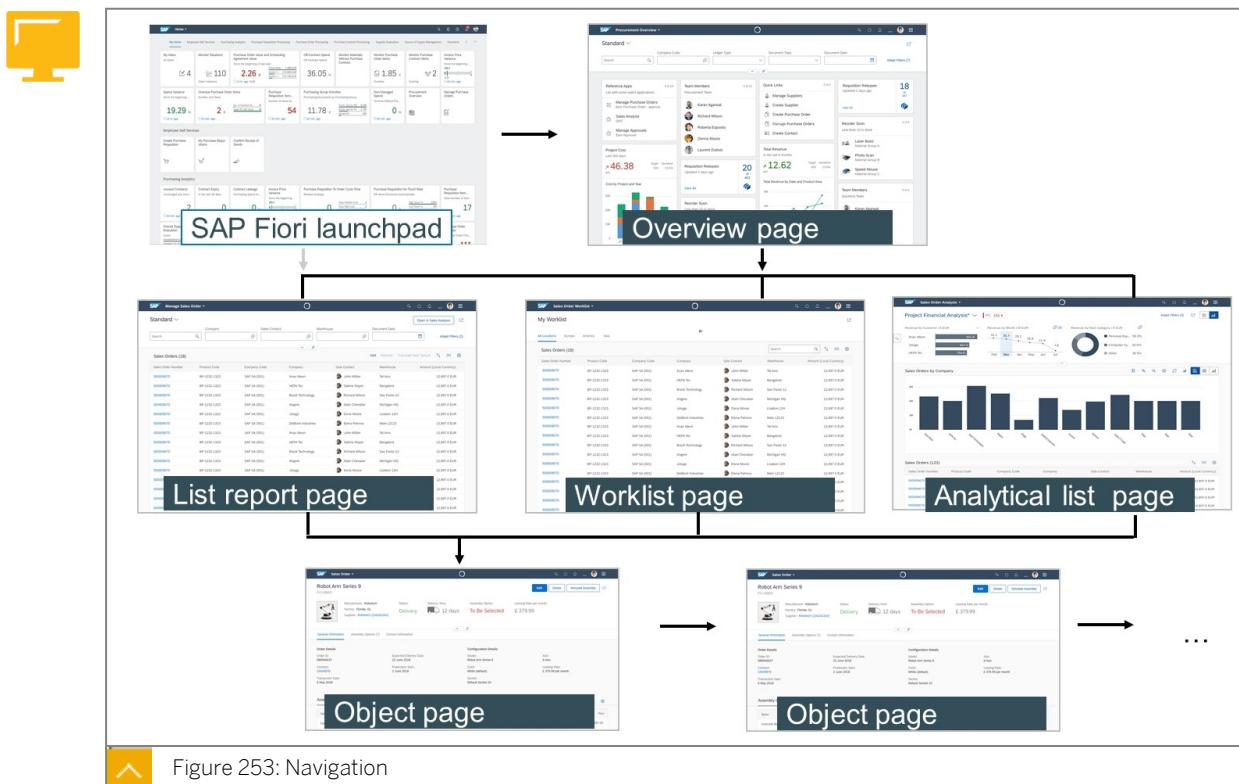
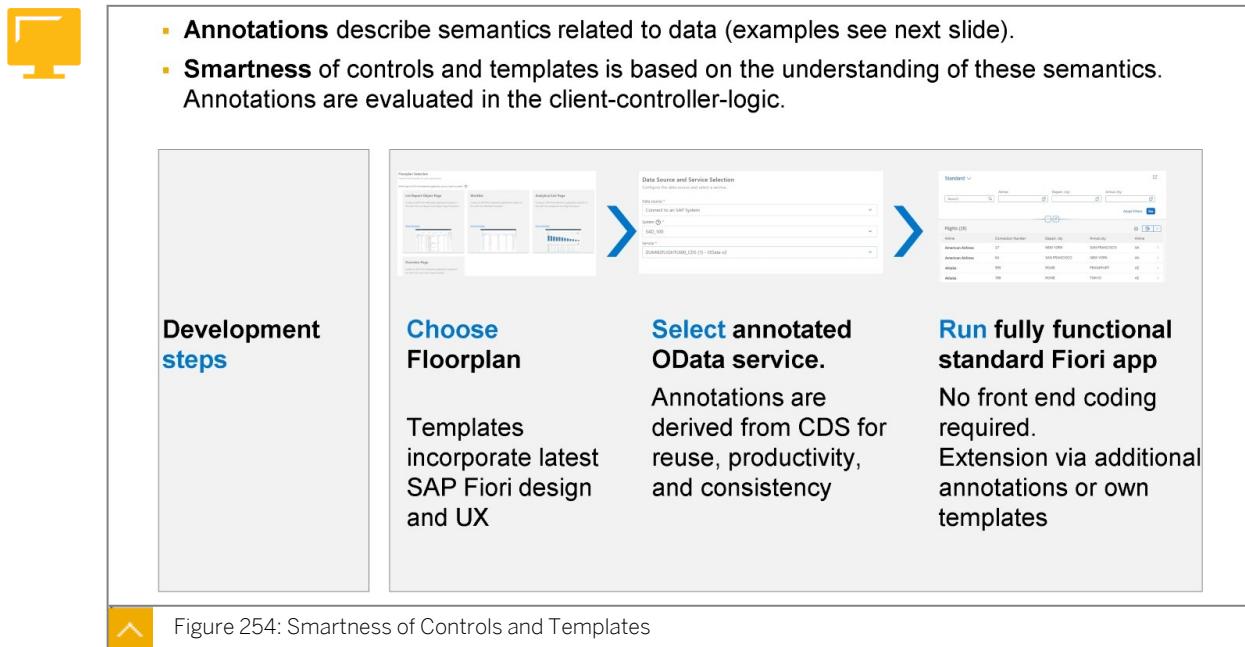


Figure 253: Navigation

SAP Fiori elements is a collection of several common page types that gives you a head start on developing applications that connect to data in SAP back-end systems. These five basic page types cover 80% of the scenarios typically found in SAP applications. We use this approach internally at SAP to create SAP Fiori apps for SAP S/4HANA.

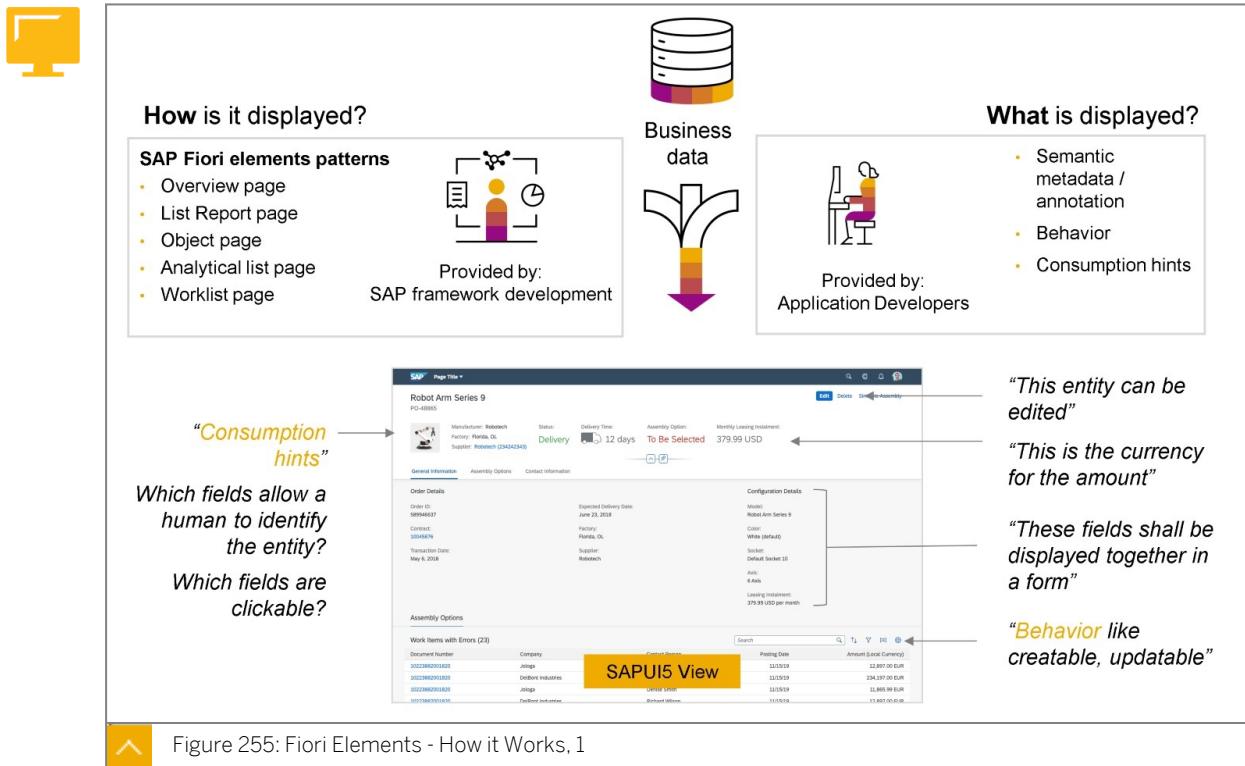
The diagram illustrates the typical flow for a user, where you start at the launchpad, go to an overview page, and then see some sort of list of objects that eventually links you to details on an object page or nested set of object pages. For some situations, you might go directly from a tile on the launchpad directly to a list page or even directly to an object page.

### Smartness of Controls and Templates



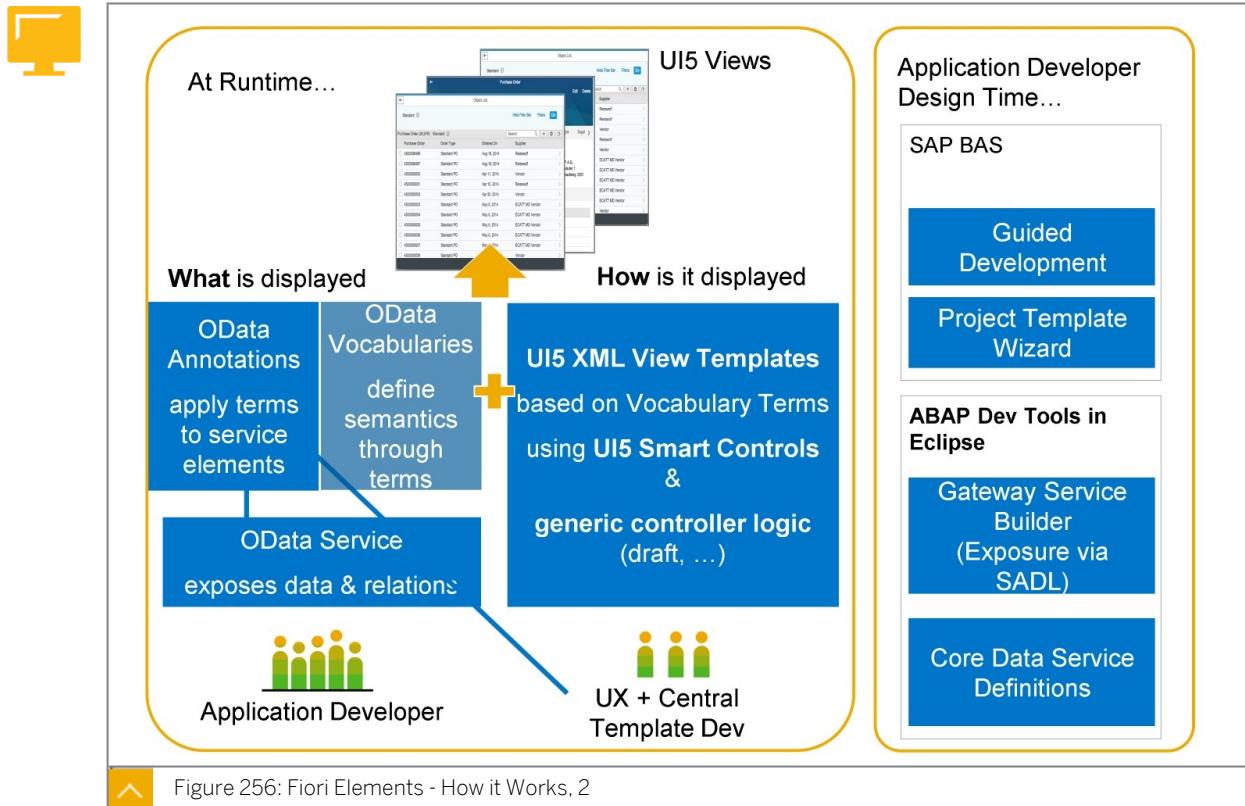
The figure shows the development steps for creating controls and templates.

## Fiori Elements - How it Works



One important goal of SAP Fiori is to achieve a consistent UI over application and industry boundaries. Therefore SAP provided central templates for common SAP Fiori patterns. Application developers/customers only has to add semantical information to the business data to describe what kind of data is displayed. The templates of SAP will read the semantical information and maps these information to UI-controls and UI-elements. So it is just tag parts of a service and tell clients how to interpret a piece of data.

The used vocabulary is language independent. Tagging a string-valued property with this term tells clients that the value they receive may depend on the Content-Language of the response they got.



At design time the developer is using the Core Data Service (CDS) or the SAP NetWeaver Gateway capabilities to implement and expose services as OData-services.

With the SAP Business Application Studio it is possible to add local UI-annotations using the Guided Development. It is not only possible to add local annotations using Guided Development, it is also possible to add annotations to the CDS-View or to the SAP Gateway implementation.

#### Annotation Modeler — Standard Vocabularies

The Annotation modeler supports only standard vocabularies. The following vocabularies are supported:

##### OData.org:

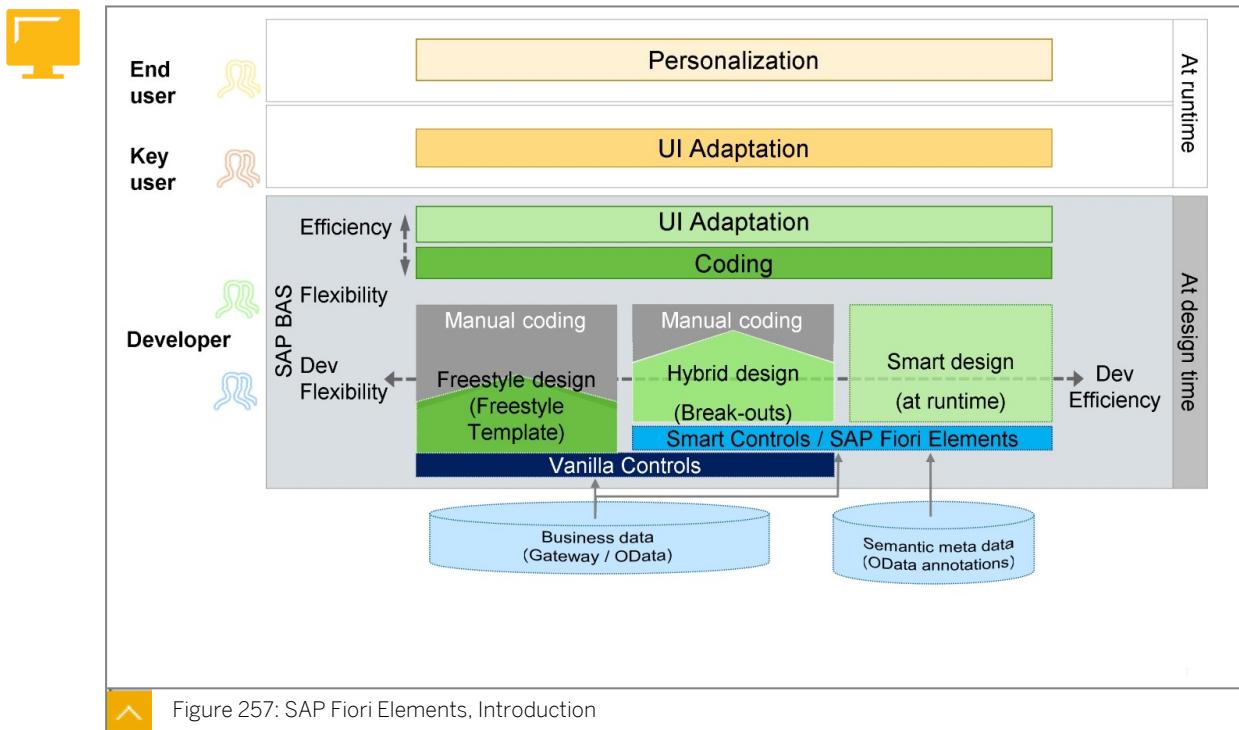
- Aggregation
- Capabilities
- Core
- Measures
- Validation
- Authorization

##### SAP:

- UI
- Common

- Communication

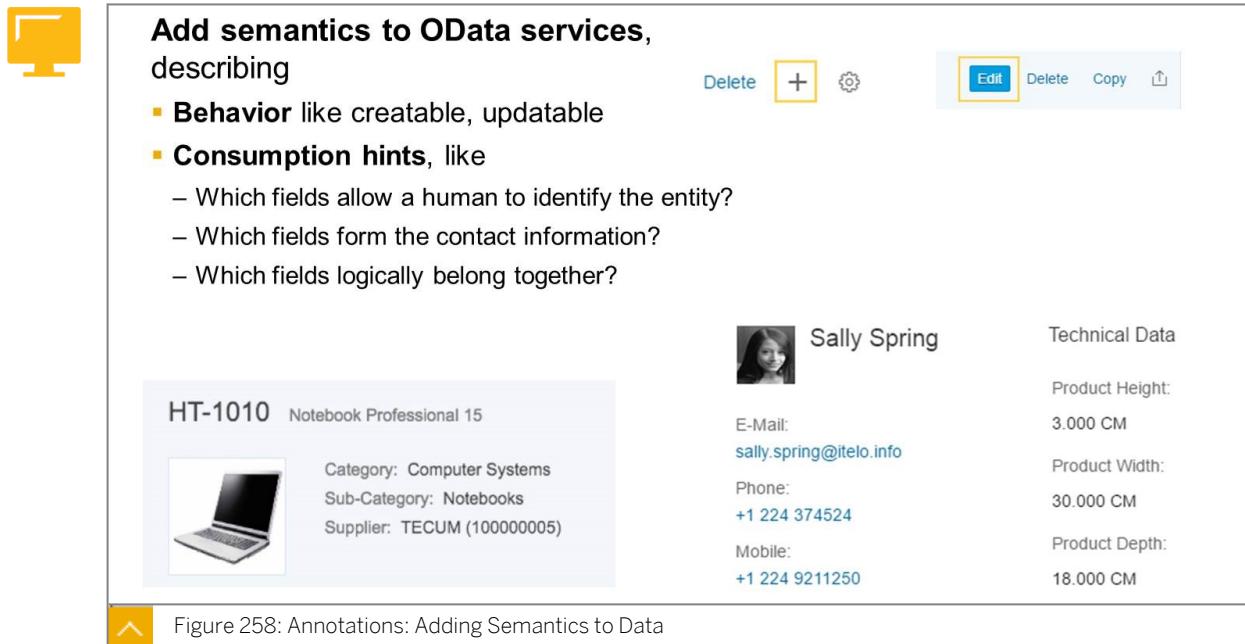
### SAP Fiori Elements, Introduction



SAPUI5 flexibility services allow you to extend your apps based on your requirements, for example, by creating your own variants or adapting the user interface at runtime.

In particular, SAPUI5 flexibility services allow you to manage smart control changes, provide personalization functions for your SAPUI5 apps, and work in multiple layers and store entities in a layered repository. This allows, for example, customers to create their own SAPUI5 entities based on the delivery of SAP without having to modify existing entities in a lower layer.

## Annotations: Adding Semantics to Data



The screenshot shows the SAP Fiori interface for managing annotations. At the top, there's a header with a monitor icon, the title "Add semantics to OData services, describing", and various action buttons like "Delete", "+", "Edit", "Delete", "Copy", and "Upload". Below the header, there's a list item for "HT-1010 Notebook Professional 15" with a small image of a laptop. To the right of the list item, there's a detailed view of the annotation data:

Sally Spring	Technical Data
	Product Height:
	3.000 CM
E-Mail: <a href="mailto:sally.spring@itelo.info">sally.spring@itelo.info</a>	Product Width:
Phone: +1 224 374524	30.000 CM
Mobile: +1 224 9211250	Product Depth:
	18.000 CM

Annotation details for HT-1010:

- Category: Computer Systems
- Sub-Category: Notebooks
- Supplier: TECUM (100000005)

Figure 258: Annotations: Adding Semantics to Data

The figure shows some possibilities by adding annotations.

## Annotations, Vocabulary: Adding Semantics to Data



The screenshot shows an EDMX (Entity Data Model XML) file with several annotations. Two annotations are highlighted with yellow arrows pointing to callout boxes:

- A yellow arrow points from the annotation `<Annotation Term="Core.Description" Qualifier="Published">` to a callout box labeled "Reference to used vocabularies".
- A yellow arrow points from the annotation `<Annotation Term="UI.ThingPerspective" />` to a callout box labeled "Definition of terms and types used in terms".

```

<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
<edmx:Reference Uri="http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/vocabularies/Org.OData.Core.V1.xml">
 <edmx:Include Namespace="Org.OData.Core.V1" Alias="Core" />
</edmx:Reference>
<edmx:Reference Uri="/.../vocabularies/Communication.xml">
 <edmx:Include Namespace="com.sap.vocabularies.Communication.v1" Alias="vCard" />
</edmx:Reference>
<edmx:Reference Uri="/.../vocabularies/Common.xml">
 <edmx:Include Namespace="com.sap.vocabularies.Common.v1" Alias="Common" />
</edmx:Reference>
<edmx:DataServices>
 <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Alias="UI" Namespace="com.sap.vocabularies.UI.v1">
 <Annotation Term="Core.Description">
 <Annotation Term="Core.Descriptive" Qualifier="Published">
 <!-- Semantic Views / Perspectives -->
 <Term Name="HeaderInfo" Type="UI.HeaderInfoType" AppliesTo="EntityType">
 <Annotation Term="UI.ThingPerspective" />
 <Annotation Term="Core.Description">
 String="Information for the header area of an entity representation. HeaderInfo is mandatory for main entity types"
 </Annotation>
 </Term>
 <ComplexType Name="HeaderInfoType">
 <Property Name="TypeName" Type="Edm.String" Nullable="false">
 <Annotation Term="Core.IsLanguageDependent" />
 <Annotation Term="Core.Description" String="Name of the main entity type" />
 </Property>
 <Property Name="TypeNamePlural" Type="Edm.String" Nullable="false">
 <Annotation Term="Core.IsLanguageDependent" />
 <Annotation Term="Core.Description" String="Plural form of the name of the main entity type" />
 </Property>
 <Property Name="Title" Type="UI.DataField" Nullable="false">
 <Annotation Term="Core.Description" String="Title, e.g. for overview pages" />
 </Property>
 <Property Name="Description" Type="UI.DataField" Nullable="true">
 <Property Name="ImageUrl" Type="Edm.String" Nullable="true">
 <Property Name="TypeImageUrl" Type="Edm.String" Nullable="true">
 ...
 </Property>
 </Property>
 </Property>
 </ComplexType>
 </Annotation>
 </Annotation>
 </Schema>
</DataServices>

```

Figure 259: Annotations, Vocabulary: Adding Semantics to Data

An annotation file contains reference to the used vocabularies and definition of terms and types from the vocabulary.

## Annotations, Adding Semantics to Data

The screenshot shows an XML snippet of an EDMX file. It includes sections for `<edmx:References>`, `<edmx:DataServices>`, and `<Annotations>`. Annotations are defined for specific targets like `STTA_PROD_MAN` and `SEPMRA_I`.

- Reference to used vocabularies:** Points to the `<edmx:References>` section.
- Reference to service metadata:** Points to the `<edmx:DataServices>` section.
- Definition of annotations for elements of an OData Service:** Points to the `<Annotations>` section.

Figure 260: Annotations, Adding Semantics to Data

In addition to the used vocabulary the annotation file contains also references to the service metadata. As you can see on the slide an annotation references also a so called target. The target references an entity or property of an entity. In the above slide you can see an entity as a target. The annotation file defines some annotations for this specific target.

### Annotations: Example - Price/Currency Fields Adding Semantics to Data

The screenshot shows a user interface with two columns: "Availability" (In Stock) and "Price" (989.00 USD). Below the UI is an annotation snippet.

**Metadata Extensions:** Points to the first part of the annotation snippet.

```
<Property Name="Price" Type="Edm.Decimal" Precision="16" Scale="3" sap:unit="Currency" sap:label="Price" />
<Property Name="Currency" Type="Edm.String" MaxLength="5" sap:semantics="currency-code" />
```

**Vocabulary-based Annotations:** Points to the second part of the annotation snippet.

```
<Annotations Target="SEPMRA_PROD_MAN.SEPMRA_C_PD_ProductType/Price">
 <Annotation Term="Org.OData.Measures.V1.ISOCurrency" Path="Currency"/>
 <Annotation Term="com.sap.vocabularies.Common.v1.Label" String="Price"/>
</Annotations>
```

Figure 261: Annotations: Example - Price/Currency Fields Adding Semantics to Data

As of SAP NetWeaver AS for ABAP 7.51 innovation package, you can use metadata extensions to add customer-specific requirements to SAP's CDS entities. Note that these changes do not result in modifications. A metadata extension is a transportable ABAP development object that provides CDS annotations in order to extend the CDS annotations used in a CDS view. Metadata extensions enable you to write the annotations for a CDS view in a different document to separate them from the CDS view. In the above slide you can see the differences between using metadata extension compared to vocabulary based annotations.

## Annotations: Example - Field Group in Form, Adding Semantics to Data

HT-1010 Notebook Professional 15

Category: Computer Systems  
Sub-Category: Notebooks  
Supplier: TECUM (100000005)

PRODUCT INFORMATION REVIEWS

Technical Data

Product Height:	3.000 CM
Product Width:	30.000 CM
Product Depth:	18.000 CM

```

<Annotations Target="SEPMRA_PROD_MAN.SEPMRA_C_PD_ProductType">
 <Annotation Term="com.sap.vocabularies.UI.v1.FieldGroup" Qualifier="TechData">
 <Record>
 <PropertyValue Property="Label" String="Technical Data" />
 <PropertyValue Property="Data">
 <Collection>
 <Record Type="com.sap.vocabularies.UI.v1.DataField">
 <PropertyValue Property="Value" Path="Height" />
 </Record>
 <Record Type="com.sap.vocabularies.UI.v1.DataField">
 <PropertyValue Property="Value" Path="Width" />
 </Record>
 <Record Type="com.sap.vocabularies.UI.v1.DataField">
 <PropertyValue Property="Value" Path="Depth" />
 </Record>
 </Collection>
 </PropertyValue>
 </Record>
 </Annotation>
</Annotations>

```

Vocabulary-based Annotations

Figure 262: Annotations: Example - Field Group in Form, Adding Semantics to Data

In the above slide you can see the mapping of an annotation to UI-controls. You can see the definition of a so called FieldGroup. A field group may contain DataField definition. A DataField, as the name says, is the description of a single field.

## Annotations: Core Data Services (CDS) Annotations, Adding Semantics to Data



- Data Definition Language (DDL) based on SQL and Entities, Associations & Annotations is converted into OData Annotations of the corresponding service

```

@AbapCatalog.sqlViewName: 'ZUX402FlightUI'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'View mit UI-Annotierungen'
@Odata.publish: true
@UI.headerInfo({
 typeName: 'ZUX402FlightUI00',
 typeNamePlural: 'ZUX402FlightsUI00',
 title: {value: 'carname'},
 imageUrl: 'imgUrl',
 description: {value: 'connid'}
})

define view ZUX402FlightUI00
as select from spfli join scarr on scarr.carrid = spfli.carrid
{
 @UI.lineItem.position:10
 key spfli.carrid,
 @UI.selectionField.position:10
 @UI.lineItem.position:20
 key scarr.carname,
 @UI.fieldGroup:[{qualifier:'FlightData',groupLabel:'FlightData',position:10}]
 @UI.lineItem.position:30
 key spfli.connid,
 @UI.fieldGroup:[{qualifier:'FlightData',groupLabel:'FlightData',position:20}]
 @UI.selectionField.position:20
 @UI.lineItem.position:40
 spfli.cityFrom,
 @UI.fieldGroup:[{qualifier:'FlightData',groupLabel:'FlightData',position:30}]
 @UI.selectionField.position:30
 @UI.lineItem.position:50
 spfli.cityTo,
 concat(concat('/webapp/img/',spfli.carrid),".png") as imgUri
}

```

Core Data Services

Definition of a **headerInfo** annotation

Definition of a **lineItem** annotation

Definition of a **selectionField** annotation

Definition of a **FieldGroup** annotation

Using the build in functions



Figure 263: Annotations: Core Data Services (CDS) Annotations, Adding Semantics to Data

The above slide shows you a CDS-view with annotations. These annotation will be used to render a ListReport-UI. You can see the result in the “Smart templates – semantic information” figure following.



Note:

The sample given shows only a simple ABAP CDS for demonstration purposes. In real-world projects, you must keep in mind that there are several aspects to consider, such as naming conventions, different CDS types, and so on.

## Smart Templates - Semantic Information

Figure 264: Smart Templates - Semantic Information

The UI-Annotations used in the CDS-View or in the Annotation modeller are mapped to UI-controls and elements at runtime. As you can see in the above slide the *UI.SelectionField-annotation* is rendered as an *InputField* with F4-capabilities as part of *SmartFilterBar-control*. A *LineItem* is rendered as part of a *List* with navigation. When the user clicks on a *ListItem* the *ObjectView-pattern* is used to display the details of the selected Item. One UI-annotation used in the *ObjectView* is the *ObjectHeader*, to show the *ObjectHeader* the *UI.HeaderInfo-annotation* is used.



### LESSON SUMMARY

You should now be able to:

- Get an Introduction to SAP Fiori Elements



## Learning Assessment

- When do you need to work with local data?

*Choose the correct answers.*

- A When working with static data.
- B When performance is poor in the productive system.
- C To perform a quick test without creating live entities on the back-end server.
- D When you want to reduce the complexity of your application.

- What query option is needed to access the data of an entity in the JSON format?

*Choose the correct answer.*

- A \$json
- B \$format=json
- C format=json

- Where is the best place to store local data when using the SAP Business Application Studio?

*Choose the correct answers.*

- A In the `models` folder.
- B In the `localService` folder.
- C In the `mockdata` folder.

- What class in the SAPUI5 API supports back-end mock up and is recommended by SAP?

*Choose the correct answer.*

- A `sap.ui.app.MockServer`
- B `sap.ui.core.util.MockServer`
- C `sap.m.MockServer`
- D `sap.ui.core.MockServer`

5. What parameter is needed when instantiating a `MockServer` object?

*Choose the correct answer.*

- A `uri`
- B `rootUri`
- C `url`
- D `rootUrl`

6. What parameter must you provide when calling the `simulate` function of the mock server?

*Choose the correct answer.*

- A The full qualified path to the model and the URI for the service document.
- B The full qualified path to the local metadata file and the folder where the model data are stored locally.
- C The `autoRespondAfter` parameter and the full qualified path to the local metadata file.

7. What type of concurrency control is offered by OData?

*Choose the correct answer.*

- A Pessimistic concurrency control
- B Exclusive concurrency control
- C No concurrency control
- D Optimistic concurrency control

8. How does the OData Model handle XSRF tokens?

*Choose the correct answer.*

- A The OData Model does not handle XSRF-tokens. The developer must implement token handling.
- B The OData Model fetches the token when reading the metadata and sends the token automatically in each write request header.
- C The OData Model fetches the token for each request and sends the token automatically with each request.

9. What is returned by the function `createEntry`?

*Choose the correct answer.*

- A The ID of the newly created entity.
- B A binding context object.
- C A newly created object in JSON-format.

10. When do you use deep insert?

*Choose the correct answer.*

- A When working with aggregation binding to update / insert an entity in the bound complex structure.
- B When adding a new entity to an entity set.
- C When working with hierarchical data where the data can only be stored in the full hierarchy.

11. What method must be implemented on the back end to support deep inserts?

*Choose the correct answer.*

- A `INSERT_ENTITY_DEEP`
- B `CREATE_ENTITY_DEEP`
- C `CREATE_DEEP_ENTITY`
- D `APPLY_DEEP_ENTITY`

12. Which function do you call on the ODataModel to trigger a deep insert?

*Choose the correct answer.*

- A `create`
- B `createDeep`
- C `insertDeep`
- D `updateDeep`

13. What is the namespace in which the smart controls are bundled?

*Choose the correct answer.*

- A sap.ui.smart
- B sap.ui.comp
- C sap.m

14. Which aggregation is used to overwrite the standard behavior of a *SmartField* control.

*Choose the correct answer.*

- A overwrite
- B configure
- C extension

15. Which of the following statements are true for SAP Fiori elements?

*Choose the correct answers.*

- A Heavy UI coding is necessary when working with SAP Fiori Elements.
- B SAP Fiori elements provide designs for UI patterns and predefined templates.
- C Apps are based on OData services and annotations.

16. Which of the following describe benefits of SAP Fiori elements?

*Choose the correct answers.*

- A Consistent UI design.
- B Full control of the generated code at design time.
- C Less support effort.
- D Flexibility and freedom in UI design.

17. Which of the following SAP Fiori element based UIs are currently available?

*Choose the correct answers.*

- A** Wizard Form
- B** List Report
- C** Master-Detail
- D** Overview Pages
- E** Object Page

18. Which UI annotation is used in the object page to display the object header?

*Choose the correct answer.*

- A** UI.headerObject
- B** UI.objectheader
- C** UI.headerInfo
- D** UI.header

19. Which UI annotation is used to declare a field for the smart filter template?

*Choose the correct answer.*

- A** UI.searchField
- B** UI.selectionField
- C** UI.filterField
- D** UI.finderField

## Learning Assessment - Answers

- When do you need to work with local data?

*Choose the correct answers.*

- A When working with static data.
- B When performance is poor in the productive system.
- C To perform a quick test without creating live entities on the back-end server.
- D When you want to reduce the complexity of your application.

Correct. You must work with local data when you use static data, for example, data provided in a file located inside the SAPUI5 project. Also, it is necessary to have local data when testing the application without changing live data on the back-end system.

- What query option is needed to access the data of an entity in the JSON format?

*Choose the correct answer.*

- A \$json
- B \$format=json
- C format=json

Correct. You use the query option `$format=json` to retrieve the data in JSON format from the OData service.

- Where is the best place to store local data when using the SAP Business Application Studio?

*Choose the correct answers.*

- A In the `models` folder.
- B In the `localService` folder.
- C In the `mockdata` folder.

Correct. You should store local data in the `localService` folder or in a folder with the name `mockdata`.

4. What class in the SAPUI5 API supports back-end mock up and is recommended by SAP?

*Choose the correct answer.*

- A sap.ui.app.MockServer
- B sap.ui.core.util.MockServer
- C sap.m.MockServer
- D sap.ui.core.MockServer

Correct. The recommended `MockServer` implementation is located in the `sap.ui.core.util` namespace.

5. What parameter is needed when instantiating a `MockServer` object?

*Choose the correct answer.*

- A `uri`
- B `rootUri`
- C `url`
- D `rootUrl`

Correct. To instantiate the `MockServer`, you must provide the `rootUri`. The `rootUri` must be a relative address and requires a trailing slash (/) character. Also, it must match the URI set in the OData/JSON models or in the simple XHR calls for the mock server to intercept them. The default value is empty/undefined.

6. What parameter must you provide when calling the `simulate` function of the mock server?

*Choose the correct answer.*

- A The full qualified path to the model and the URI for the service document.
- B The full qualified path to the local metadata file and the folder where the model data are stored locally.
- C The `autoRespondAfter` parameter and the full qualified path to the local metadata file.

Correct. You must provide the full qualified path to the local metadata file and the folder where the model data are stored locally.

7. What type of concurrency control is offered by OData?

*Choose the correct answer.*

- A Pessimistic concurrency control
- B Exclusive concurrency control
- C No concurrency control
- D Optimistic concurrency control

Correct. OData offers optimistic concurrency control.

8. How does the OData Model handle XSRF tokens?

*Choose the correct answer.*

- A The OData Model does not handle XSRF-tokens. The developer must implement token handling.
- B The OData Model fetches the token when reading the metadata and sends the token automatically in each write request header.
- C The OData Model fetches the token for each request and sends the token automatically with each request.

Correct. The OData Model fetches the token when reading the metadata and sends the token automatically in each write request header.

9. What is returned by the function `createEntry`?

*Choose the correct answer.*

- A The ID of the newly created entity.
- B A binding context object.
- C A newly created object in JSON-format.

Correct. The `createEntry` function returns a binding context object. This object can be used to bind to a form for example.

10. When do you use deep insert?

*Choose the correct answer.*

- A When working with aggregation binding to update / insert an entity in the bound complex structure.
- B When adding a new entity to an entity set.
- C When working with hierarchical data where the data can only be stored in the full hierarchy.

Correct. A deep insert is used when you are working with hierarchical data, where the data can only be stored in the full hierarchy.

11. What method must be implemented on the back end to support deep inserts?

*Choose the correct answer.*

- A INSERT\_ENTITY\_DEEP
- B CREATE\_ENTITY\_DEEP
- C CREATE\_DEEP\_ENTITY
- D APPLY\_DEEP\_ENTITY

Correct. The back-end developer must implement the CREATE\_DEEP\_ENTITY method in the ODatas service implementation.

12. Which function do you call on the ODataModel to trigger a deep insert?

*Choose the correct answer.*

- A create
- B createDeep
- C insertDeep
- D updateDeep

Correct. A deep insert is triggered using the create function. The developer must pass the full hierarchy.

13. What is the namespace in which the smart controls are bundled?

*Choose the correct answer.*

- A sap.ui.smart
- B sap.ui.comp
- C sap.m

Correct. Smart controls are bundled in the sap.ui.comp namespace.

14. Which aggregation is used to overwrite the standard behavior of a *SmartField* control.

*Choose the correct answer.*

- A overwrite
- B configure
- C extension

Correct. The configuration aggregation is used to overwrite the standard behavior of a *SmartField* control.

15. Which of the following statements are true for SAP Fiori elements?

*Choose the correct answers.*

- A Heavy UI coding is necessary when working with SAP Fiori Elements.
- B SAP Fiori elements provide designs for UI patterns and predefined templates.
- C Apps are based on OData services and annotations.

Correct. SAP Fiori elements provides designs for UI patterns and predefined templates. The apps are based on OData services and use annotations.

16. Which of the following describe benefits of SAP Fiori elements?

*Choose the correct answers.*

- A Consistent UI design.
- B Full control of the generated code at design time.
- C Less support effort.
- D Flexibility and freedom in UI design.

Correct. The benefits of using SAP Fiori elements are that you have consistent UI designs and less support effort because you do not need to implement any SAPUI5 relevant code.

17. Which of the following SAP Fiori element based UIs are currently available?

*Choose the correct answers.*

- A Wizard Form
- B List Report
- C Master-Detail
- D Overview Pages
- E Object Page

Correct. Currently, there are templates for the List Report, Overview Page, and Object Page pattern.

18. Which UI annotation is used in the object page to display the object header?

*Choose the correct answer.*

- A `UI.headerObject`
- B `UI.objectheader`
- C `UI.headerInfo`
- D `UI.header`

Correct. The `UI.headerInfo` annotation is used to annotate the object header data.

19. Which UI annotation is used to declare a field for the smart filter template?

*Choose the correct answer.*

- A `UI.searchField`
- B `UI.selectionField`
- C `UI.filterField`
- D `UI.finderField`

Correct. The `UI.selectionField` annotation is used to declare a smart filter template.



## UNIT 4

# Application Extensibility, Introduction

### Lesson 1

Introducing SAPUI5 Flexibility

269

### Lesson 2

Explaining Extension Points

273

### Lesson 3

Describing Other Types of Extensibility in SAPUI5

283

### UNIT OBJECTIVES

- Understand SAPUI5 Flexibility
- Explain Extension Capabilities in SAPUI5
- Describe other Types of Extensibility in SAPUI5 Application



# Introducing SAPUI5 Flexibility



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand SAPUI5 Flexibility

## SAPUI5 Flexibility, an Introduction



- SAPUI5 flexibility enables functions for different user groups to adapt SAPUI5 applications in a simple and modification-free way.
- Available on ABAP platform, SAP Cloud Platform services in the Cloud Foundry environment.
- Replaces the extensibility concept by broadening the adaptability of SAPUI5 application and simultaneous increase of maintainability and simplicity.



Ensures **lifecycle-stable and modification-free**  
UI changes based on deltas



Facilitates **cost-efficient UI change process** for extending apps



Provides **intuitive tooling** tailored to the needs of special target groups



Figure 265: SAPUI5 Flexibility

- SAPUI5 flexibility enables functions for different user groups to adapt SAPUI5 applications in a simple and modification-free way.
- Available on ABAP platform, SAP Cloud services on SAP Business Technology Platform.
- Replaces the extensibility concept by broadening the adaptability of SAPUI5 application and simultaneous increase of maintainability and simplicity.

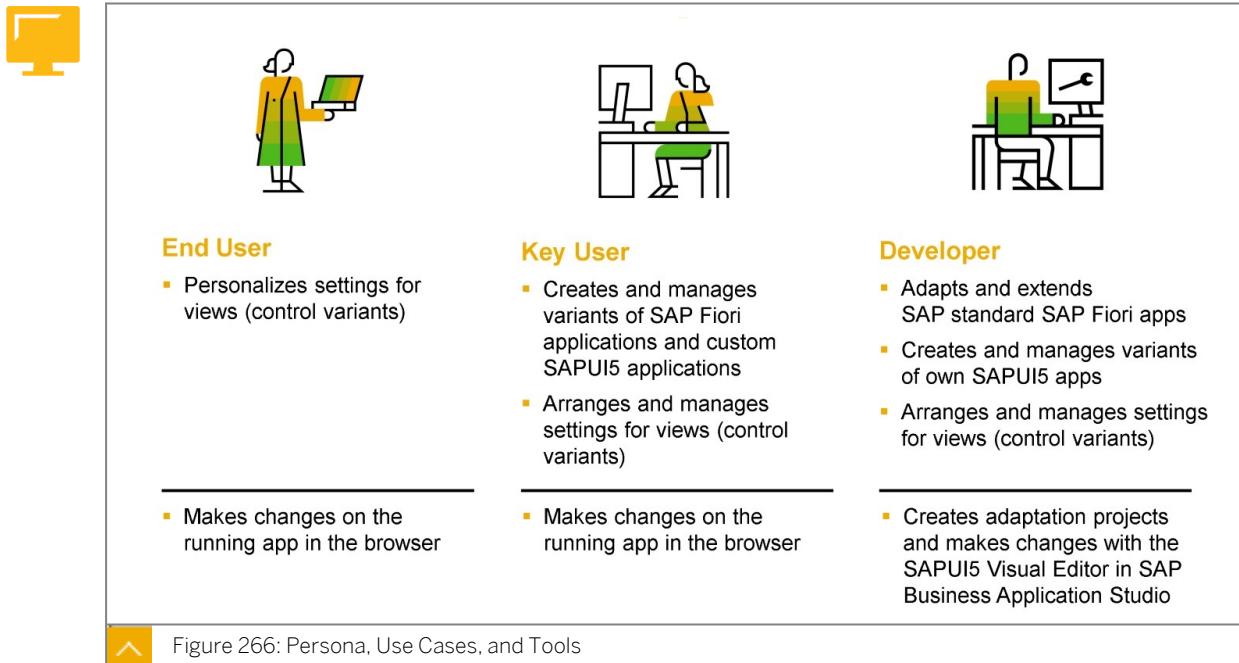


Figure 266: Persona, Use Cases, and Tools

UI adaptation is a feature of SAPUI5 flexibility that allows key users without technical knowledge and developers to easily make UI changes in a WYSIWYG manner.

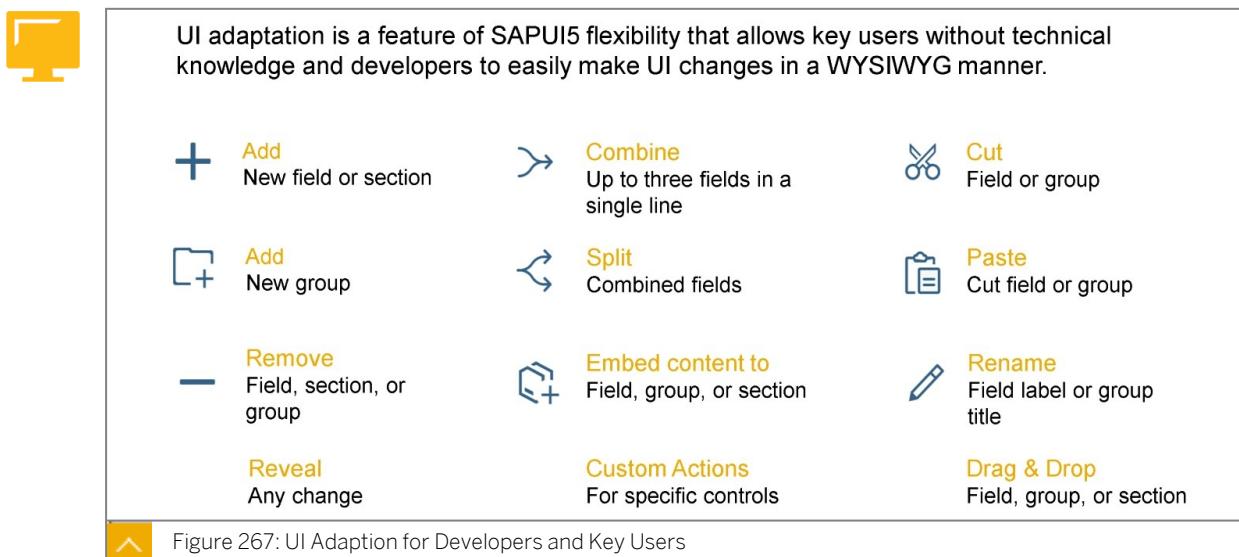


Figure 267: UI Adaption for Developers and Key Users



**SAPUI5 flexibility allows UI adjustments by creating app variants from existing applications. The UI of the applications can therefore be adapted separately without touching the original app.**

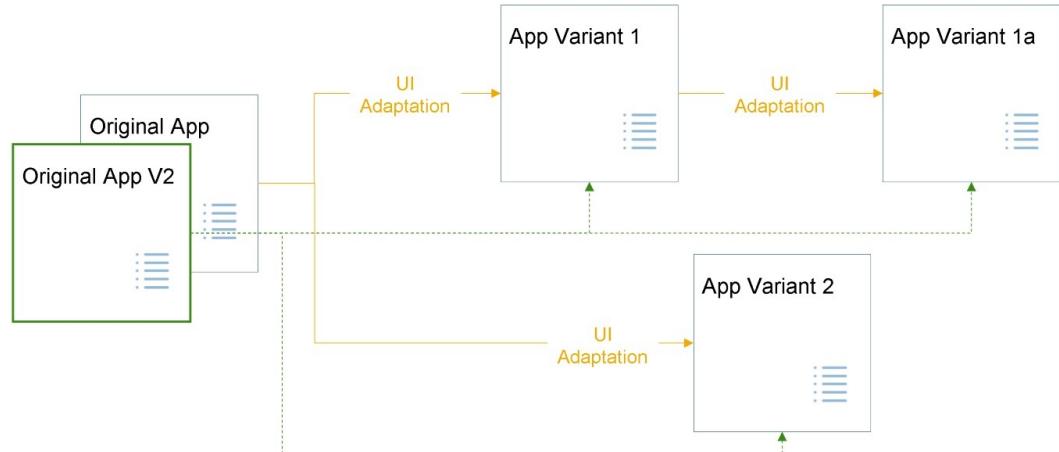


Figure 268: App Variants

Further details on the App Variant concept can be found at: <https://help.sap.com/viewer/a7b390faab1140c087b8926571e942b7/201809.002/en-US/af47058ad66144579db6a990f3b7b919.html>



### LESSON SUMMARY

You should now be able to:

- Understand SAPUI5 Flexibility



## Explaining Extension Points



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain Extension Capabilities in SAPUI5

### Extension Concept and Extension Points

#### Scenario

As a Business Process owner you are requested to permanently improve the user experience in your area of responsibility.

Creating user interfaces is a critical aspect in the user experience. In this training, you will learn how to develop SAPUI5 user interfaces to induce a great user experience.

#### Extension Capabilities in SAPUI5

The following are basic statements about SAPUI5:



- SAPUI5 applications can be extended to meet specific user requirements.
- Placeholders for extensions, called extension points, are placed into application views.
  - This allows developers to add extensions in these locations, if desired.
- If use of an extension point is desired, then the physical extensions are documented in the application's component (*Component.js*):
  - The standard application is not changed.
  - The customized application becomes the start-up project which launches the standard application with the additional customizations.
- Many SAP delivered Fiori applications have extension points for easy customization.

## SAP UI5 Extensibility Concept

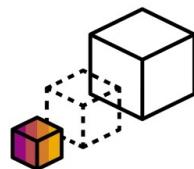


In SAP Business Application Studio, an adaptation project lets customers and SAP developers create an app variant for an existing application, adapt them via SAPUI5 Visual Editor, store created projects to Git, and deploy to the system.

The SAPUI5 Visual Editor is a design-time editor in SAP BAS providing capabilities to adapt existing SAPUI5 applications without altering its base code.



**Extension**  
of standard SAP Fiori  
applications as app  
variants  
(semantic/property  
changes, view/controller  
/i18n extension)



**Creation**  
of views (control  
variants – flex  
variant  
management).



Figure 269: SAP UI5 Extensibility Concept: UI Extensibility

### Configuration of Components

- Customization is based on the configuration of Components. A special area of their configuration is dedicated to customization information. This configuration is located in a JavaScript file named *Component.js*.
- Customization can be performed on a custom application that extends a delivered standard application. The custom application is located in a separate project. Both applications contain the *Component.js* file and the custom application contains all the changes.

### Modification free

- The delivered standard application remains unchanged and hence the extension is considered to be modification free.

### Custom application

- The custom application becomes the start-up project which then launches the delivered standard application with the additional customizing configuration.

For more details, see <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Customization.html>.

### View Extensions and Extension Points

The following are basic facts about View Extensions and Extension Points:



- Allowing for view extensions requires that extension points be placed in the view code.
- An extension point can be placed anywhere in a view that you want to provide a placeholder for later customization.
- Multiple extension points can be placed in a view.
- Extension points have one property, name, that makes the point unique.
- While other components in SAPUI5 applications, like controllers can be extended, extension points are only put into your view code.

## Example - Extension Point in a View

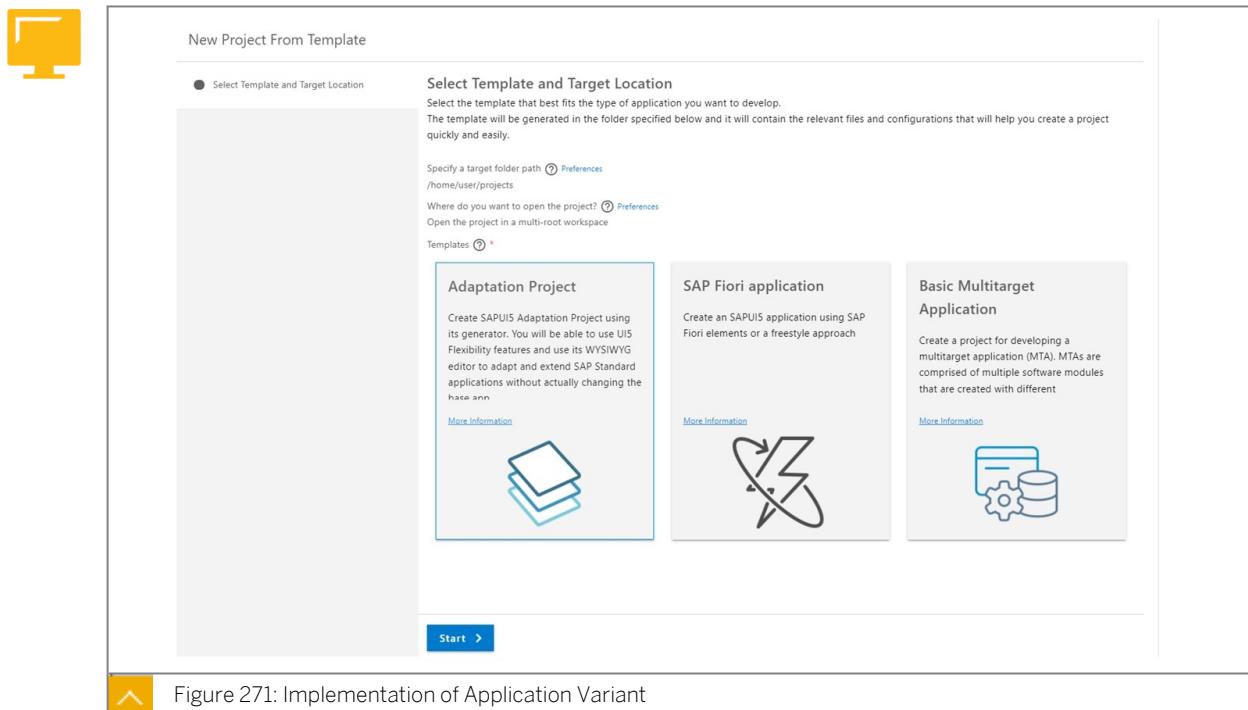


```
Carrier.view.xml x
11
12 <uxap:snappedHeader>
13 <Title text="{Carrname}"/>
14 </uxap:snappedHeader>
15 <uxap:expandedContent>
16 <FlexBox alignItems="Start" justifyContent="SpaceBetween" id="idCarrierDetails">
17 <items>
18 <layout:HorizontalLayout allowWrapping="true">
19 <layout:VerticalLayout class="sapUiMediumMarginEnd">
20 <ObjectAttribute title="{i18n>currLabelText}" text="{Currcode}"/>
21 <ObjectAttribute title="{i18n>urlLabelText}" text="{Url}"/>
22 </layout:VerticalLayout>
23 </items>
24 </FlexBox>
25 <core:ExtensionPoint name="ux410_extension"/>
```

Figure 270: Example - Extension Point in a View

The figure shows an example of an implementation of an extension point in a view.

## Implementation of Application Variant



SAPUI5 Adaptation Project allows developers to extend SAP Fiori application in SAP Business Application Studio.

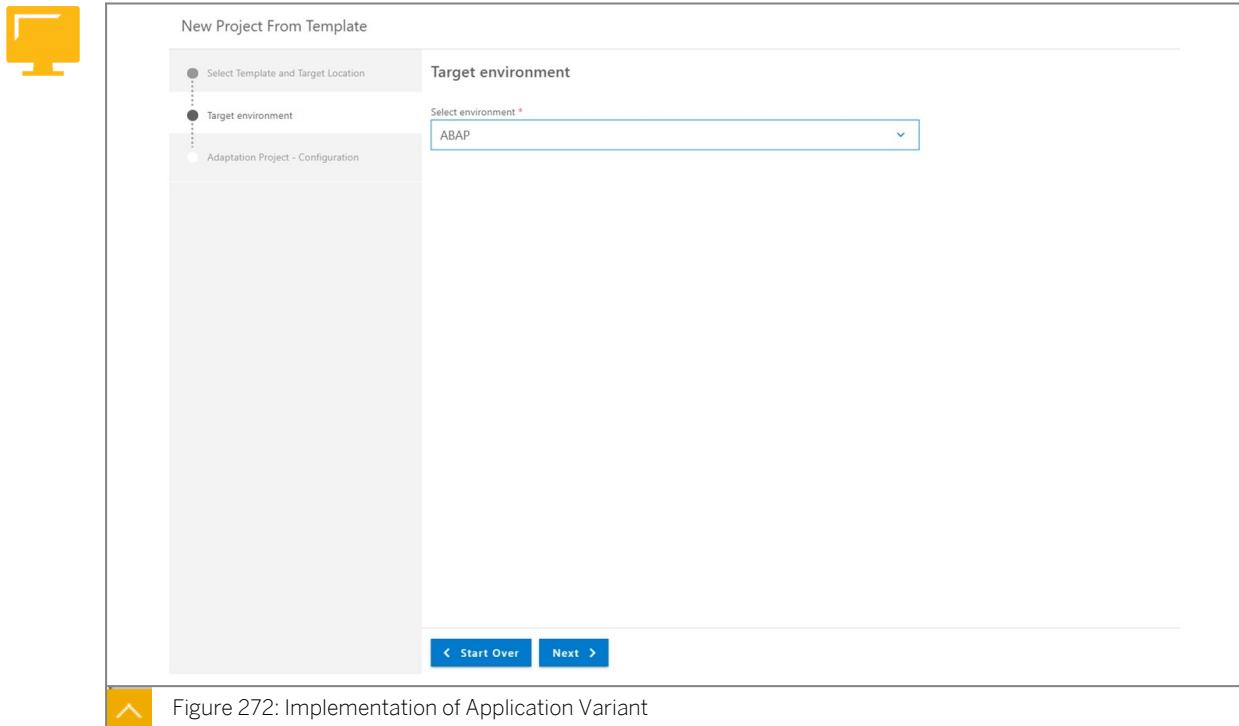


Figure 272: Implementation of Application Variant

In the second step of the wizard, you have to select the environment where the base application is hosted. Currently ABAP and Cloud Foundry is available.

### Provide Basic Project Information

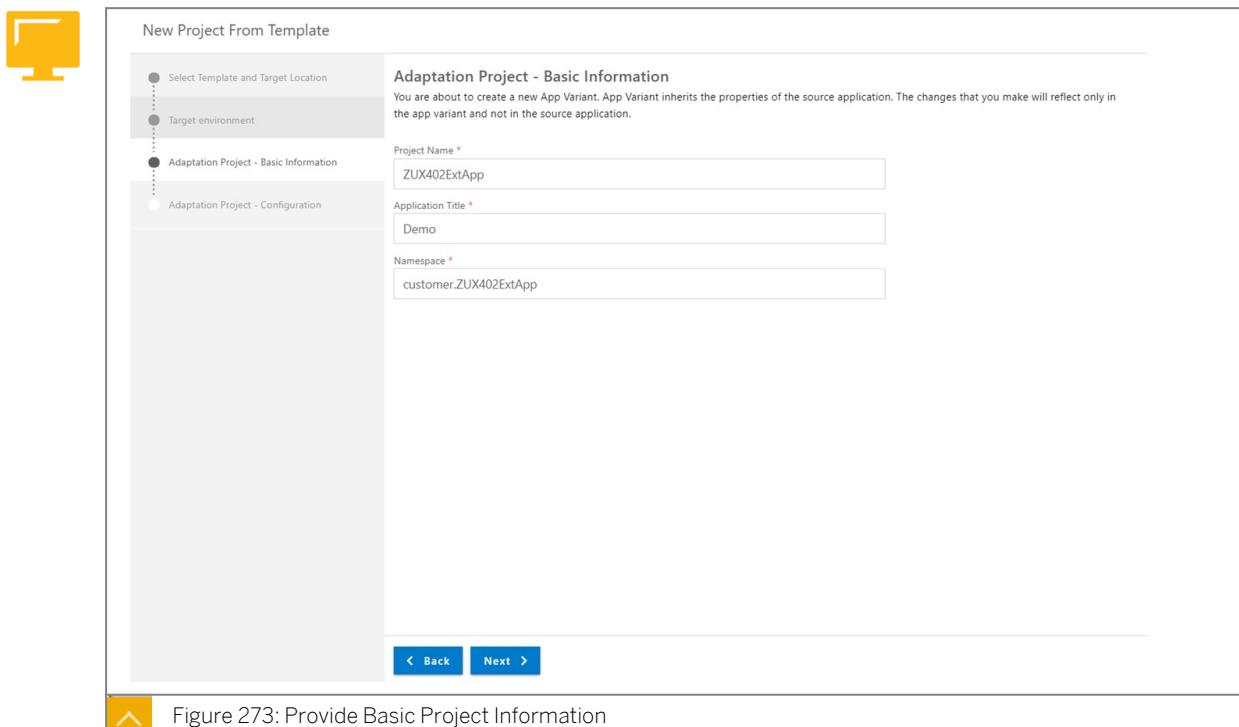
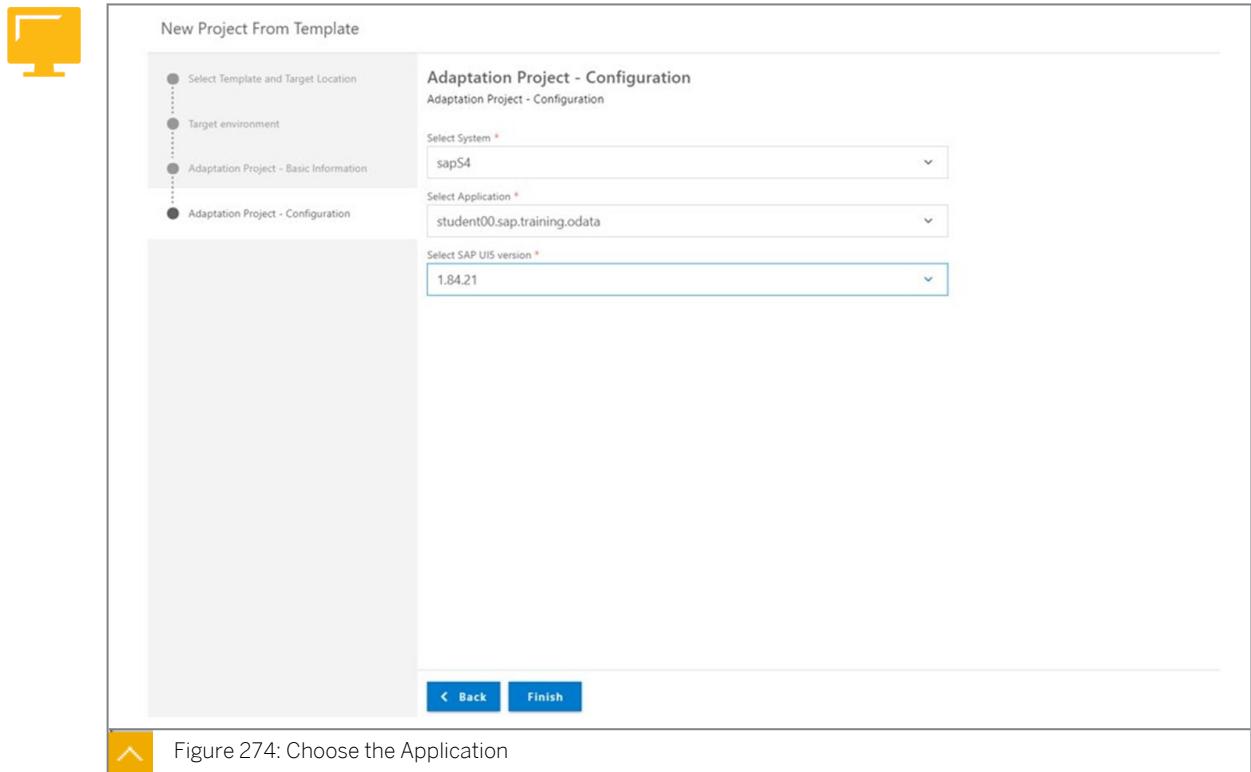


Figure 273: Provide Basic Project Information

The developer must select the source were the application that needs to be extended can be found.

## Choose the Application



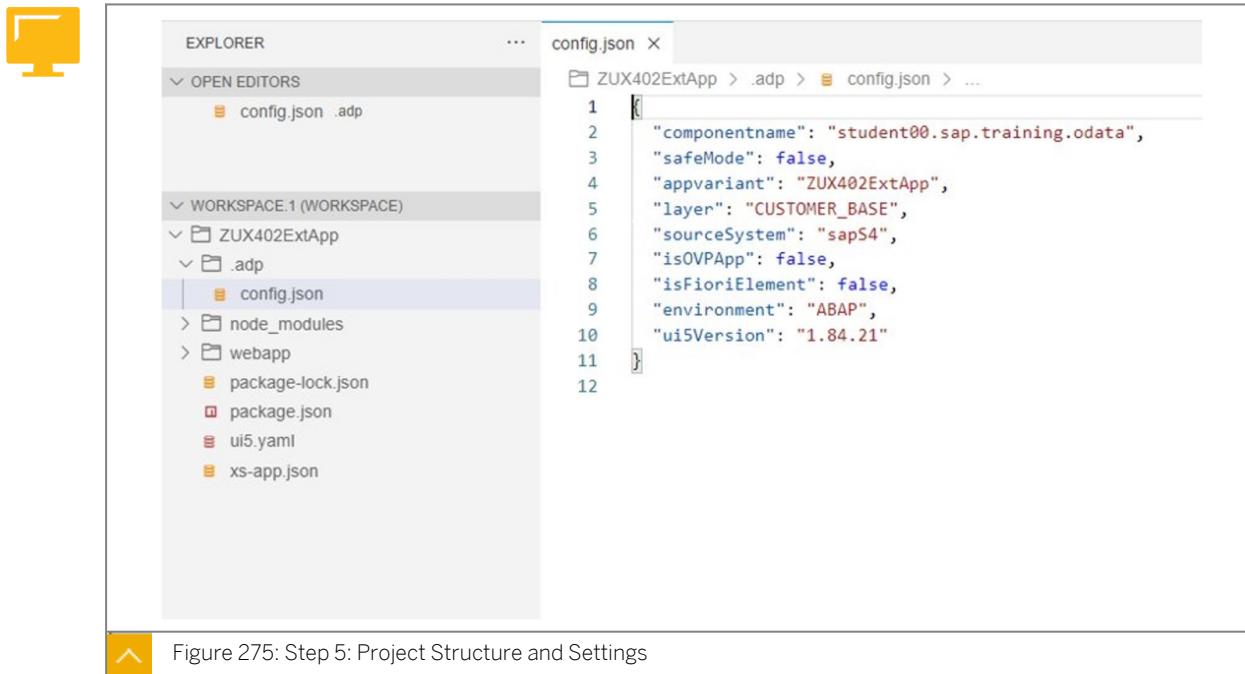
In the configuration step the system is selected where the base application is provided. In the select application drop down you have to choose the namespace of the base application. In the SAP UI5 version drop down you have to choose the SAPUI5 version.

Please be aware that in the SAPUI5-Version 1.96.0/1.96.1 and few 1.97 snapshot version the functional scope of the visual editor is limited. Therefore don't use these SAPUI5-versions from the list.

### Open Adaption Project in New Workspace

After the adaption project is generated the new project will be opened in a new workspace. Please be ware that, when opening the adaption project inside a workspace with more projects than the adaption project some functional limitations may arise.

## Project Structure and Settings



The screenshot shows the SAP Studio IDE interface. On the left is the Explorer view, which lists the project structure under 'WORKSPACE.1 (WORKSPACE)'. Inside 'ZUX402ExtApp' are '.adp', 'node\_modules', 'webapp', and several configuration files: 'config.json', 'package-lock.json', 'package.json', 'ui5.yaml', and 'xs-app.json'. The 'config.json' file is currently open in the main editor area. Its content is as follows:

```

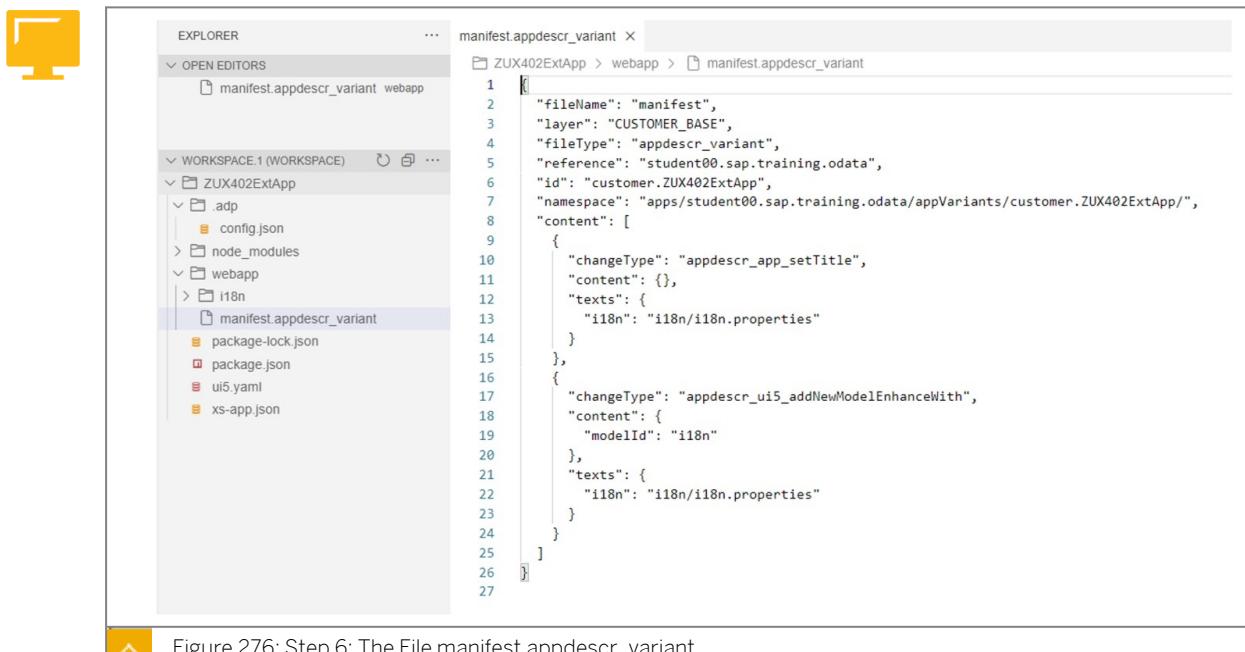
1 [
2 "componentname": "student00.sap.training.odata",
3 "safeMode": false,
4 "appvariant": "ZUX402ExtApp",
5 "layer": "CUSTOMER_BASE",
6 "sourceSystem": "sapS4",
7 "isOVPApp": false,
8 "isFioriElement": false,
9 "environment": "ABAP",
10 "ui5Version": "1.84.21"
11]
12

```

Figure 275: Step 5: Project Structure and Settings

The file *settings.json* in the folder *vscode* contains important information of the project. Here you are also able to change the SAPUI5 version that should be used to start the project in the Visual Editor preview if you want to test features in a higher SAPUI5-version.

## Step 6: The File manifest.appdescr\_variant



The screenshot shows the SAP Studio IDE interface. The Explorer view shows the project structure. In the 'webapp' folder, the 'manifest.appdescr\_variant' file is selected and open in the editor. Its content is as follows:

```

1 [
2 "fileName": "manifest",
3 "layer": "CUSTOMER_BASE",
4 "fileType": "appdescr_variant",
5 "reference": "student00.sap.training.odata",
6 "id": "customer.ZUX402ExtApp",
7 "namespace": "apps/student00.sap.training.odata/appVariants/customer.ZUX402ExtApp",
8 "content": [
9 {
10 "changeType": "appdescr_app_setTitle",
11 "content": {},
12 "texts": {
13 "i18n": "i18n/i18n.properties"
14 }
15 },
16 {
17 "changeType": "appdescr_ui5_addNewModelEnhanceWith",
18 "content": {
19 "modelId": "i18n"
20 },
21 "texts": {
22 "i18n": "i18n/i18n.properties"
23 }
24 }
25]
26]
27

```

Figure 276: Step 6: The File manifest.appdescr\_variant

The file *manifest.appdescr\_variant* is the manifest-file for the application variant, it contains the App id of the application variant. It also contains the configuration of the changes in section content. In the above figure you can see, that there is a model extension for the i18n-

model. This i18n-extension is generated by default and can be used to override Texts from the base application or add new i18n-texts.

### Step 7: Open SAPUI5 Visual Editor

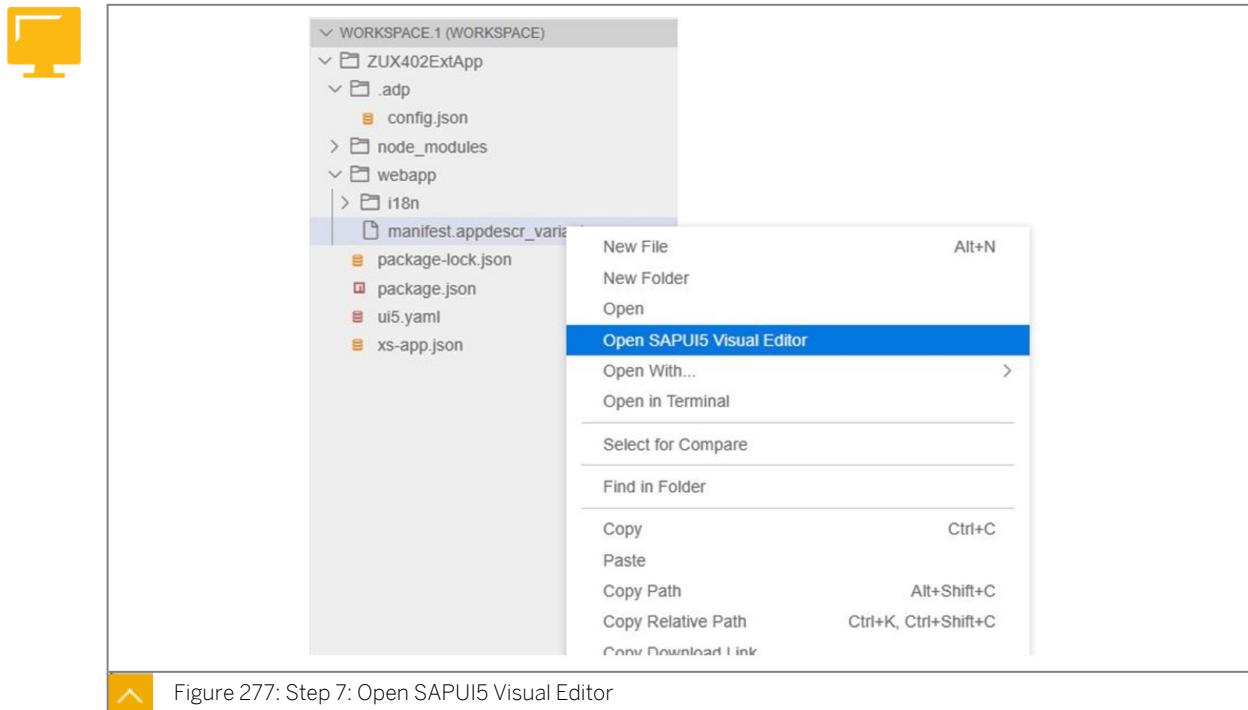


Figure 277: Step 7: Open SAPUI5 Visual Editor

To extend the base application by creating an application variant the SAPUI5 Visual Editor is used.

### Anatomy of an Extension Project

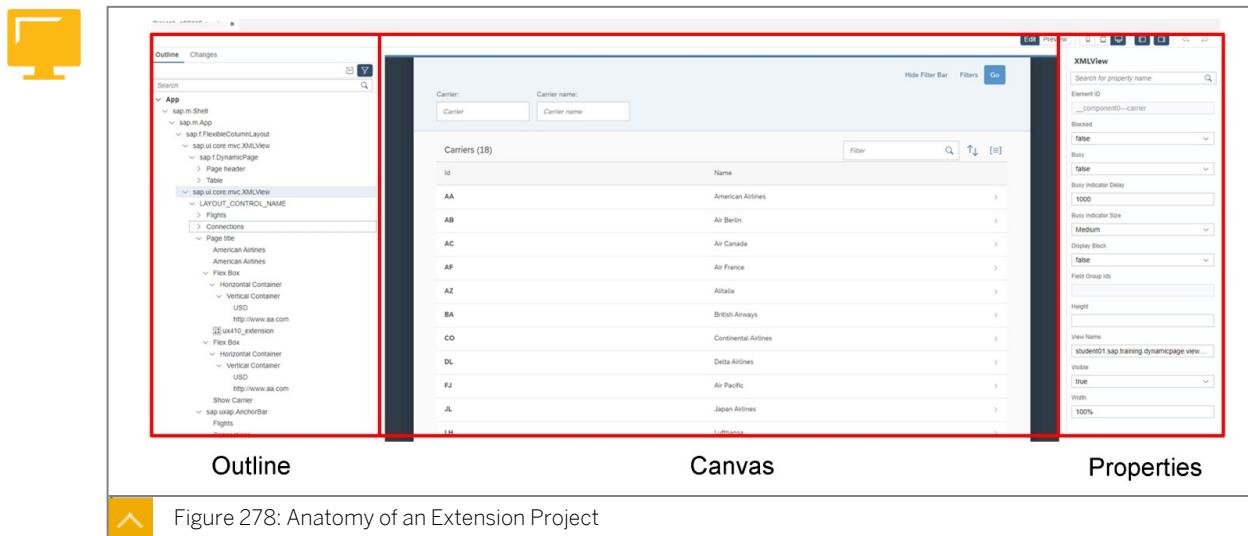


Figure 278: Anatomy of an Extension Project

As already mentioned in the class the application descriptor *manifest.json* is a concept introduced with SAPUI5 version 1.28. So this means, that it is possible that some older extensions are using the *Component.js* file to declare the extensions.

## Anatomy of the Component.js

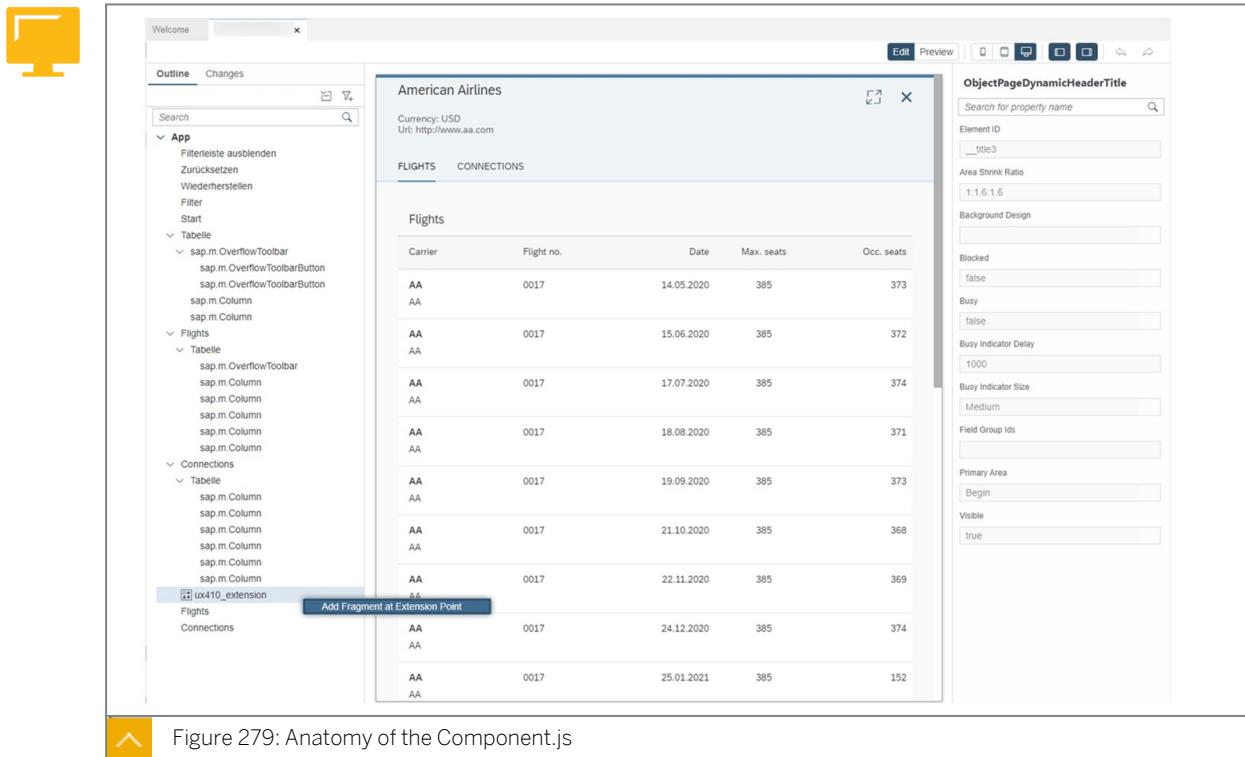


Figure 279: Anatomy of the Component.js

The figure shows an example of the *Component.js* file.

### Example: Implementing an Extension Point in the Component

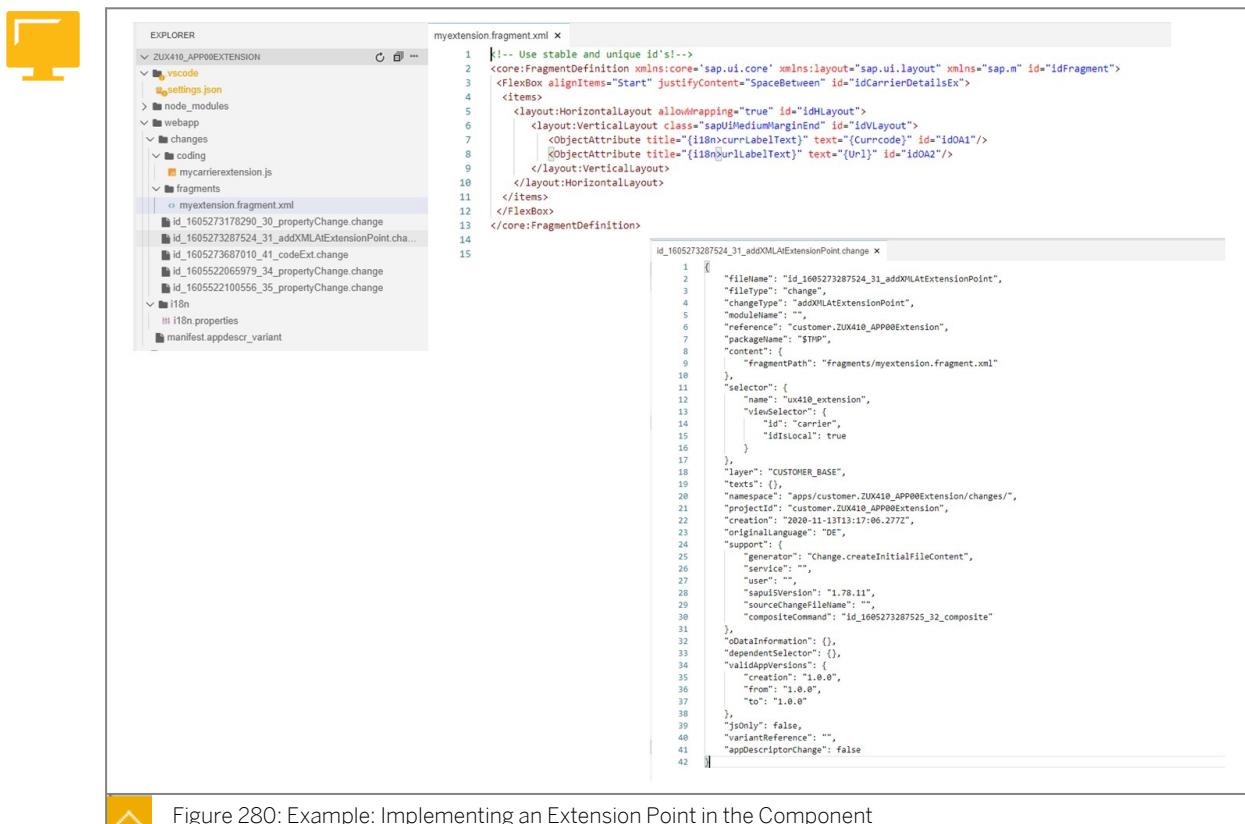


Figure 280: Example: Implementing an Extension Point in the Component

These are facts about the implementation point in the *Component.js* file.

- A view or a fragment can be used in a view extension.
- An extension is configured inside the *manifest.json* file.
- A customizing object is added under the *extends* object that is part of the *sap.ui5* object.
- In this case, the fragment named `Flights_carrierdetailsCustom` is being inserted at the *carrierdetails* extension point in the view named *Flights*.



## LESSON SUMMARY

You should now be able to:

- Explain Extension Capabilities in SAPUI5



## Describing Other Types of Extensibility in SAPUI5



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe other Types of Extensibility in SAPUI5 Application

### Other Types of Extensibility

#### What Other Types of Customizations are Possible in SAPUI5 Applications?

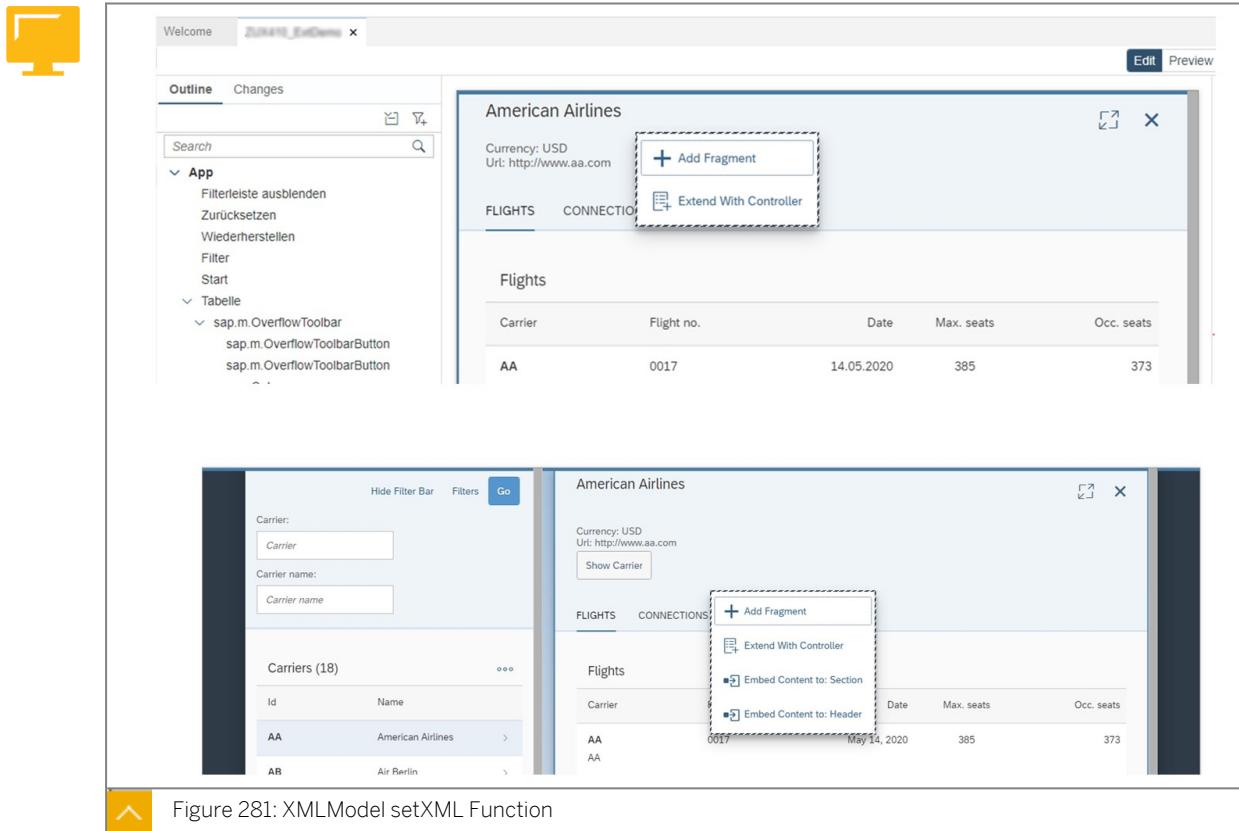


- Existing views can be totally replaced with customized views.
- Existing views can have specific properties of its controls be modified.
- Existing controllers can have code added.
  - Can also overwrite existing code.
- Customizations due to localization/internationalization.

You can adapt an SAPUI5 app to your specific requirements. For example, you can adapt or replace views, extend or replace controllers, or change language-specific texts.

The customization of SAPUI5 apps uses the component configuration: The customizing information is stored in a specific area of the component configuration. Such a customization can be performed on a custom app that extends a delivered standard app. A replacement or extension of views and custom controllers can also be part of a custom app, but may not always be required. If no replacement and no custom controller exists, the custom app project only contains the component definition with the customization configuration.

The standard app itself is not changed. The customized app becomes the start-up project and launches the standard app with the additional customizing configuration.



The above slide shows the currently available possible replacements.

### **View Replacements**

Customers are able to exchange complete views of a standard SAPUI5 application.

Therefore the developer of the standard SAPUI5 application do not implement any aspect in the application. Customers are free to exchange any view of an application.

The view that should replace a view of the standard application will be part of the extension project and is configured in the `sap.ui5` object of the `manifest.json` file of the extension project. The customizing object `sap.ui.viewReplacements` contains for each replacer view a configuration.

## Changing View Properties

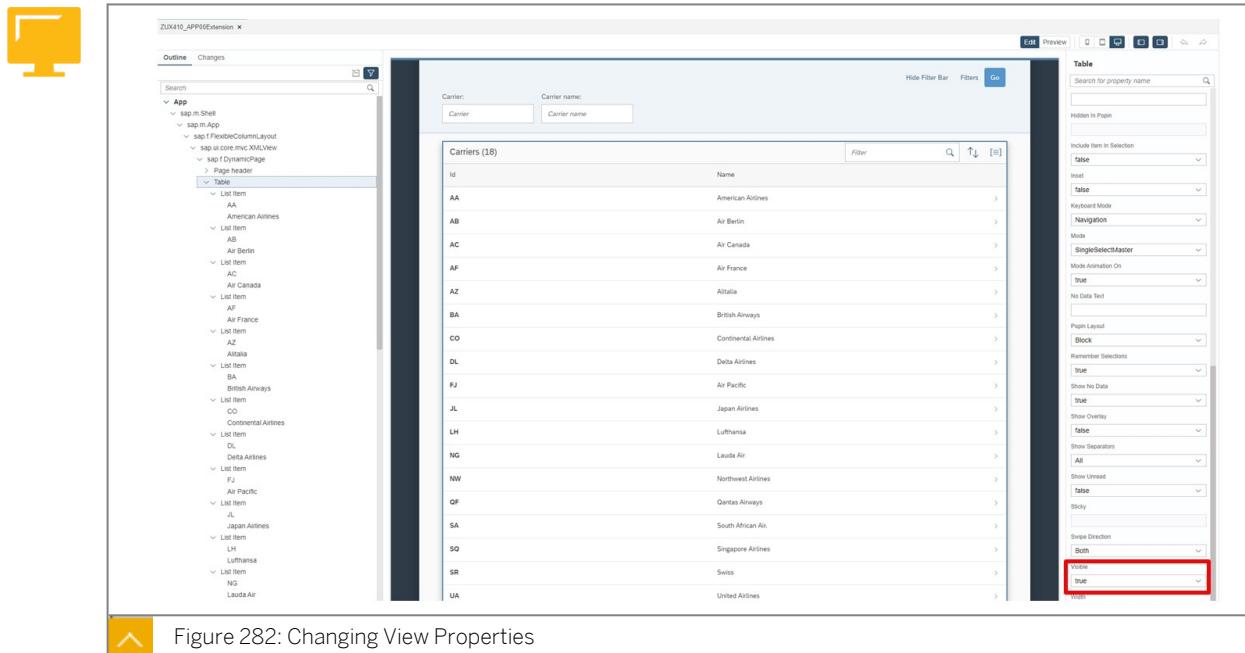


Figure 282: Changing View Properties

With a view modification customers are able to hide controls from the standard application using an extension.

The concept of view modification is based on the usage of ids. This means, only controls (including, for example, Layouts) with an id can be modified.

Currently the modification is only available using the visible-attribute of the control.

### Controller Extensions

The following are facts about controller extensions:



- Can extend or overwrite functionality in an existing controller with that of a new controller.
- If the new controller has methods with the same name as the existing controller, the new controller's methods will override those of the existing.
- Methods of the existing controller that were not overridden are still available for use.
- If the methods in the new controller do not match any of those in the existing controller, you have an extend situation.

SAPUI5 supports the extension of a base controller by merging the delivered standard controller with a custom controller on JavaScript object level.

## Example of Controller Extensions

 Extending the controller

Controller metadata in base application

```
Carrier.controller.js x
1 sap.ui.define([
2 "sap/ui/core/mvc/Controller"
3],
4 /**
5 * @param {typeof sap.ui.core.mvc.Controller} Controller
6 */
7 function (Controller) {
8 "use strict";
9
10 return Controller.extend("student01.sap.training.dynamicpage.controller.Carrier", {
11 metadata: {
12 methods: {
13 onShowCarrierData: {
14 public: true,
15 final: false
16 }
17 }
18 },
19 });
}
```

 Figure 283: Extending the Controller

If customers should be able to override the functionality of an controller the developer has to implement the controller facade using the metadata.

 Controller metadata in application variant

```
mycarrierextension.js x
1 /**
2 * @controller Name student01.sap.training.dynamicpage.controller.Carrier,
3 * @viewId __-component__-carrier
4 */
5 /**
6 * OpenSIS
7 * (c) Copyright 2000-2020 SAP SE or an SAP affiliate company.
8 * Licensed under the Apache License, Version 2.0 - see LICENSE.txt.
9 */
10
11 sap.ui.define([
12 "sap/ui/core/mvc/ControllerExtension",
13 // 'sap/ui/core/mvc/OverrideExecution'
14],
15 function (
16 ControllerExtension
17 // _OverrideExecution
18) {
19 "use strict";
20 return ControllerExtension.extend("ZUX410_APP900Extension.mycarrierextension", {
21 metadata: {
22 // extension can declare the public methods
23 // all general methods that start with "_" are private
24 methods: {
25 /**
26 publicMethod: {
27 public: true /*default*/,
28 final: false /*default*/
29 },
30 /**
31 finalPublicMethod: {
32 final: true
33 },
34 /**
35 onlyHook: {
36 public: true /*default*/,
37 final: false /*default*/
38 },
39 /**
40 couldBePrivate: {
41 public: false
42 }
43 }
44 }
45 });
46 },
```

 Figure 284: Controller Metadata in Application Variant

When adding a controller extension using the Visual Editor a new JS-file is generated and stored in the coding folder of the adaption project. The file contains the metadata information from the base controller.



Controller Extension

SAP Business Application Studio

```

mycarrierextension.js ×
1 /**
2 * Controller Name:student01.sap.training.dynamicspage.controller.Carrier,
3 * @sapddc_componentid--carrier
4 */
5
6 /* Opened */
7 // Copyright 2000-2009 SAP AG or an SAP affiliate company.
8 // Licensed under the Apache License, Version 2.0 - see LICENSE.txt.
9
10 sap.ui.define([
11 "sap/ui/core/mvc/ControllerExtension",
12 "sap/ui/core/mvc/OverrideExecution",
13 "sap/m/MessageBox"
14],
15 function (
16 ControllerExtension,
17 OverrideExecution,
18 MessageBox
19) {
20
21 "use strict";
22
23 return ControllerExtension.extend("ZUMAS_AP_PNNExtension.mycarrierextension", {
24
25 metadata: {
26 methods: {
27 onOpenCarrierData: {
28 public: true,
29 final: false,
30 overrideExecution: OverrideExecution.Instead
31 }
32 }
33 },
34
35 overrides: {
36 onOpenCarrierData: function() {
37 MessageBox.show(
38 "You are currently view the data of carrier " + this.getView().getProperty("Carriername"),
39 {
40 icon: MessageBox.Icon.INFO_MESSAGE,
41 title: "Carrier data"
42 }
43);
44 }
45 }
46 });
47 }

```

Figure 285: Controller Extension

In the adaption project the customer is able to override the functions of the controller from the base application. Using the `sap.ui.core.mvc.OverrideExecution` enumeration it is possible to describe if the function should be overwritten or enhanced.

## Controller Extensions and the LifeCycle Events

Controllers have the following lifecycle events:



- `onInit`
- `onBeforeRendering`
- `onAfterRendering`
- `onExit`

If you put the lifecycle events in the controller extension, here is what will happen:



- `onInit` and `onAfterRendering` in the extension are called after `onInit` and `onAfterRendering` of the standard/original controller.
- `onExit` and `onBeforeRendering` in the extension are called before `onExit` and `onBeforeRendering` of the standard/original controller.

The SAPUI5 controller extension concept does not use inheritance. Instead, methods of the custom controller override standard methods with the same name. The following controller lifecycle methods are, however, an exception to this rule: `onInit`, `onExit`, `onBeforeRendering`, `onAfterRendering`. For these methods, the controller methods of your custom application are called either after (for `onInit` and `onAfterRendering`), or before (for `onExit` and `onBeforeRendering`) the standard lifecycle methods.

### Providing Hooks in the Standard Controller



#### Note:

Extension points were only available for view code.

The following process is the equivalent for controller extension points:



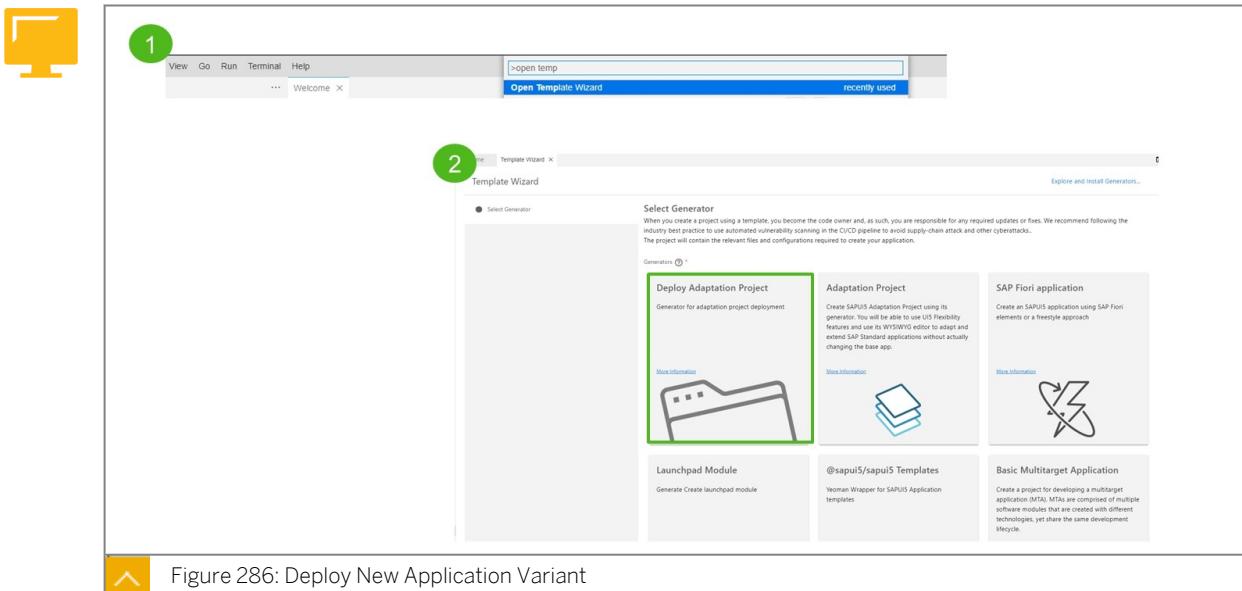
- Identify location in the standard controller where someone might want to add some customized code.
- Document a function to be used for the customized code:
  - Include arguments this function may receive or return.
  - In essence, this becomes a “reserved” function.
- Add code in the standard controller to check whether the function has been implemented, and if so, to call the function.
- Create the controller extension as shown earlier.
- Implement the “reserved” function in the extension.
- Perform the component customizing for the controller extension as shown earlier.

Hooks are extension points in the controller code that are used to make controller extensions more stable.

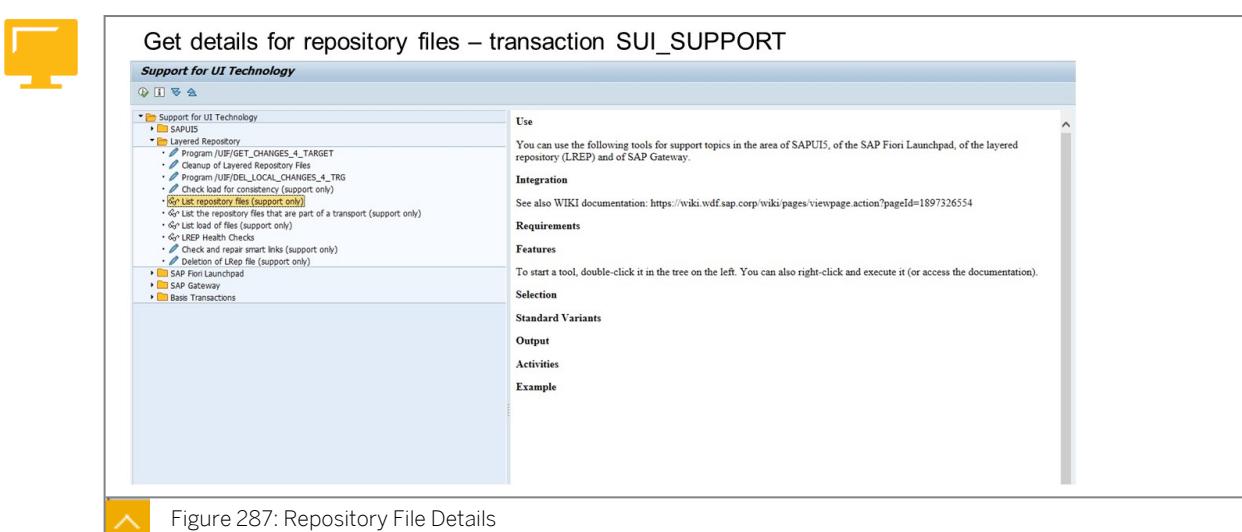
The controller extension concept enables you to override any method. This is a powerful but also fragile feature. Extension points, so-called hooks, can be provided in the controller code. These hooks can be documented and kept stable, thus providing more robust hooks across application updates for controller extensions.

The process for this is as follows:

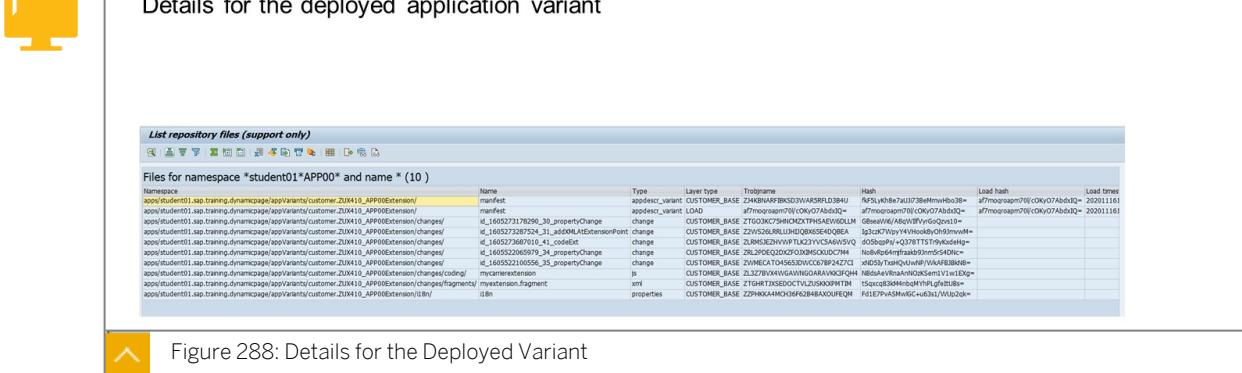
1. In the application, identify a strategic location within the controller code where customers may want to plug in and execute their customized code.
2. In the application, define a new function name which is reserved for the extension, document the function and any arguments the function may receive or return.
3. Add code lines in the application (see code snippet below) to check whether the function has been implemented, and, if so, to call the function. We also recommend to implement sanity checks for return values.
4. The customer can then configure a controller extension, implementing exactly this one function.
5. SAPUI5 runtime merges the new controller extension into the standard controller. If customizing is enabled, the new function can be executed.



Use the yo tool to deploy a new application variant use as namespace parameter the namespace specified in the file `manifest.appdescr` variant of your adaption project.



Details for other students can be found in the student section.



Having this detail the administrator can now create a new target mapping to the application variant.

If you want to hide the base application in general add the URL parameter `sap-appvar-id` with the namespace of your application variant to the target mapping of the base application.



### LESSON SUMMARY

You should now be able to:

- Describe other Types of Extensibility in SAPUI5 Application

## Learning Assessment

1. What is meant by the phrase “SAPUI5 supports modification free enhancements”?

*Choose the correct answer.*

- A The developer must create a copy of the application that should be enhanced and the enhancement is done in the copy.
- B The delivered standard application remains unchanged and hence the extension is considered to be modification free.
- C SAP provides a service in the cloud to generate an enhanced application using aspect-oriented programming.

2. Is it possible to add an extension point in the component implementation?

*Choose the correct answer.*

- A Yes
- B No
- C It depends on the type of component.

3. To what namespace is the `ExtensionPoint` class assigned?

*Choose the correct answer.*

- A `sap.ui.extension`
- B `sap.m`
- C `sap.ui.core`
- D `sap.ui.core.extension`

4. What types of extensions/replacements are supported by SAPUI5?

*Choose the correct answers.*

- A Component replacement
- B View replacement
- C View modification
- D Replace service
- E Implement UI Controller Hooks
- F Manifest.json replacement

5. When you implement a controller extension and you implement the `onInit` and `onAfterRendering` functions in the extension, when are they called?

*Choose the correct answer.*

- A After the corresponding functions from the standard controller.
- B Only the functions of the controller extensions are called.
- C Before the corresponding functions from the standard controller.

6. How is the extension type called to hide UI controls?

*Choose the correct answer.*

- A Control replacement
- B Control modification
- C View modification
- D Element modification

## Learning Assessment - Answers

1. What is meant by the phrase “SAPUI5 supports modification free enhancements”?

*Choose the correct answer.*

- A The developer must create a copy of the application that should be enhanced and the enhancement is done in the copy.
- B The delivered standard application remains unchanged and hence the extension is considered to be modification free.
- C SAP provides a service in the cloud to generate an enhanced application using aspect-oriented programming.

Correct. The enhancement concept of SAPUI5 makes it possible to extend delivered standard applications without changing the code of the original application.

2. Is it possible to add an extension point in the component implementation?

*Choose the correct answer.*

- A Yes
- B No
- C It depends on the type of component.

Correct. It is not possible to add an extension point inside the component implementation. Extension points are only supported in views.

3. To what namespace is the `ExtensionPoint` class assigned?

*Choose the correct answer.*

- A `sap.ui.extension`
- B `sap.m`
- C `sap.ui.core`
- D `sap.ui.core.extension`

Correct. The extension point is part of the `sap.ui.core` namespace.

4. What types of extensions/replacements are supported by SAPUI5?

*Choose the correct answers.*

- A Component replacement
- B View replacement
- C View modification
- D Replace service
- E Implement UI Controller Hooks
- F Manifest.json replacement

Correct. SAPUI5 supports view replacement, view modification, the implementation of UI controller hooks, and the replacement of service.

5. When you implement a controller extension and you implement the `onInit` and `onAfterRendering` functions in the extension, when are they called?

*Choose the correct answer.*

- A After the corresponding functions from the standard controller.
- B Only the functions of the controller extensions are called.
- C Before the corresponding functions from the standard controller.

Correct. The extension `onInit` and `onAfterRendering` functions are called before the corresponding functions from the standard controller.

6. How is the extension type called to hide UI controls?

*Choose the correct answer.*

- A Control replacement
- B Control modification
- C View modification
- D Element modification

Correct. The extension type is called to hide UI controls by view modification.

# UNIT 5

# Version Control - Working in Teams

## Lesson 1

Working with GIT

297

## Lesson 2

Working with GIT Repositories

305

## Lesson 3

Working with Branches

313

## UNIT OBJECTIVES

- Understand GIT and how it is used
- Use GIT repositories
- Use GIT branches



# Unit 5

## Lesson 1

# Working with GIT



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand GIT and how it is used

### GIT Version Control System

The following figure summarizes the features of the GIT version control system.



- GIT is a free and open-source distributed version control system.
- It is designed to handle small to very large projects.
- It is designed for speed and efficiency.
- Supports local branching and staging.

### Non-linear development

- GIT-Client is built into the SAP Business Application Studio.
- For more information, visit <https://git-scm.com/>



Figure 289: GIT High-level Features

The fundamentals of GIT are shown in the following figure.



- GIT works like a set of snapshots.
- With every commit, GIT takes a snapshot of the current state of the underlying files.
- For unchanged files, only a reference to the file is stored.
- Most operations require local files and resources only.
- Almost every operation can be done offline.
- GIT uses checksums based on SHA-1 hashes.

#### 40-character hex string



Figure 290: Git Basics



#### In GIT, a file can reside in one of three main states:

- **Committed**  
The data is safely stored in the local database.
- **Modified**  
Files were changed but not yet committed to the local database.
- **Changed**  
Files were marked as modified in the current version and go into the next commit snapshot.



Figure 291: GIT States



The basic GIT workflow is shown in the following figure.

1. **Modify files in the working tree.**
2. **Stage the files / add snapshots of files to staging areas.**
3. **Perform a Commit.**  
Takes the files from the staging area and stores the snapshot permanently in the local GIT repository.



Figure 292: GIT Basic Workflow

The following figure shows the possibilities for installing GIT. GIT is integrated into SAP Business Application Studio, therefore no installation is required.



- **Installing GIT is a prerequisite when using GIT from a local machine.**
- **GIT is available for:**
  - \_ Linux
  - \_ Solaris
  - \_ Windows
  - \_ Mac
- **It can be downloaded from: <https://git-scm.com/downloads>**
- **It is already available in the SAP BAS, therefore no installation is required.**

Figure 293: GIT Installation

The procedures for getting started with GIT repositories are provided in the following figure.



- 1. Import an existing directory into GIT.**
  - a) Create / Switch to the project directory (cd /yourprojectdirectory)
  - b) Initialize the Repository (git init)
  - c) Stage all files (git add -A)
  - d) Commit staged files (git commit –m 'Message')
- 2. Clone an existing GIT repository from a server.**
  - a) Create / Switch to the project directory (cd /yourprojectdirectory)
  - b) Clone the directory (git clone https://github.com/SAP/hana-shine)

Figure 294: GIT Repositories – Getting Started

The following figure provides an overview of files and working directories in GIT.



- **Each file in the working directory can have the following states**
  - **Tracked**  
All files that were part of the last snapshot.  
Tracked files can be in the following stages:
    - Unmodified
    - modified
    - Staged
  - **Untracked**  
All files in the working directory that were not part of the last snapshot and are not in the staging area.
- **When a remote repository is cloned, all files are in the tracked state and unmodified.**

Figure 295: GIT Working Directories

The concept of a Staging Area is described in the following figure.



- One of Git's more unique features.
- Lets you group related changes into highly focused snapshots before committing the snapshot.
- Staging commands:
  - Single File: `git add <Filename>`
  - Single Directory: `git add <Directory>`
  - Interactive: `git add -p`



Figure 296: GIT Staging Area

The following figure shows the concept of a commit operation in GIT.



- Commits the staged snapshots to the project history.
- Committed snapshots are safe versions of a project.
- Snapshots are committed to the local repository.
- Committing command:  
`git commit -m "Commit Message"`



Figure 297: GIT Commit

The concept of checkout in GIT is shown in the following figure.



- Three distinct functions:

- Checking out Files
- Checking out Commits
- Checking out Branches

- Usage:

- Branch Checkout: `git checkout master`
- Commit Checkout: `git checkout <commit>`
- File Checkout: `git checkout <commit> <file>`
- See all commits: `git log --oneline`

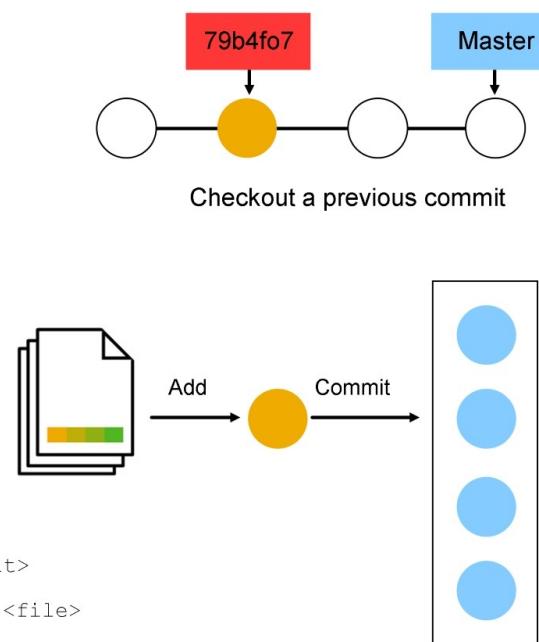


Figure 298: GIT Checkout

## GIT and SAP Business Application Studio

The integration of GIT in the SAP Business Application Studio is shown in the following figure.

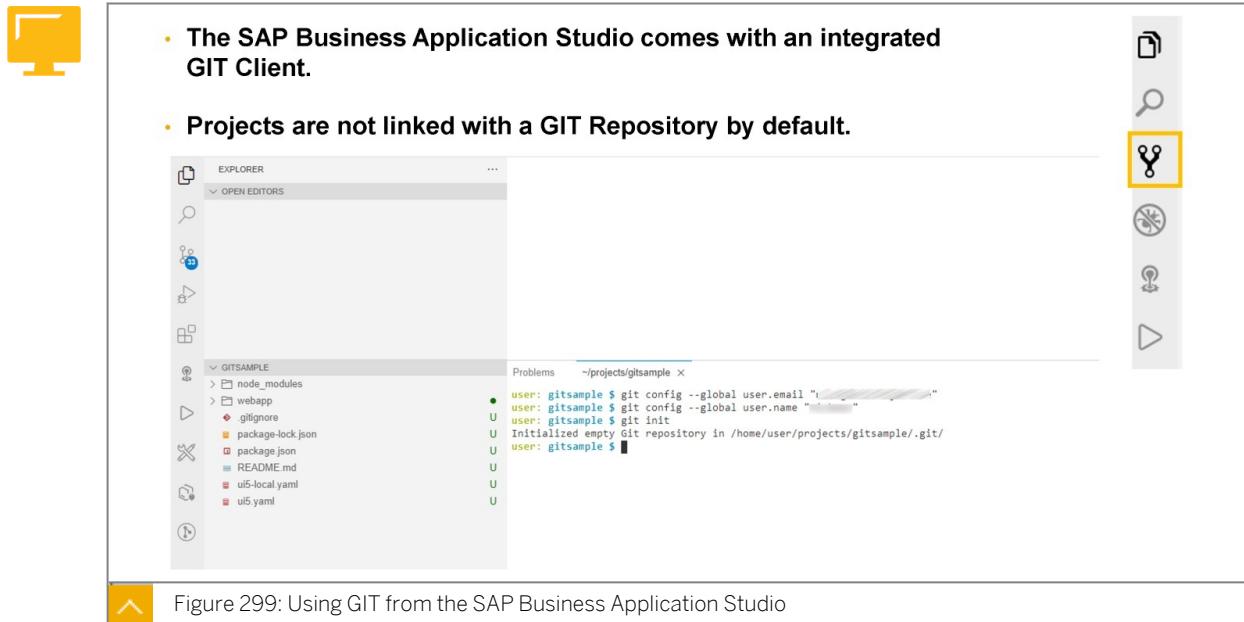


Figure 299: Using GIT from the SAP Business Application Studio

The following figure shows the GIT workflow in the SAP Business Application Studio.

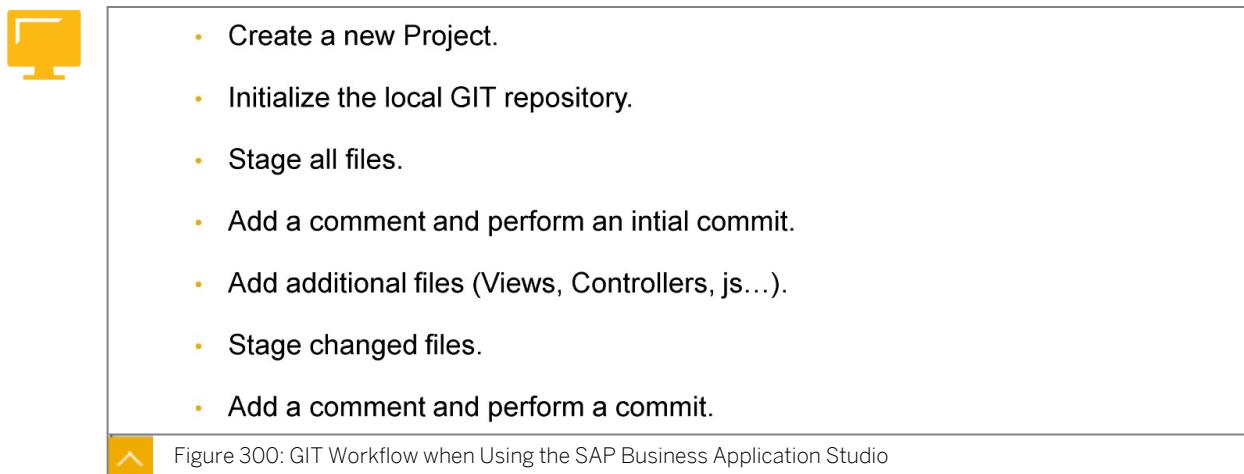


Figure 300: GIT Workflow when Using the SAP Business Application Studio

The use of GIT branches in the SAP Business Application Studio is shown in the following figure.

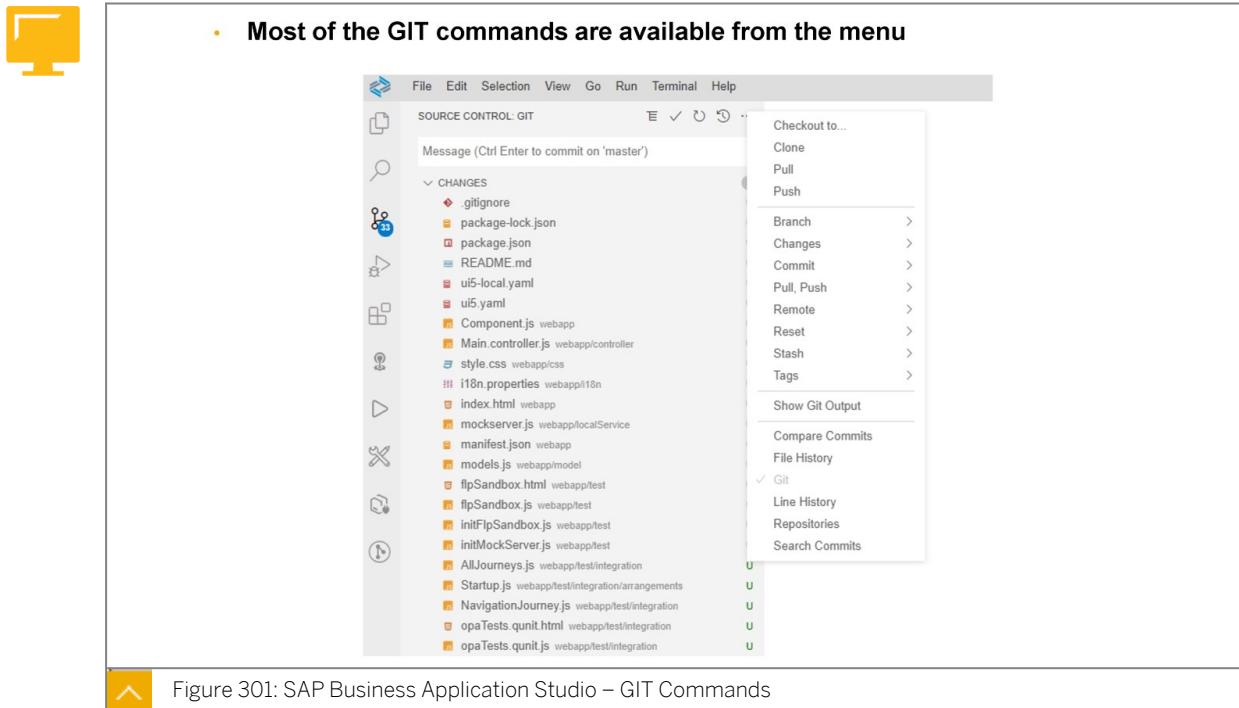


Figure 301: SAP Business Application Studio – GIT Commands

The following figure shows how GIT staging is represented in SAP Business Application Studio.

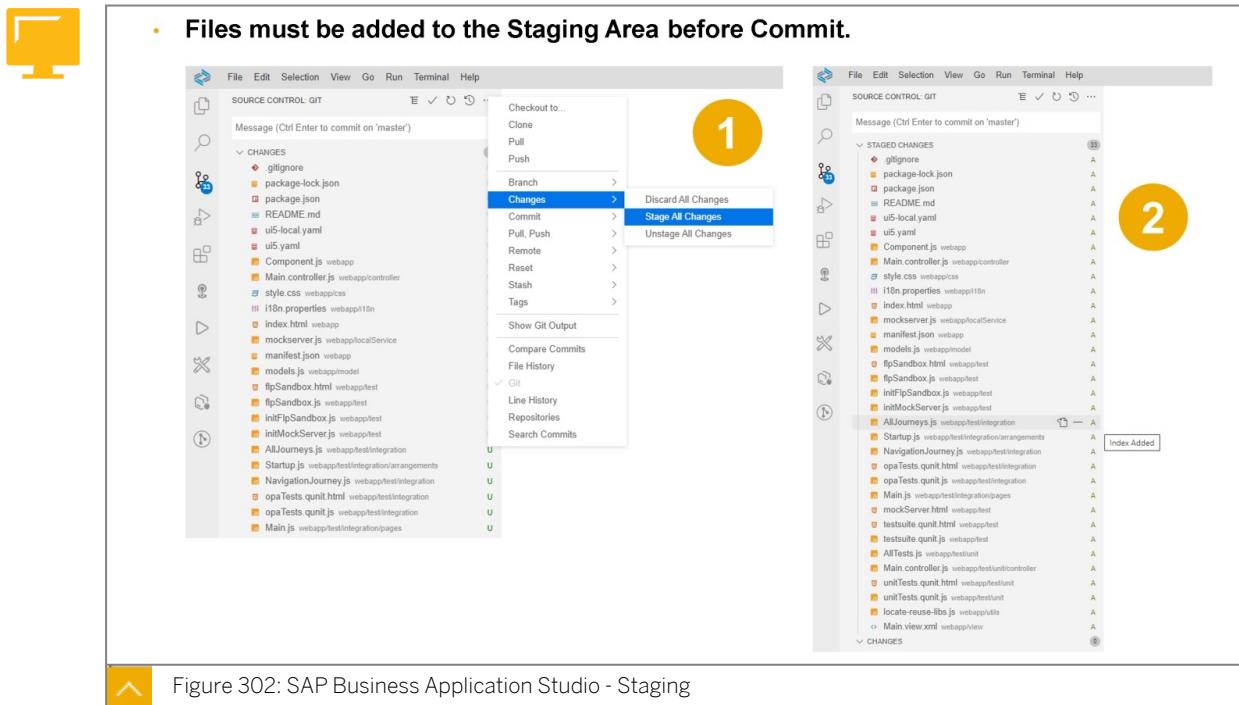


Figure 302: SAP Business Application Studio - Staging

Modified files are marked with the letter M. To add changes to the staging area, use the menu to add all changed files to the staging area or the plus symbol at file level.

The following figure shows how to add a comment when performing a GIT commit in SAP Business Application Studio.

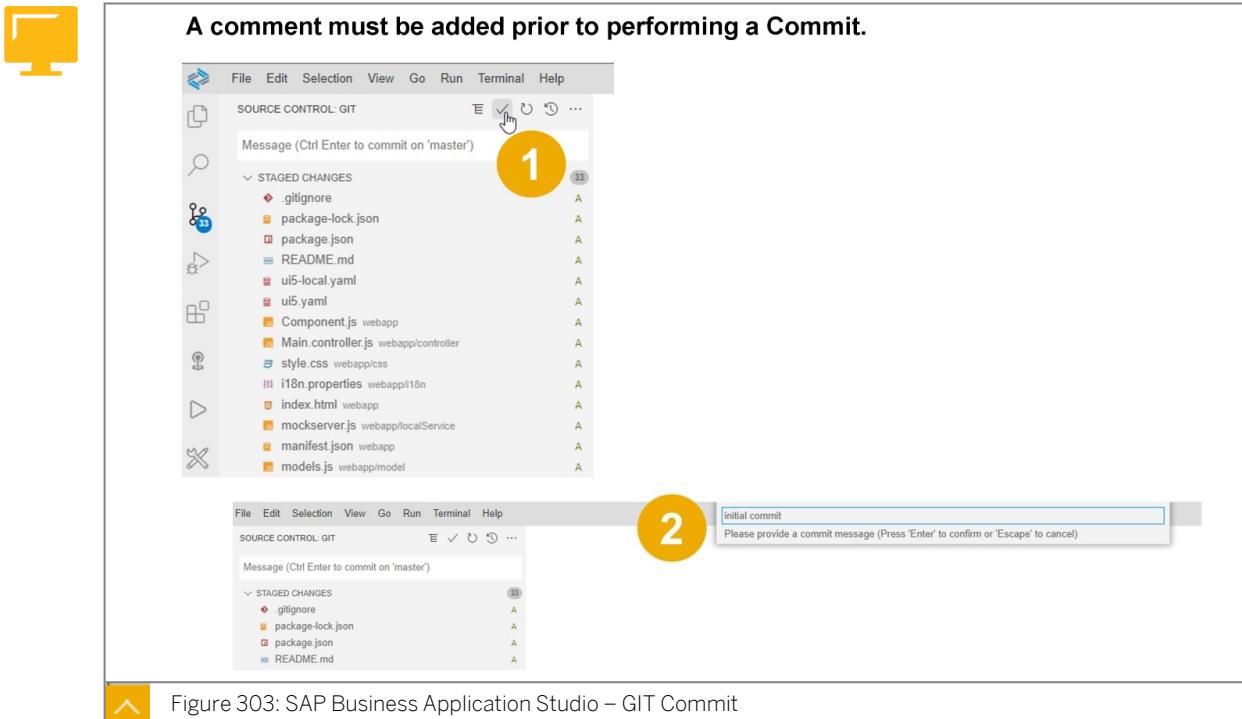


Figure 303: SAP Business Application Studio – GIT Commit

The following figure shows how the “New” file state is represented in the SAP Business Application Studio. A plus symbol is shown in front of the file name.

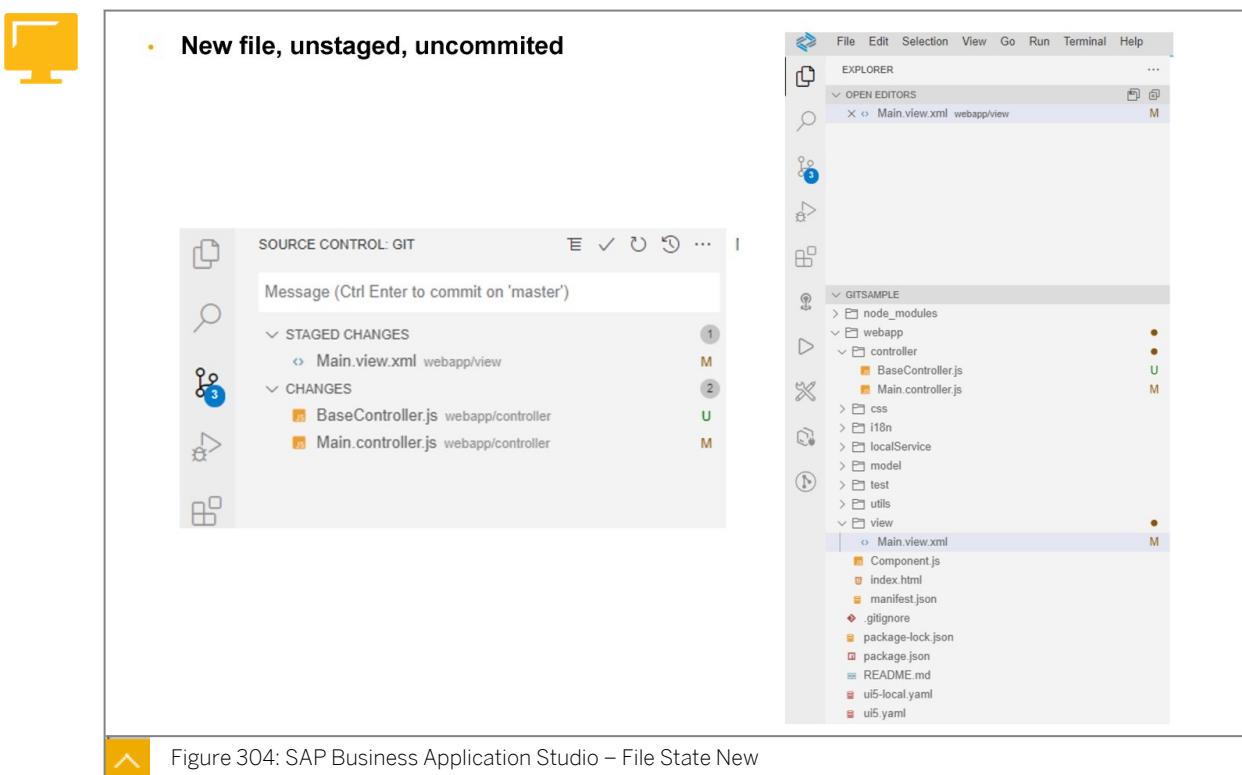
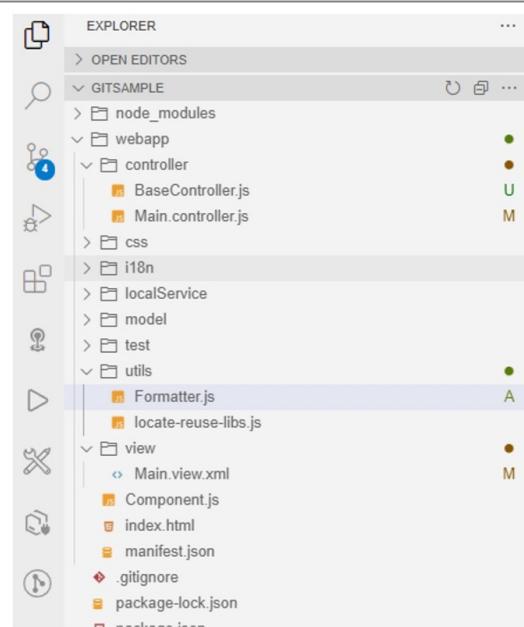


Figure 304: SAP Business Application Studio – File State New

The following figure shows how the “Staged” file state is represented in the SAP Business Application Studio. After staging new files are marked with A, modified files are marked with M. If a file is deleted from the project it is marked with D.



The screenshot shows the SAP Business Application Studio's Explorer view. The tree structure displays a project named 'GITSAMPLE' with various folders and files. The 'utils' folder is currently selected. The right side of the interface shows the status of each file: some are marked with green dots (green dot), some with orange dots (orange dot), and some with a combination of both (green dot with orange dot). A specific file, 'Formatter.js', is highlighted with a blue selection bar at the bottom of the list.

Figure 305: SAP Business Application Studio – File State Staged



## LESSON SUMMARY

You should now be able to:

- Understand GIT and how it is used

## Working with GIT Repositories



### LESSON OBJECTIVES

After completing this lesson, you will be able to:

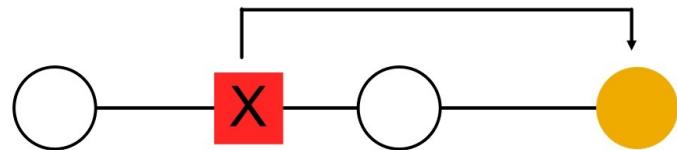
- Use GIT repositories

### Remote GIT Repositories

The following figure shows the possibility to revert to a previous state in GIT branch, undoing a committed snapshot in the process.



- **Undo a committed snapshot.**
- **Commit is not removed from the project history.**
  - GIT determines how to revert the changes.
  - GIT appends a new commit.
  - The history is never lost.
- **Usage:**
  - `git revert <commit>`



Revert a previous commit



Figure 306: GIT Revert

You can also revert back to the previous state of a project as shown in the following figure.



- Reverts back to the previous state of the project.
- Removes all subsequent commits (changes the history of the project).
- Works backwards from the current commit only.
- Usage:
  - git reset
  - git reset <file>

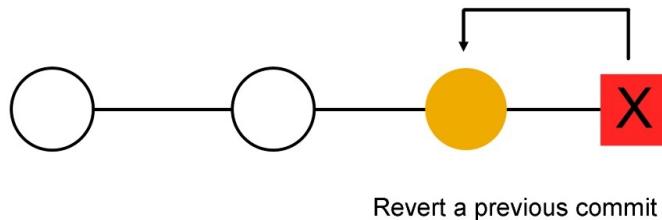


Figure 307: GIT Reset

The following figure summarizes the GIT Clean command that is used to remove all untracked files from the working directory.



- Removes all untracked files from the working directory.
- Undo is not possible.
  - A dry run can be done.
- Usage:
  - git clean
  - Dry run: git clean -n
  - Forced clean: git clean -f

Figure 308: GIT Clean

## Other GIT Features

It is also possible to do a fast-forward merge in GIT as explained in the following figure.



- Possible if a linear path from the current branch to the target branch exists.
- GIT always tries a fast-forward merge first.
- Typically used for bug fixes or minor features.
- Not possible if the branches have diverged.
  - A three-way merge with a dedicated commit is required.
- You can suppress fast-forward merge.
  - git merge --no-ff <branch>

Figure 309: GIT Fast Forward Merge

Conflicts may prevent a GIT merge and they must be resolved manually as indicated in the following figure.



- **A merge is not possible if the same file was changed in different branches.**
- **Conflicts must be resolved manually:**
  - Using the edit / stage / commit workflow:
    - Find the affected files first.
  - Using git status:
    - Perform manual changes on the affected files.
    - Run git add on the conflicted files.
    - Run git commit.

Figure 310: GIT Conflict Resolution

The following figure shows the use of tagging in GIT.



- **GIT supports the tagging of specific points in history.**
- **Tags are not pushed to the remote repositories by default using git push.**
- **Usage:**
  - List all tags: `git tag`
  - Create a tag: `git tag -a v1.5 -m "UX402 V 1.5"`
  - Show a specific tag: `git show <version>`, for example, `git show v1.5`
  - Push tags to a remote server: `git push origin <tagname>`
  - Checkout a tag: `git checkout <tagname>`, for example, `git checkout v1.5`

Figure 311: GIT Tagging

The following figure shows some commands for working with remote GIT repositories.



- **Git clone adds the origin shortname reference to your repository.**
- **Usage:**
  - Manually add a repository: `git remote add <remote-name> <url>`
  - Fetch information from the repository: `git fetch <remote-name>`
  - Push to the remote branch: `git push <remote-name> <branch-name>`
  - Information about the remote: `git remote show <remote-name>`
  - Remove the remote: `git remote remove <remote-name>`

Figure 312: Working with Remote GIT Repositories

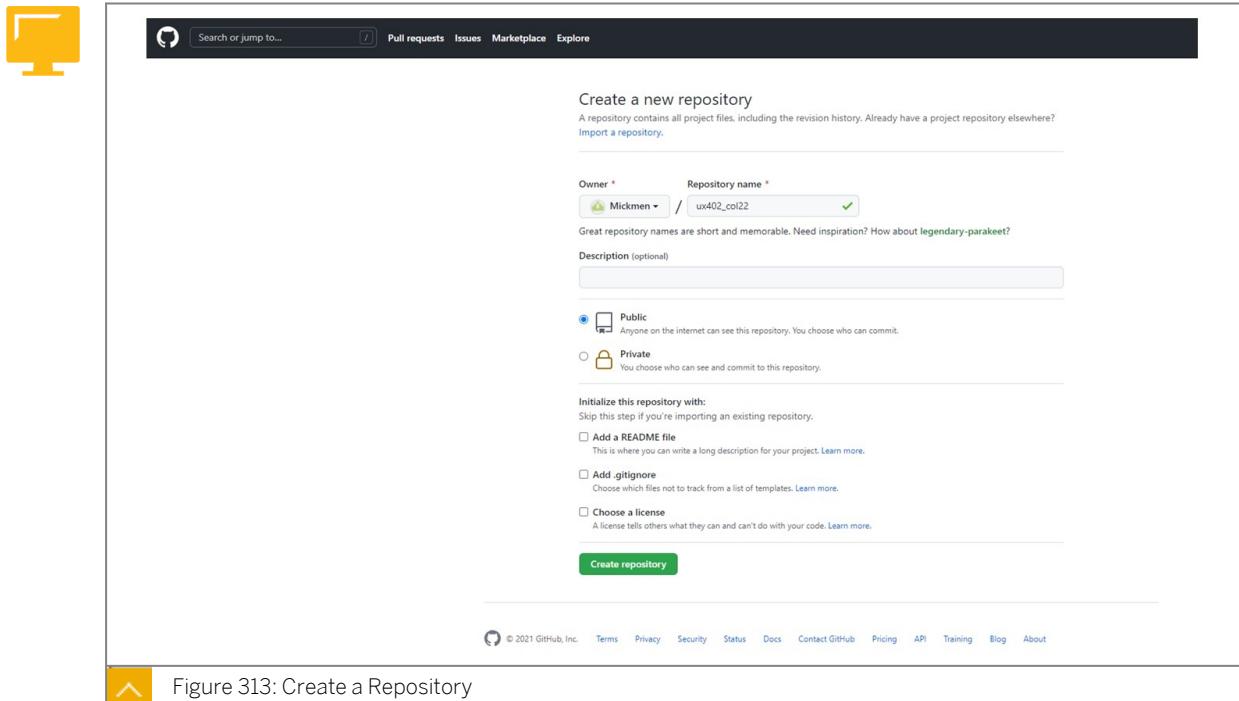


Figure 313: Create a Repository

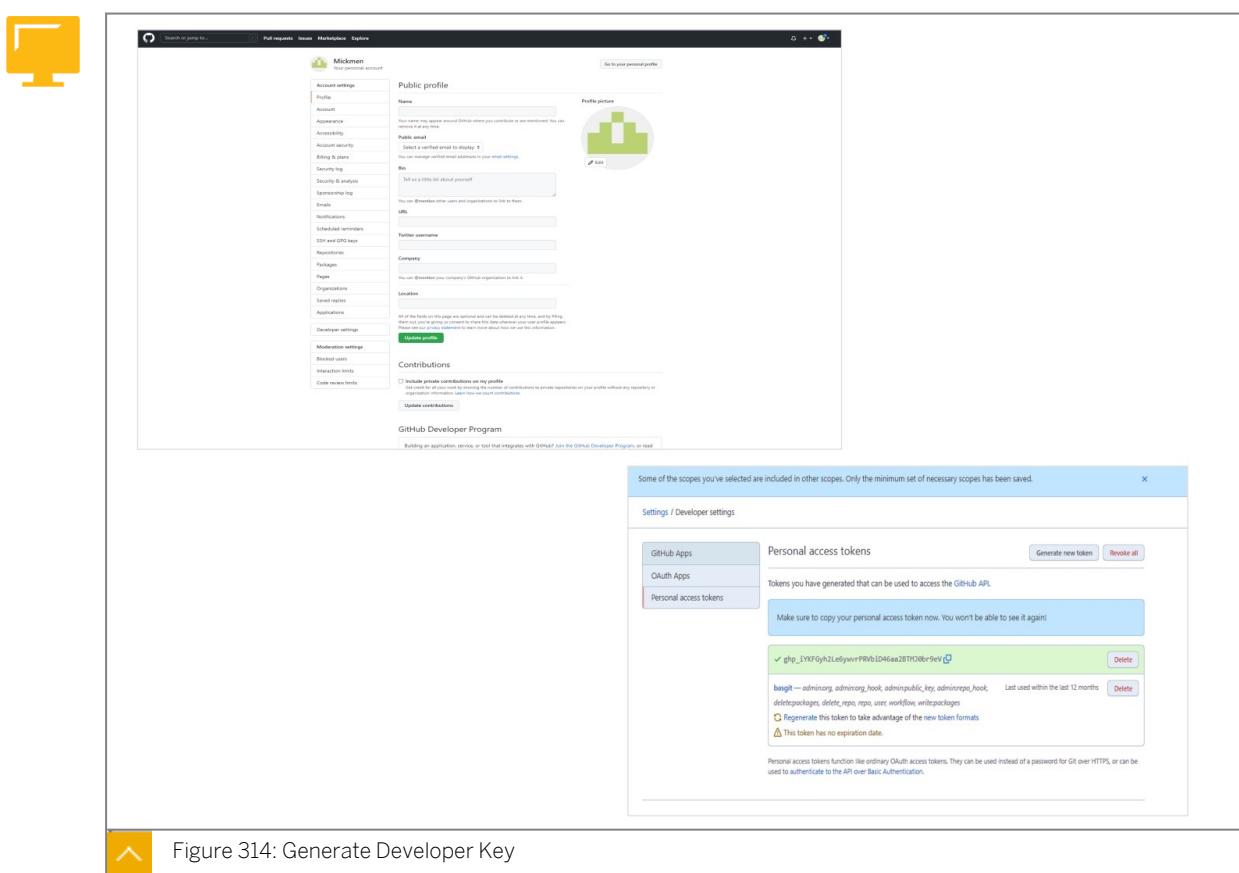


Figure 314: Generate Developer Key

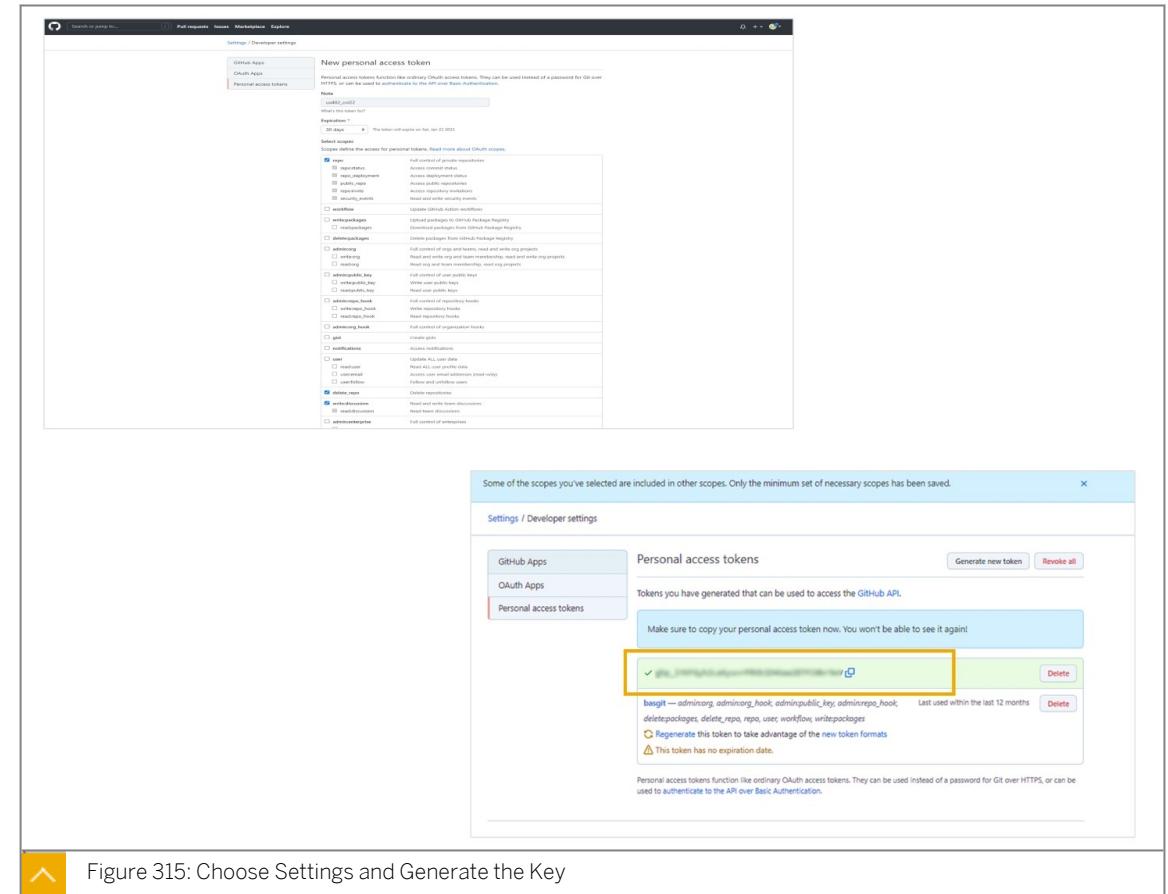


Figure 315: Choose Settings and Generate the Key

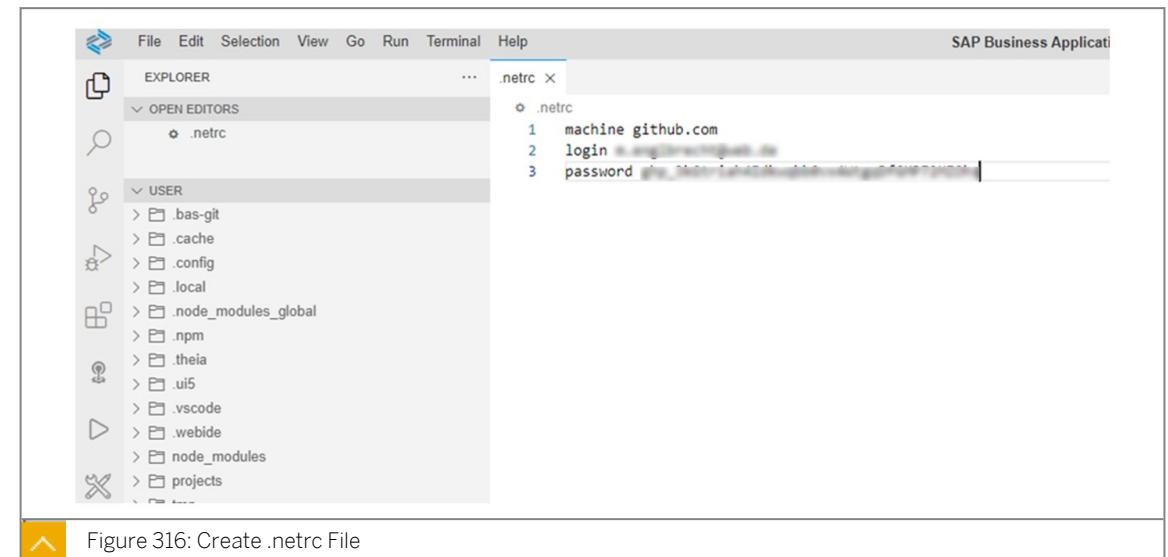


Figure 316: Create .netrc File

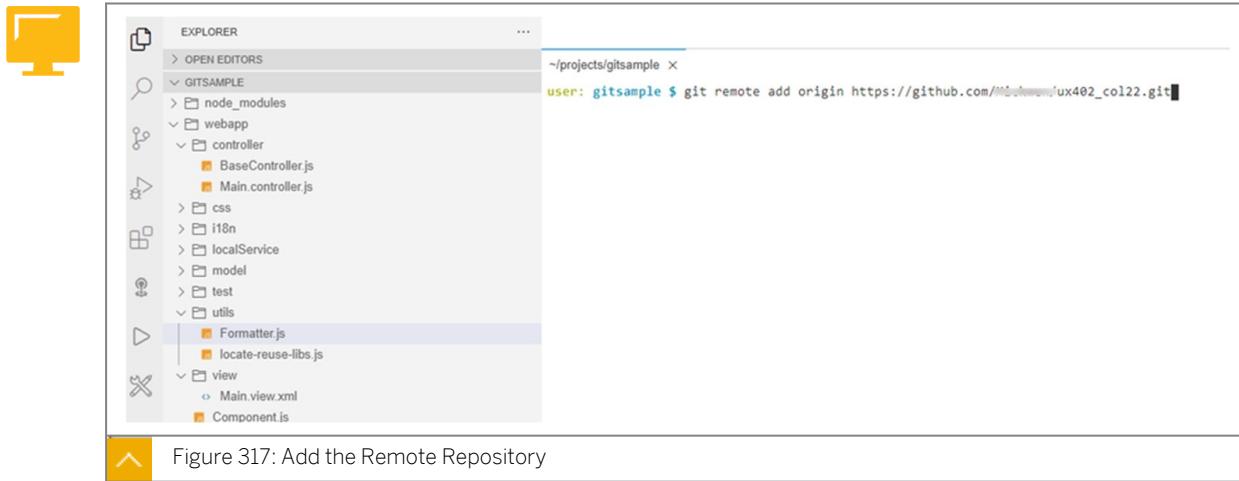


Figure 317: Add the Remote Repository

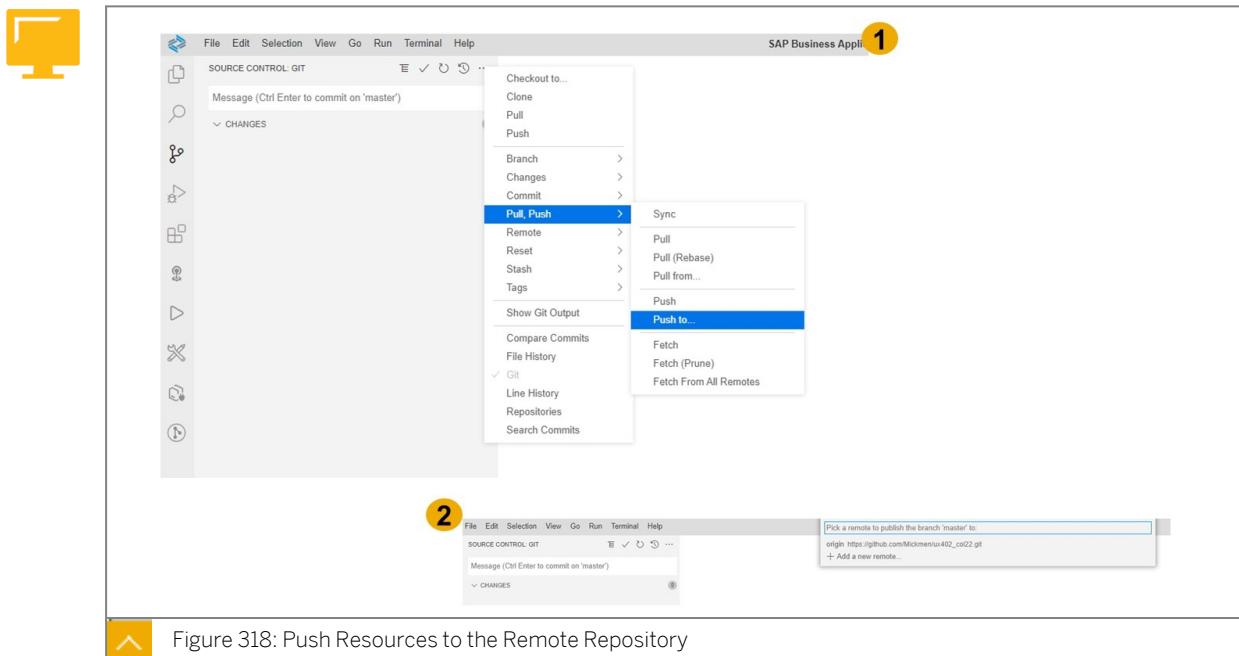


Figure 318: Push Resources to the Remote Repository

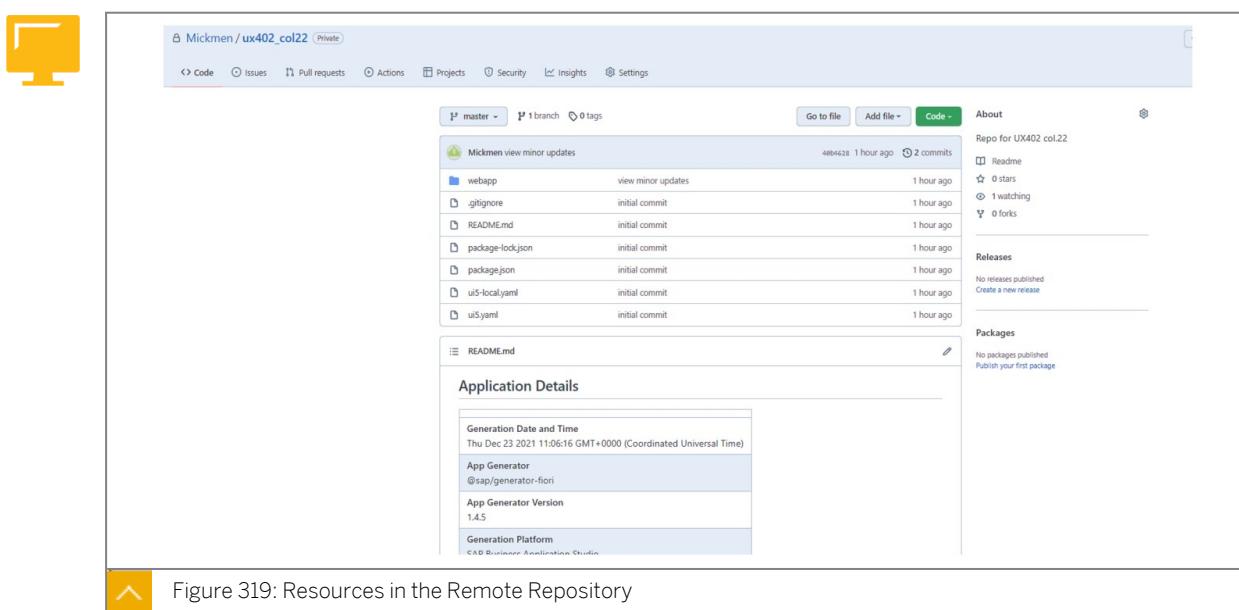


Figure 319: Resources in the Remote Repository



## LESSON SUMMARY

You should now be able to:

- Use GIT repositories



# Working with Branches



## LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use GIT branches

## Branches in GIT

The following figure explains what branches are and their usage in GIT.

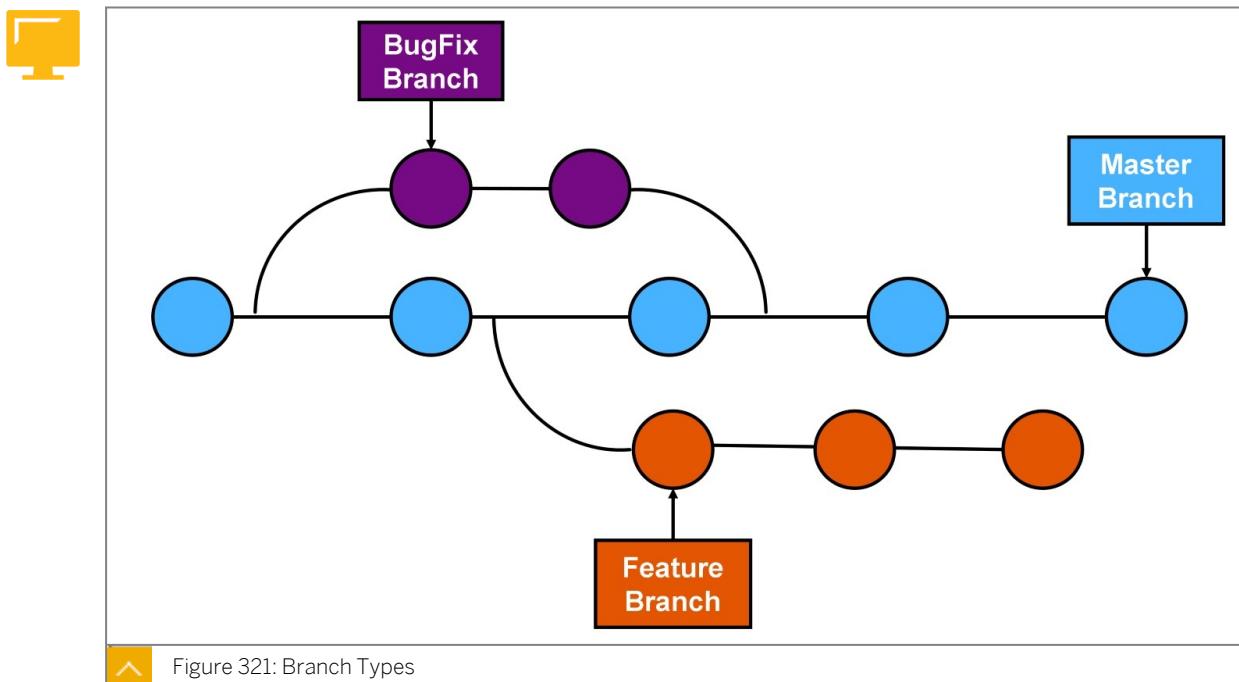


- **Represents an independent line of development.**
- **Serves as an abstraction for the staging/committing process.**
- **Works with an independent working directory and staging area.**
- **A branch is a reference to a commit.**
- **Usage:**
  - List all branches in the repository: `git branch`
  - Create a new branch: `git branch <branch name>`
  - Delete a branch: `git branch -d <branch name>`

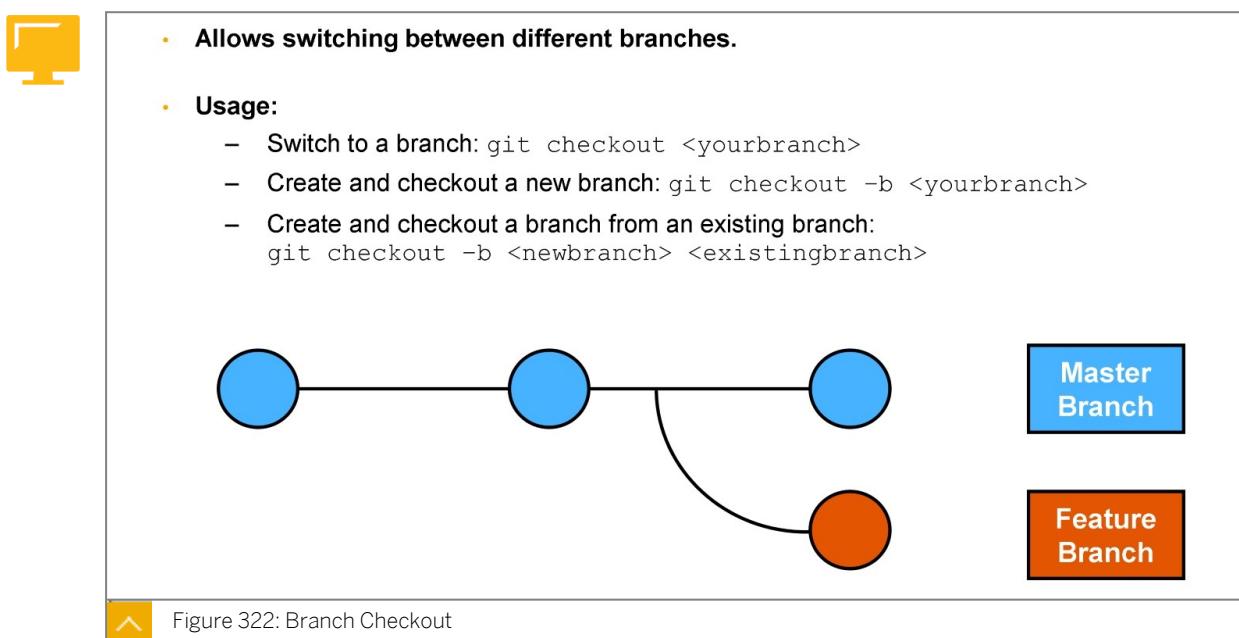


Figure 320: Branches Explained

Different branch types are used to represent different development activities as indicated by the example branches in the following figure.



The following figure shows how a developer can switch focus by checking out a different branch.



A branch that was forked from the master branch can later be merged back into the master branch using a GIT merge as shown in the following figure.



- Allows the integration of one branch into another.
- Usage:
  - Merge a branch into the current branch: `git merge <yourbranch>`
  - Create and checkout a new branch: `git checkout -b <yourbranch>`
  - Create and checkout a branch from an existing branch: `git checkout -b <newbranch> <existingbranch>`

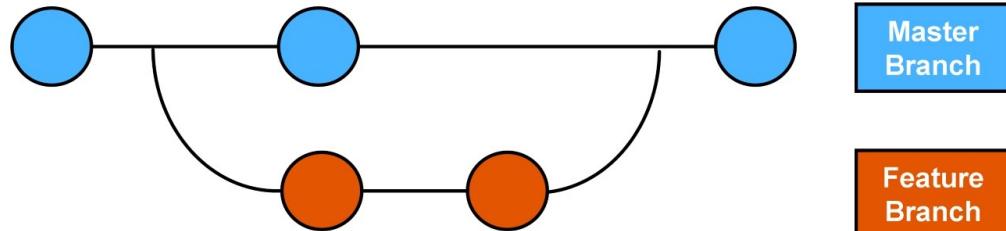


Figure 323: GIT Merge

### GIT Branches and SAP Business Application Studio

You can create and work with GIT branches from SAP Business Application Studio as shown in the following figure.



- Creating a new branch from SAP BAS statements is required.

```
gttkh-deployment-ffff64ccb4-jl28b: ~/projects/sampleproject Deb
user: test $ git branch testbranch
```

- To merge, checkout etc. is done using statements

Figure 324: Creating a GIT Branch from SAP Business Application Studio

In the SAP Business Application Studio, you can also merge GIT branches back into the master branch as shown in the following figure.

The following figure shows the GIT workflow basics when working in the SAP Business Application Studio.



- Create a branch (feature branch).
- Add new functionality (Views, Controllers, js...).
- Stage files.
- Test your new functionality.
- Add a comment and perform an commit.
- Switch to the master branch.
- Merge the feature branch into the master branch.



Figure 325: Workflow Basics When Using GIT Branches from within the SAP Business Application Studio

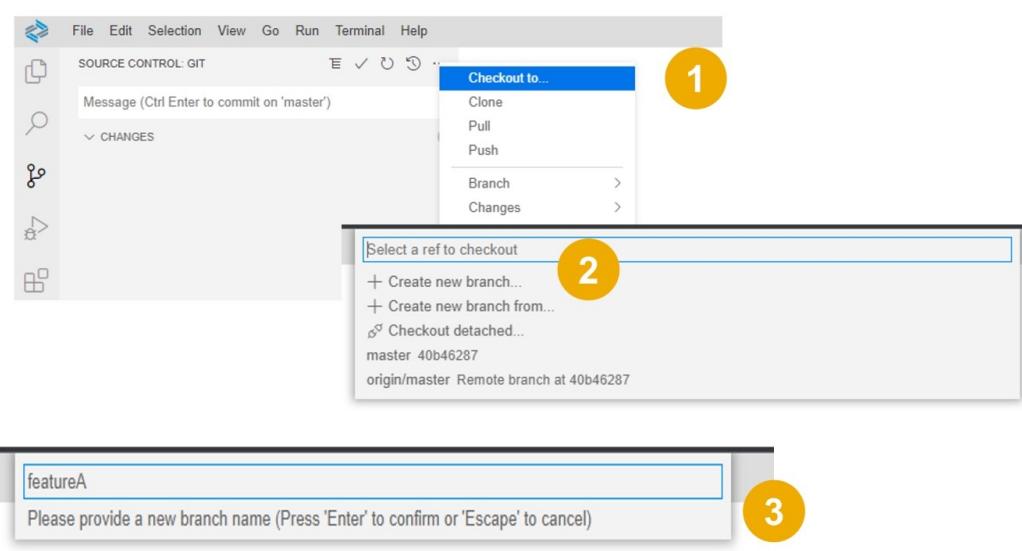


Figure 326: Create a New Branch

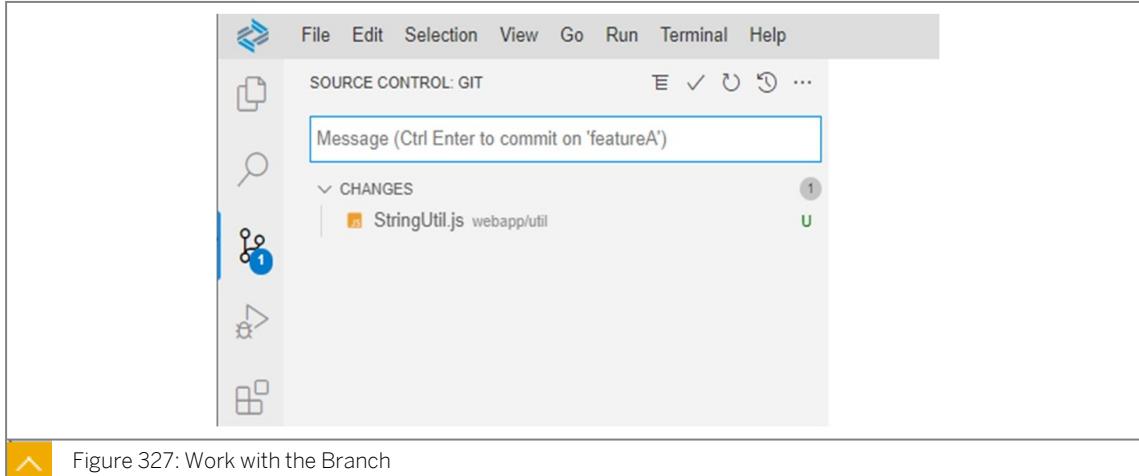


Figure 327: Work with the Branch

After the new branch is created the developer can work as usual with the local repository. The changes are shown in the git-view of BAS.

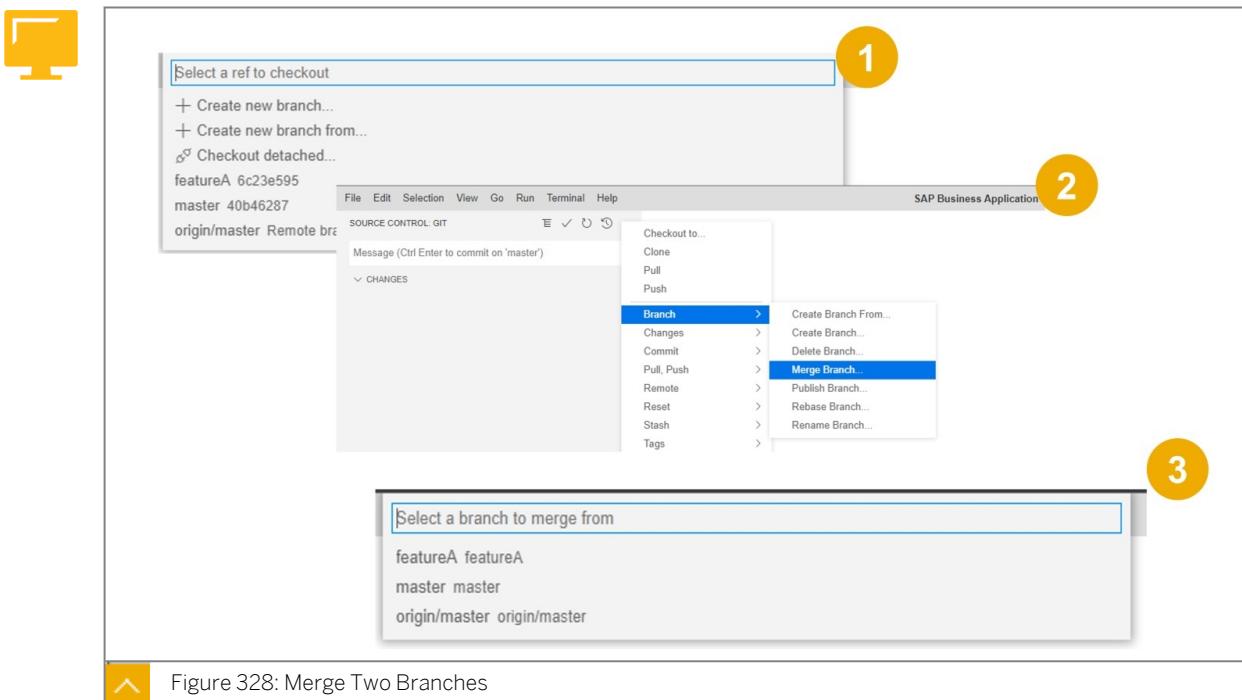


Figure 328: Merge Two Branches

1. Check out the branch that should be merged.
2. Choose merge command.
3. Select the branch that should be merged with your local repository.

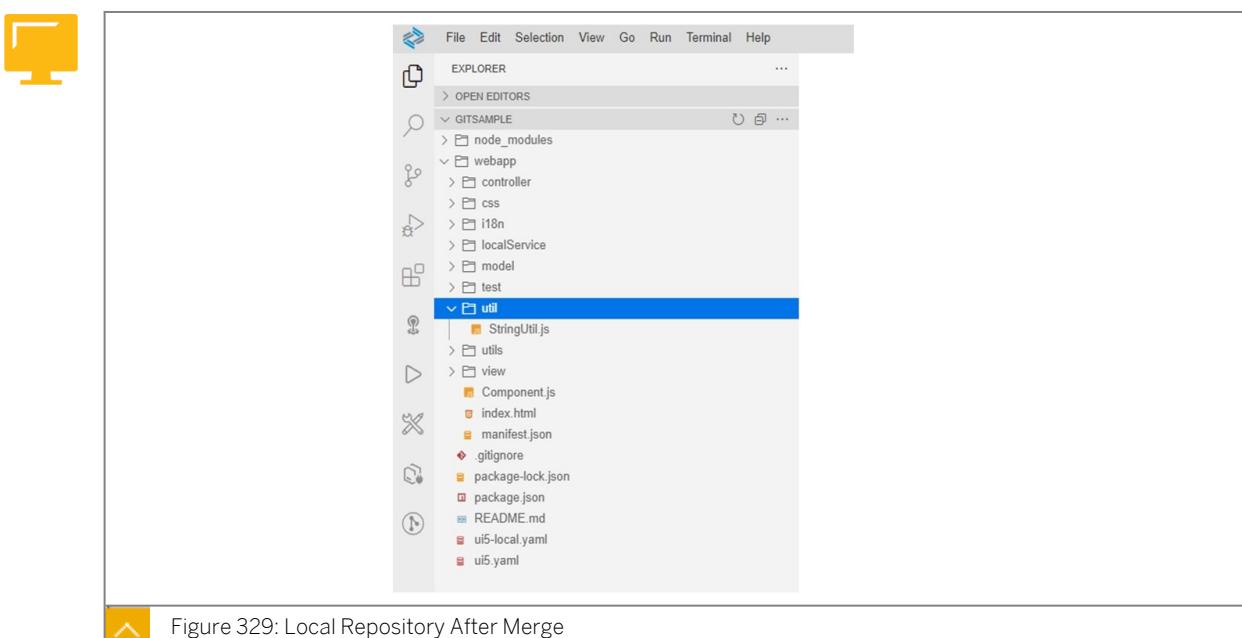


Figure 329: Local Repository After Merge

After the merge is done the local repository contains the changes done in the branch

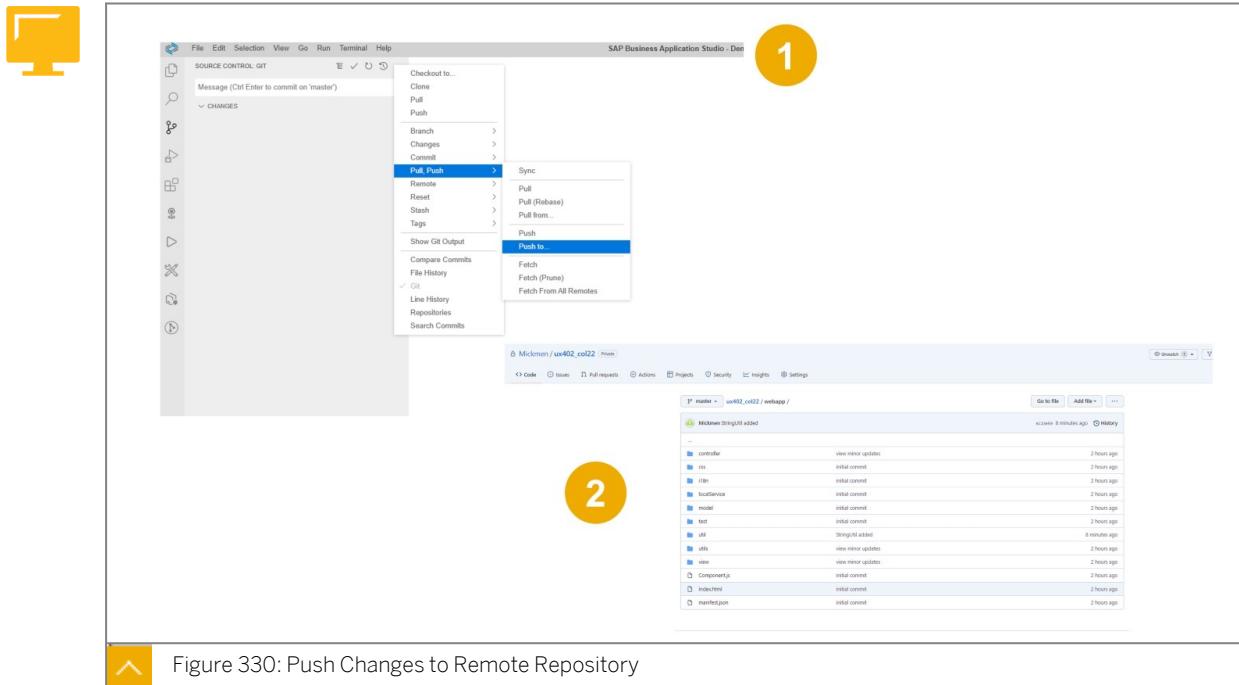


Figure 330: Push Changes to Remote Repository

1. To push local changes to the remote branch choose *Pull, Push* → *Push to*.
2. After the push is finished you can find the changes in the remote repository.



## LESSON SUMMARY

You should now be able to:

- Use GIT branches

## Learning Assessment

1. Which of the following best describes a commit?

*Choose the correct answer.*

- A With every commit, GIT create a new branch.
- B With every commit, GIT takes a snapshot of the current state of the underlying files.
- C With every commit, a new local repository is created.
- D A commit in GIT is a local operation.

2. Which of the following are the main states of a file in GIT?

*Choose the correct answers.*

- A Committed
- B Changed
- C Released
- D Modified

3. In which state are the files in GIT when a remote repository is cloned?

*Choose the correct answers.*

- A Staged and modified
- B Tracked and unmodified
- C Tracked and staged
- D Modified and tracked

4. What is the result of a GIT Reset?

*Choose the correct answer.*

- A Undo a committed snapshot.
- B Removes all untracked files from the working directory.
- C Returns a project back to the previous state.

5. What is the result of a GIT Revert?

*Choose the correct answer.*

- A Undoes a committed snapshot.
- B Removes all untracked files from the working directory.
- C Reverts back to the previous state of the project.

6. What is the result of a GIT Clean?

*Choose the correct answer.*

- A Undoes a committed snapshot.
- B Removes all untracked files from the working directory.
- C Reverts back to the previous state of the project.

7. What is a GIT branch?

*Choose the correct answer.*

- A A GIT branch represents a local working copy of the main development line.
- B A GIT branch always represents the main development line.
- C A GIT branch represents an independent line of development.
- D A GIT branch is the SAP implementation of GIT.

8. Which of the following statements are true about the merge functionality in GIT?

*Choose the correct answer.*

- A A merge deletes the content of a branch.
- B Allows the merging of two local branches into one local branch.
- C Allows integration of a branch into another branch.

## Learning Assessment - Answers

1. Which of the following best describes a commit?

*Choose the correct answer.*

- A With every commit, GIT create a new branch.
- B With every commit, GIT takes a snapshot of the current state of the underlying files.
- C With every commit, a new local repository is created.
- D A commit in GIT is a local operation.

Correct. When a commit is processed, GIT takes a snapshot of the current state of the underlying files.

2. Which of the following are the main states of a file in GIT?

*Choose the correct answers.*

- A Committed
- B Changed
- C Released
- D Modified

Correct. Committed, Modified, and Changed are the main states of a file in GIT.

3. In which state are the files in GIT when a remote repository is cloned?

*Choose the correct answers.*

- A Staged and modified
- B Tracked and unmodified
- C Tracked and staged
- D Modified and tracked

Correct. When a remote GIT repository is cloned, all of the files are in the tracked and unmodified state.

4. What is the result of a GIT Reset?

*Choose the correct answer.*

- A Undo a committed snapshot.
- B Removes all untracked files from the working directory.
- C Returns a project back to the previous state.

Correct. A GIT Reset reverts the project back to the previous state.

5. What is the result of a GIT Revert?

*Choose the correct answer.*

- A Undoes a committed snapshot.
- B Removes all untracked files from the working directory.
- C Reverts back to the previous state of the project.

Correct. A GIT Revert undoes a committed snapshot.

6. What is the result of a GIT Clean?

*Choose the correct answer.*

- A Undoes a committed snapshot.
- B Removes all untracked files from the working directory.
- C Reverts back to the previous state of the project.

Correct. A GIT Clean removes all untracked files from the working directory.

7. What is a GIT branch?

*Choose the correct answer.*

- A A GIT branch represents a local working copy of the main development line.
- B A GIT branch always represents the main development line.
- C A GIT branch represents an independent line of development.
- D A GIT branch is the SAP implementation of GIT.

Correct. A GIT branch represents an independent line of development.

8. Which of the following statements are true about the merge functionality in GIT?

*Choose the correct answer.*

- A A merge deletes the content of a branch.
- B Allows the merging of two local branches into one local branch.
- C Allows integration of a branch into another branch.

Correct. A merge in GIT integrates a branch into another branch.