



# SAP Fiori® Launchpad

## Development and Extensibility

- › Learn to work with SAP Fiori launchpad
- › Use client-side services in your SAPUI5 applications
- › Customize SAP Fiori launchpad with UI extensions, plugins, and custom tiles

Steve Guo



Rheinwerk  
Publishing



SAP PRESS is a joint initiative of SAP and Rheinwerk Publishing. The know-how offered by SAP specialists combined with the expertise of Rheinwerk Publishing offers the reader expert books in the field. SAP PRESS features first-hand information and expert advice, and provides useful skills for professional decision-making.

SAP PRESS offers a variety of books on technical and business-related topics for the SAP user. For further information, please visit our website: <http://www.sap-press.com>.

Anil Bavaraju

SAP Fiori Implementation and Development (2<sup>nd</sup> Edition)

2017, 615 pages, hardcover and e-book

[www.sap-press.com/4401](http://www.sap-press.com/4401)

Goebels, Nepraunig, Seidel

SAPUI5: The Comprehensive Guide

2016, 672 pages, hardcover and e-book

[www.sap-press.com/3980](http://www.sap-press.com/3980)

Bönnen, Drees, Fischer, Heinz, Strothmann

SAP Gateway and OData (3<sup>rd</sup> Edition)

2019, 841 pages, hardcover and e-book

[www.sap-press.com/3904](http://www.sap-press.com/3904)

Krishna Kishor Kammaje

SAP Fiori Certification Guide: Development Associate Exam

2018, 474 pages, paperback and e-book

[www.sap-press.com/4501](http://www.sap-press.com/4501)

Steve Guo

# SAP Fiori® Launchpad

Development and Extensibility

# Dear Reader,

Someone once said that a UI is like a joke; if you have to explain it, it isn't very good. Well, no one is laughing at SAP Fiori launchpad, but that doesn't mean that it can't be improved. Your users are always going to want something new—a button, a footer, or a way to navigate directly between their most commonly used apps. It's your job to make sure they laugh with delight when they see it!

Expert author Steve Guo has been a delight to work with throughout the writing process. I have been continually impressed with his knowledge of the subject and his dedication to providing the best possible book for you, the readers! It's certainly been a labor of love, and we're both excited for it to finally be in your hands.

What did you think about *SAP Fiori Launchpad: Development and Extensibility*? Your comments and suggestions are the most useful tools to help us make our books the best they can be. Please feel free to contact me and share any praise or criticism you may have.

Thank you for purchasing a book from SAP PRESS!

**Meagan White**

Editor, SAP PRESS

[meaganw@rheinwerk-publishing.com](mailto:meaganw@rheinwerk-publishing.com)

[www.sap-press.com](http://www.sap-press.com)

Rheinwerk Publishing • Boston, MA

## Notes on Usage

This e-book is **protected by copyright**. By purchasing this e-book, you have agreed to accept and adhere to the copyrights. You are entitled to use this e-book for personal purposes. You may print and copy it, too, but also only for personal use. Sharing an electronic or printed copy with others, however, is not permitted, neither as a whole nor in parts. Of course, making them available on the Internet or in a company network is illegal as well.

For detailed and legally binding usage conditions, please refer to the section [Legal Notes](#).

This e-book copy contains a **digital watermark**, a signature that indicates which person may use this copy:

# Imprint

This e-book is a publication many contributed to, specifically:

**Editor** Meagan White

**Acquisitions Editor** Hareem Shafi

**Copyeditor** Melinda Rankin

**Cover Design** Graham Geary

**Photo Credit** iStockphoto.com/474014228/© Maxiphoto

**Production E-Book** Kelly O'Callaghan

**Typesetting E-Book** III-satz, Husby (Germany)

We hope that you liked this e-book. Please share your feedback with us and read the [Service Pages](#) to find out how to contact us.

**The Library of Congress has cataloged the printed edition as follows:**

Names: Guo, Steve, author.

Title: SAP Fiori Launchpad : development and extensibility / Steve Guo.

Description: 1st edition. | Bonn ; Boston : Rheinwerk Publishing, [2019] |  
Includes index.

Identifiers: LCCN 2018045672 (print) | LCCN 2018046737 (ebook) | ISBN  
9781493216468 (ebook) | ISBN 9781493216451 (alk. paper)

Subjects: LCSH: SAP Fiori. | Enterprise application integration (Computer  
systems) | Cross-platform software development. | User interfaces  
(Computer systems) | Business enterprises--Data processing.

Classification: LCC QA76.76.A65 (ebook) | LCC QA76.76.A65 G865 2019 (print) |  
DDC 005.4/37-dc23

LC record available at <https://lccn.loc.gov/2018045672>

**ISBN 978-1-4932-1645-1 (print)**

**ISBN 978-1-4932-1646-8 (e-book)**

**ISBN 978-1-4932-1647-5 (print and e-book)**

© 2019 by Rheinwerk Publishing, Inc., Boston (MA)

1<sup>st</sup> edition 2019

# Contents

|               |    |
|---------------|----|
| Preface ..... | 13 |
|---------------|----|

## 1 Overview of SAP Fiori Launchpad 17

---

|  |    |
|--|----|
| <b>1.1 Introduction to SAP Fiori Launchpad .....</b>                   | 17 |
| 1.1.1 End User's Perspective .....                                     | 18 |
| 1.1.2 Administrator's Perspective .....                                | 18 |
| 1.1.3 Developer's Perspective .....                                    | 19 |
| <b>1.2 Versions of SAP Fiori Launchpad .....</b>                       | 19 |
| 1.2.1 SAP Fiori Launchpad for SAP NetWeaver with SAP_UI Component .... | 19 |
| 1.2.2 SAP Fiori Launchpad in UI Add-On for SAP NetWeaver .....         | 20 |
| 1.2.3 SAP Fiori Launchpad for SAP Cloud Platform .....                 | 20 |
| 1.2.4 SAP Fiori Launchpad for SAP S/4HANA Cloud .....                  | 21 |
| 1.2.5 SAP Fiori Launchpad for SAP HANA XS Advanced .....               | 21 |
| 1.2.6 SAP Fiori Launchpad for SAP Enterprise Portal .....              | 21 |
| <b>1.3 Development Capabilities of SAP Fiori Launchpad .....</b>       | 21 |
| 1.3.1 Embed SAPUI5 Applications in SAP Fiori Launchpad .....           | 22 |
| 1.3.2 Client-Side Services .....                                       | 22 |
| 1.3.3 Extend SAP Fiori Launchpad .....                                 | 22 |
| 1.3.4 Custom Tile Types for SAP Fiori Launchpad .....                  | 23 |
| 1.3.5 Plug-ins for SAP Fiori Launchpad .....                           | 23 |
| <b>1.4 Summary .....</b>   | 23 |

## 2 Development Environment Setup 25

---

|  |    |
|--|----|
| <b>2.1 Cloud-Based Development Environment .....</b>   | 25 |
| 2.1.1 Register SAP Cloud Platform Trial Account .....  | 26 |
| 2.1.2 Open SAP Web IDE Full-Stack Version .....        | 29 |
| 2.1.3 Activate SAP Cloud Platform Portal Service ..... | 31 |

|            |   |    |
|------------|---|----|
| <b>2.2</b> | <b>On-Premise Development Environment</b>                       | 38 |
| 2.2.1      | Prerequisites   | 39 |
| 2.2.2      | Create a Virtual Machine  | 45 |
| 2.2.3      | Install the Operating System                                    | 49 |
| 2.2.4      | Prepare the Operating System for SAP NetWeaver AS ABAP          | 55 |
| 2.2.5      | Install SAP NetWeaver AS ABAP Components                        | 64 |
| 2.2.6      | Post-Installation Steps   | 66 |
| <b>2.3</b> | <b>Connect a Cloud Environment to an On-Premise Environment</b> | 80 |
| 2.3.1      | Install SAP Cloud Connector                                     | 80 |
| 2.3.2      | Set Up SAP Cloud Connector                                      | 84 |
| 2.3.3      | Create a Destination in SAP Cloud Platform                      | 92 |
| <b>2.4</b> | <b>Summary</b>  | 95 |

---

## **3 SAPUI5 Applications in SAP Fiori Launchpad**

---

|            |  |     |
|------------|--|-----|
| <b>3.1</b> | <b>Architecture</b>                                      | 97  |
| <b>3.2</b> | <b>Intent-Based Navigation</b>                           | 101 |
| <b>3.3</b> | <b>Embedding SAPUI5 Applications</b>                     | 103 |
| 3.3.1      | Testing SAPUI5 Apps                                      | 103 |
| 3.3.2      | Provisioning Application Title and Description           | 106 |
| 3.3.3      | Proposing an Intent                                      | 107 |
| <b>3.4</b> | <b>Navigation between SAPUI5 Applications</b>            | 112 |
| 3.4.1      | Set Up Test Environment for Cross-Application Navigation | 112 |
| 3.4.2      | Calling Navigation Services                              | 115 |
| 3.4.3      | Test Supportability                                      | 117 |
| 3.4.4      | Navigation Back to Previous App                          | 119 |
| 3.4.5      | Configuring Navigation Targets in App Descriptor         | 121 |
| <b>3.5</b> | <b>Passing Parameters between Apps</b>                   | 124 |
| <b>3.6</b> | <b>Summary</b>   | 126 |

|   |     |
|---|-----|
| <b>4 Client-Side Services</b>                                 | 127 |
| <b>  4.1 User Info Service .....</b>                          | 128 |
| 4.1.1 Creating and Testing a Simple SAPUI5 Application .....  | 129 |
| 4.1.2 Exploring the User Info Service API .....               | 140 |
| 4.1.3 Creating and Testing a Complex SAPUI5 Application ..... | 148 |
| <b>  4.2 Bookmark Service .....</b>                           | 167 |
| 4.2.1 Creating an SAPUI5 Application .....                    | 167 |
| 4.2.2 Using Additional Functions .....                        | 177 |
| <b>  4.3 Personalization Service .....</b>                    | 179 |
| 4.3.1 Creating an SAPUI5 Application .....                    | 179 |
| 4.3.2 Data Storage Locations .....                            | 184 |
| 4.3.3 Handling Complex Data .....                             | 185 |
| <b>  4.4 Summary .....</b>                                    | 188 |
| <br>  |     |
| <b>5 Extensibility</b>  | 189 |
| <b>  5.1 Extension Options .....</b>                          | 189 |
| <b>  5.2 App Title Information Extensions .....</b>           | 192 |
| 5.2.1 Preparing a Project .....                               | 193 |
| 5.2.2 Changing the App Title .....                            | 196 |
| 5.2.3 Changing the Title Context Menu .....                   | 198 |
| <b>  5.3 Shell Header Extensions .....</b>                    | 201 |
| 5.3.1 Preparing a Project .....                               | 201 |
| 5.3.2 Setting a Secondary Header Title .....                  | 203 |
| 5.3.3 Managing Header Items .....                             | 204 |
| 5.3.4 Managing Extension Element States .....                 | 210 |
| <b>  5.4 Launch Page Extensions .....</b>                     | 211 |
| 5.4.1 Preparing a Project .....                               | 211 |
| 5.4.2 Adding a Subheader .....                                | 213 |
| 5.4.3 Adding a Footer Bar .....                               | 214 |
| 5.4.4 Adding Tool Area Items .....                            | 215 |

|   |     |
|---|-----|
| <b>5.5 Me Area Extensions .....</b>                   | 218 |
| 5.5.1 Preparing a Project .....                       | 218 |
| 5.5.2 Adding a Button to the Me Area .....            | 220 |
| 5.5.3 Adding Setting Options .....                    | 221 |
| 5.5.4 Fetching Data from Custom Setting Options ..... | 223 |
| <b>5.6 Summary .....</b>                              | 225 |

## **6 Custom Tile Types**

---

|   |     |
|---|-----|
| <b>6.1 Creating a Custom Tile .....</b>                           | 227 |
| 6.1.1 Basics of a Generic Tile .....                              | 227 |
| 6.1.2 Creating a Generic Tile .....                               | 229 |
| 6.1.3 Organizing Tile Content .....                               | 231 |
| 6.1.4 Creating a Slide Tile .....                                 | 235 |
| 6.1.5 Adding Content to Your Tile .....                           | 236 |
| <b>6.2 Deploying a Custom Tile to SAP Cloud Platform .....</b>    | 237 |
| 6.2.1 Deploying Your Tile to SAP Cloud Platform Portal .....      | 237 |
| 6.2.2 Applying Your Tile to an SAP Fiori App .....                | 240 |
| 6.2.3 Setting and Parsing Parameters .....                        | 243 |
| 6.2.4 Implementing Navigation .....                               | 247 |
| <b>6.3 Deploying a Custom Tile to SAP NetWeaver AS ABAP .....</b> | 248 |
| 6.3.1 Developing a Tile .....                                     | 249 |
| 6.3.2 Creating a CHIP Description File .....                      | 251 |
| 6.3.3 Deploying Your Tile as an SAPUI5 Application .....          | 253 |
| 6.3.4 Registering Your Tile .....                                 | 255 |
| 6.3.5 Creating a Configuration Screen .....                       | 258 |
| 6.3.6 Setting and Getting Parameters .....                        | 264 |
| <b>6.4 Summary .....</b>  | 267 |

|               |   |     |
|---------------|---|-----|
| <b>7</b>      | <b>Plug-Ins</b>                                       | 269 |
| <b>7.1</b>    | <b>Developing a Plug-In</b>                           | 269 |
| 7.1.1         | Creating a Plug-In Using a Template                   | 270 |
| 7.1.2         | Adjusting Implementation Code                         | 274 |
| 7.1.3         | Testing Your Plug-In                                  | 275 |
| <b>7.2</b>    | <b>Deploying the Plug-In on SAP Cloud Platform</b>    | 277 |
| 7.2.1         | Deployment and Activation                             | 278 |
| 7.2.2         | Avoiding Multiple Code Executions                     | 282 |
| 7.2.3         | Working with Configurable Parameters                  | 284 |
| <b>7.3</b>    | <b>Deploying the Plug-In on SAP NetWeaver AS ABAP</b> | 287 |
| 7.3.1         | Deployment  | 287 |
| 7.3.2         | Configuration   | 289 |
| <b>7.4</b>    | <b>Predefined Plug-Ins</b>                            | 295 |
| 7.4.1         | Plug-In for Setting User Defaults                     | 295 |
| 7.4.2         | Plug-In for Activating Runtime Authoring              | 296 |
| <b>7.5</b>    | <b>Summary</b>  | 296 |
| <br>          |   |     |
| The Author    |   | 297 |
| Index         |   | 299 |
| <br>          |   |     |
| Service Pages |   |     |
| Legal Notes   |   |     |



# Preface

Welcome to the first book on development for and extensibility of SAP Fiori launchpad. Since SAP's introduction of SAP Fiori in 2015, all SAP applications have migrated to SAP Fiori-style UIs for user experience. In addition, there are many SAP Fiori-style apps developed by partners and customers in the SAP ecosystem. SAP Fiori launchpad is the most popular portal for all the SAP Fiori apps, both in cloud and on-premise environments.

In this book, we'll focus on SAP Fiori launchpad, especially from a developer's perspective. You'll get insight into how your SAPUI5 applications interact with SAP Fiori launchpad and develop solid skills for how to extend SAP Fiori launchpad to cater to your specific needs.

## Who This Book Is For

In this book, we're focusing on the development and extensibility of SAP Fiori launchpad. As a reader, you should have at least a basic understanding of SAP Fiori and web applications. Throughout the book, we'll guide you through SAP Fiori launchpad step by step. You can finish all the demos in this book with only a moderate amount of prior knowledge of SAPUI5 programming and SAP Fiori launchpad.

In general, in a real project, by the time you need the knowledge in this book, you should have already finished some SAPUI5 applications. So when you want to put knowledge from this book into your real project, you need to have already acquired basic knowledge about how to develop an SAPUI5 application using SAP Web IDE, including drawing a view, controller programming, and data binding. You'll likely already have knowledge about how to deploy and manage your SAPUI5 application in both SAP Cloud Platform and SAP NetWeaver Application Server for ABAP (SAP NetWeaver AS ABAP).

For content associated with SAP NetWeaver AS ABAP, you need basic knowledge of ABAP, such as how to enter a transaction and how to log onto the system.

## How This Book Is Organized

We've organized this book into different topics, ranging from easy to hard.

We recommend reading this book sequentially from [Chapter 1](#) onward. However, if you prefer, you can directly go to any chapter to start reading about that chapter's topic:

- **Chapter 1**

This chapter provides an overview of the SAP Fiori launchpad. We'll discuss its overall structure and the functions provided by SAP Fiori launchpad from different perspectives. You will also learn about the different versions of SAP Fiori launchpad that exist in the market. Finally, we'll provide an overview of its development and extension capabilities.

- **Chapter 2**

This chapter focuses on the registration, setup, and installation of development environments for the demos in this book. We'll guide you setting up the cloud environment based on SAP Cloud Platform and the on-premise environment based on the developer edition of SAP NetWeaver AS ABAP 7.52.

- **Chapter 3**

In this chapter, we'll explain the architecture of SAP Fiori launchpad, especially how SAPUI5 applications are loaded into SAP Fiori launchpad. You'll become familiar with the concept of intent-based navigation and why you can benefit from it. Finally, you'll learn how to navigate between SAPUI5 apps using the cross-application navigation service.

- **Chapter 4**

This chapter focuses on client services provided by SAP Fiori launchpad. You'll learn how to get a user's information, add a page as a tile to SAP Fiori launchpad, and persist data of any type temporarily. All these functions are realized by a set of JavaScript APIs provided by SAP Fiori launchpad.

- **Chapter 5**

In this chapter, we'll review extensibility options for SAP Fiori launchpad. You'll learn how to add custom elements to various areas of the home page, as well as the Me Area and the **Settings** dialog. Then we'll discuss how to interact with those extended elements.

- **Chapter 6**

This chapter introduces the custom tile type. You'll learn how to create your own tile template and use it for your app, both on SAP Cloud Platform and SAP NetWeaver AS ABAP.

- **Chapter 7**

Plug-ins are SAPUI5 components executed at the startup of SAP Fiori launchpad. In this chapter, you'll learn how to create a plug-in and deploy it on both SAP Cloud Platform and SAP NetWeaver AS ABAP.

## Acknowledgments

As I'm sure you understand, a book such as this one, which covers so much detailed and deep knowledge, requires lots of effort and time to create. I'm so grateful that I've had the understanding and support of my family while I worked on this project, especially that of my wife, Zhang Jialei. I would also like to thank Hareem Shafi and Meagan White at Rheinwerk Publishing.

## Conclusion

Reading this book will provide you with a deep knowledge of how to make use of SAP Fiori launchpad as an SAPUI5 developer. This book will guide you to various development topics for SAP Fiori launchpad, including navigation, client services, extensibility, custom tiles, and plug-ins. For each topic, there will be several step-by-step, hands-on exercises to help you learn, test, and understand. After reading this book, you should be an expert on how to best make use of the development functions provided by SAP Fiori launchpad.



# Chapter 1

## Overview of SAP Fiori Launchpad

*SAP Fiori launchpad is an ideal way for all users to access all apps, regardless of what SAP platform they use. Before diving into the details of SAP Fiori launchpad, we'll give you an overview of the solution in this chapter.*

As a developer, you know that when a simple program grows to encompass a whole system, things become complex very quickly. The unique development experience provided by SAP (and other enterprise-level development platforms), compared to just developing and deploying a program on an open source-server such as Apache Tomcat, lets you separate a complex system into many small applications, which are loosely coupled. As a developer, you're thus working on a small piece of the system at a time.

This is also true for SAPUI5 development. You can create an SAPUI5 project with tens or even hundreds of views and controllers, allowing access to all files of a system in a single SAPUI5 application, but the best practice is to keep a single SAPUI5 application to no more than three levels of user interface. Given that there are sometimes alternative views for the same level of UI, normally you'll want to have no more than five SAPUI5 views in an app. If you follow this guideline, you'll have tens or even hundreds of SAPUI5 apps for a module or system.

SAP Fiori launchpad can help you manage all the apps you've developed and make them easy to manage. It also provides application programming interfaces (APIs) to help your application communicate with it. Furthermore, it also works with non-SAPUI5 apps such as Web Dynpro apps, SAP GUI apps, or generic HTML pages.

In this chapter, we'll introduce you to SAP Fiori launchpad with a high-level overview.

### 1.1 Introduction to SAP Fiori Launchpad

SAP Fiori launchpad is the ideal system entry point for end users. It provides a portal to serve as the single point of entry for all applications in your system, regardless of

which technology they're based on and which backend system they're installed on. SAP Fiori launchpad can also help administrators manage all apps in a system easily. It can control access to apps by role and by device, and it also provides an architecture that separates development and deployment to make a developer's work easier. In this section, we'll highlight some of the benefits of SAP Fiori launchpad from different perspectives.

### 1.1.1 End User's Perspective

From an end user's perspective, SAP Fiori launchpad is an easy-to-use portal with the following functionality:

- **Friendly user interface**

The SAP Fiori launchpad home page consists of a group navigator and a list of tiles.

Each tile is a shortcut to an app. It's user-friendly and easy to understand. An end user can enter any app by simply clicking its tile.

Some tiles also contain a number or a microchart to give users quick insights into important numbers.

- **Search capability**

A global search function lets users search for apps or even business data systemwide.

- **Me Area**

The Me Area allows end users to change settings and open recently accessed apps.

- **Notification center**

The notification center can get all notifications from the backend system and summarize all messages on the right side of SAP Fiori launchpad.

- **App finder**

The app finder is used to find apps not on the user's home page. It can also be used for accessing the easy access menu in SAP GUI.

- **Customizable home page**

The home page can be customized by the end user. An end user can add, delete, or adjust groups and add, delete, or move apps.

### 1.1.2 Administrator's Perspective

Almost every function in SAP Fiori launchpad can be implemented via configuration.

The administrator plays an important role during implementation of SAP Fiori launchpad.

All apps that can be accessed via SAP Fiori launchpad need to be configured. A unified representation of different types of apps is called *target mapping*, and target mapping is defined by an *intent*. An intent consists of a *semantic object* and an *action*. In the *SalesOrder-create* intent, for example, the semantic object is *SalesOrder* and the action is *create*. Whether you're configuring an SAPUI5 app, an SAP GUI transaction, a Web Dynpro application, or a generic HTML page, you need to define an intent first. Then you can create a tile based on this intent and manage it using a *tile catalog*.

### 1.1.3 Developer's Perspective

As a developer, you need to develop your own SAPUI5 applications that are compliant with SAP Fiori launchpad, as follows:

- First, the SAPUI5 application must be a UI component. SAP Fiori launchpad loads your app as a component, not a web page.
- Second, all access to other apps must happen through services provided by SAP Fiori launchpad. Do not access other apps directly via URLs.
- Finally, SAP Fiori launchpad provides lots of APIs that can be called using JavaScript. Try to use them as much as possible so that your code can work even when deployed on a different backend system.

## 1.2 Versions of SAP Fiori Launchpad

One purpose of SAP Fiori launchpad is to make sure SAPUI5 applications can be transported into systems with different technologies without changing a single line of code.

As you may know, SAP uses both on-premise and cloud architecture. So how do you make sure all your code is working when you're planning a migration from an on-premise ABAP system to cloud services that are mainly based on Java? APIs interact with the system for cases such as these, performing such tasks as getting the current user's information.

Therefore, it's important that SAP Fiori launchpad be available for all SAP platforms and that the APIs provided by SAP Fiori launchpad remain stable.

### 1.2.1 SAP Fiori Launchpad for SAP NetWeaver with SAP\_UI Component

SAP NetWeaver is SAP's major technical platform for an on-premise system. The ABAP-based system can be the platform for various products, such as SAP ERP or SAP CRM.

Starting with version 7.40, the SAP\_UI component is included as a default component in the system. And for each version of SAP NetWeaver, there is a new version of SAP\_UI with same version number. The inclusion of the SAP\_UI component means that SAP Fiori launchpad is also contained in the system by default.

SAP NetWeaver with the SAP\_UI component uses SAP Fiori launchpad as the default portal page and provides an SAP Fiori launchpad designer to help administrators do their work. This version of SAP Fiori launchpad may be the most powerful version because it supports all functions of SAP Fiori launchpad and gives you maximum extensibility capabilities.

The following versions of SAP Fiori launchpad for SAP NetWeaver currently exist:

- SAP Fiori launchpad for SAP NetWeaver Application Server (AS) for ABAP (SAP NetWeaver AS ABAP) 7.52 with SAP\_UI 752
- SAP Fiori launchpad for SAP NetWeaver AS ABAP 7.51 with SAP\_UI 751
- SAP Fiori launchpad for NetWeaver AS ABAP 7.5 with SAP\_UI 750
- SAP Fiori launchpad for SAP NetWeaver AS ABAP 7.4 with SAP\_UI 740

### 1.2.2 SAP Fiori Launchpad in UI Add-On for SAP NetWeaver

For SAP NetWeaver AS ABAP with a version number lower than 7.40, you can still use SAP Fiori launchpad as the SAP\_UI component. The only difference is you need to install the new UI add-on yourself. The functions of the UI add-on are almost identical to the SAP\_UI component.

### 1.2.3 SAP Fiori Launchpad for SAP Cloud Platform

SAP Cloud Platform Portal allows you to create numerous portal sites, but SAP Fiori launchpad is used in most cases. The main features of SAP Fiori launchpad for SAP Cloud Platform are as follows:

- You can have more than one SAP Fiori launchpad site.
- Configuration is handled by SAP Fiori configuration cockpit, which is much easier to use than the SAP Fiori launchpad designer.
- It can be set up for free; anyone can try it at no cost.
- The backend for SAP Cloud Platform Portal services can be Java, but all the client-side APIs remain.

### 1.2.4 SAP Fiori Launchpad for SAP S/4HANA Cloud

From technical perspective, SAP Fiori launchpad for SAP S/4HANA Cloud is just SAP Fiori launchpad on an SAP Cloud Platform Portal site. But it also contains predefined standard apps and standard configuration information. It will accelerate the adoption of standard SAP Fiori apps provided by SAP S/4HANA.

### 1.2.5 SAP Fiori Launchpad for SAP HANA XS Advanced

SAP HANA XS Advanced is a basic microservice-based engine that enables SAP HANA as a development or testing environment for microservices. This version of SAP Fiori launchpad is just a simple version on SAP Cloud Platform.

### 1.2.6 SAP Fiori Launchpad for SAP Enterprise Portal

The SAP Enterprise Portal system provided by SAP also provides an SAP Fiori launchpad page. But keep in mind that it only realizes SAP Fiori launchpad on the end user level. The deployment and client APIs are quite different compared to other versions of SAP Fiori launchpad. For access to the system, you need to call APIs provided directly by SAP Enterprise Portal, not SAP Fiori launchpad itself. That means when you move applications from other platforms to SAP Enterprise Portal, you should evaluate if all SAP Fiori launchpad APIs are available and functional for your application.

---

#### Note

In this book, we only discuss SAP Fiori launchpad in the UI add-on for SAP NetWeaver and SAP Fiori launchpad for SAP Cloud Platform.

## 1.3 Development Capabilities of SAP Fiori Launchpad

In this book, we're focusing on how to enable all the potential of SAP Fiori launchpad by writing SAPUI5 code. In this section, we'll provide an overview of what you can do with SAP Fiori launchpad.

### 1.3.1 Embed SAPUI5 Applications in SAP Fiori Launchpad

The first thing you must figure out is how an SAPUI5 application is loaded into home page of SAP Fiori launchpad. Because SAP Fiori launchpad has several types of runtime and supports highly flexible configuration and personalization, loading an SAPUI5 application is much more complex than just navigating to a webpage. You need to know what exactly the intent-based navigation means. Because all SAPUI5 applications are components loaded into SAP Fiori launchpad, you can call the Cross-ApplicationNavigation API to load other SAPUI5 apps.

### 1.3.2 Client-Side Services

A series of client-side services are available for you to communicate with the backend system. The reason to use client-side services instead of calling services directly from the backend is that you can keep your application backend platform-independent.

Services available in SAP Fiori launchpad are as follows:

- The user info service is used to fetch a user's preferences from the backend.
- The bookmark service is used to bookmark a current or specific page as a tile in SAP Fiori launchpad.
- The personalization service is used to save temporary data to the cache at various levels and retrieve it when needed.

### 1.3.3 Extend SAP Fiori Launchpad

SAP Fiori launchpad also provides a set of APIs for adding or changing UI elements in specific areas, such as the header toolbar, Me Area, or settings dialog.

Extensibility options for the SAP Fiori launchpad UI include the following:

- Changing an app tile dynamically using code
- Adding button in the Me Area
- Adding settings options in the user settings dialog
- Adding buttons in a header toolbar
- Creating a subheader bar
- Creating a footer bar
- Creating a left-hand toolbar

### 1.3.4 Custom Tile Types for SAP Fiori Launchpad

There are some predefined tile types (static, dynamic, news, and a set of KPI tiles), which serve for common SAP Fiori application needs. In some cases, however, you need a custom tile type, and you'll also want management functions for the tile types you create.

Custom tiles are supported in both cloud and on-premise versions, but both the implementation and deployment are different. For SAP Cloud Platform Portal, these tiles are quite simple and easy to implement, but SAP Cloud Platform Portal can't provide a customized configuration UI; it uses a generic configuration UI. For SAP NetWeaver AS ABAP, the process is much more complex but provides much more flexibility.

### 1.3.5 Plug-ins for SAP Fiori Launchpad

Plug-ins allow you to execute code on startup of SAP Fiori launchpad. They can also call client APIs and execute some initialize code.

For different versions of SAP Fiori launchpad, the code implementation is the same, but deployment processes are different. In SAP Cloud Platform Portal, you can set an app as a plug-in directly. In SAP NetWeaver AS ABAP, you need to use the predefined Shell-plugin intent.

## 1.4 Summary

Now you've seen the big picture of SAP Fiori launchpad, which is powerful and easy to use. Before you step into all the details, you'll need to set up the development and deployment environment. In [Chapter 2](#), we'll walk you through this setup, for both cloud and on-premise systems.



# Chapter 2

## Development Environment Setup

*In this chapter, we'll walk you through setting up an SAP Fiori launchpad development environment, which is essential for this book. You can either start from a setup-free, cloud-based environment or install an on-premise development system.*

This chapter contains step-by-step guides to help you set up your development environment. We'll begin by showing you how to set up a cloud-based environment ([Section 2.1](#)). Then we'll move on to show you how to set up an on-premise development environment by looking at the preinstallation tasks, the installation itself, and the post-installation tasks ([Section 2.2](#)). We'll finish the chapter by discussing how to connect a cloud environment to an on-premise one ([Section 2.3](#)).

### 2.1 Cloud-Based Development Environment

A cloud-based environment is based on a trial account of SAP Cloud Platform. [Figure 2.1](#) shows what the architecture of this environment looks like and its main components, as follows:

- SAP Web IDE full-stack version for development tools
- HTML5 application repository, a place to store your SAPUI5 applications
- SAP Cloud Platform Portal, a portal service that handles SAP Fiori launchpad
- SAP Fiori launchpad site, an SAP Fiori launchpad runtime in SAP Cloud Platform Portal
- SAP Fiori configuration cockpit, a place to configure your SAP Fiori launchpad site

To set up a cloud-based environment, you first need to have a free trial account for SAP Cloud Platform. Then you'll open SAP Web IDE full-stack for the development tools. Finally, you'll activate SAP Cloud Platform Portal and create an SAP Fiori launchpad site for testing in a production environment.

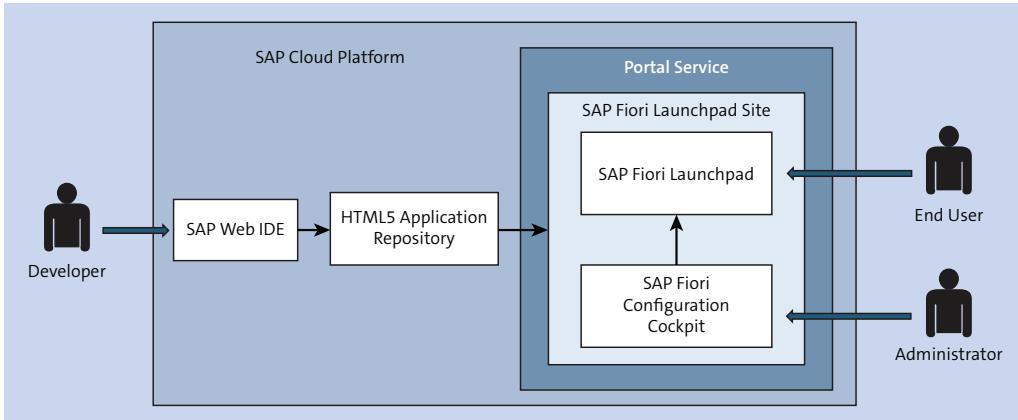


Figure 2.1 Cloud-Based Development Environment Architecture

### 2.1.1 Register SAP Cloud Platform Trial Account

First you need a trial account, which is open to anyone at no cost and with no time limit. Follow this procedure to get a trial account:

1. Open your browser and go to <https://cloudplatform.sap.com>; you'll see the index page of SAP Cloud Platform. Click the **Start Your Free Trial** button, as shown in [Figure 2.2](#).



Figure 2.2 Index Page of SAP Cloud Platform

- Fill in the form with your information. If you already have an SAP account (this can be your SAP Community account or your support account or any account on SAP's website), you can logon directly from the form on the right side (see [Figure 2.3](#)).

Thank you for your interest in SAP

Tell us about yourself

E-mail address: fptest@outlook.com

First name: Steve

Last name: Guo

Company\*: [input field]

Office location: China

Phone: +86 186 1234 5678

Relationship to SAP: Consultant

Create password: [input field]

Confirm password: [input field]

SAP will use any of the data provided hereunder in accordance with the [Privacy Statement](#).

Submit

Log On

steve.guo@outlook.com

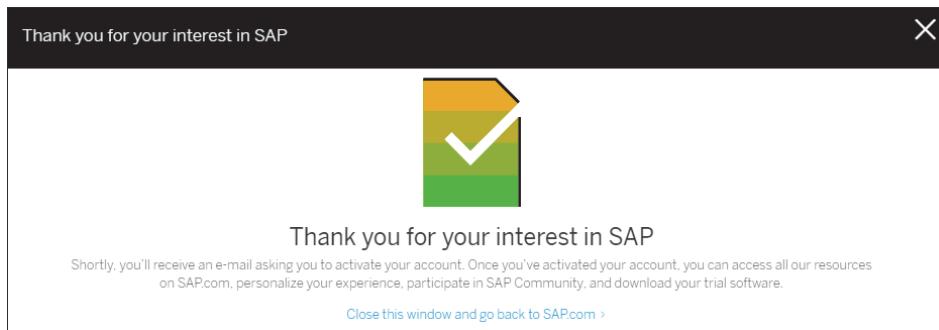
[redacted]

[Log On](#)

[Forgot password?](#)

**Figure 2.3** Registration Form for SAP Cloud Platform

- After you register, you'll see a pop-up telling you to check your email, as shown in [Figure 2.4](#). You need to click the link in your email to activate your registration.



**Figure 2.4** Result of Registration

- After the logon process, you'll enter the SAP Cloud Platform Cockpit trial landing page shown in [Figure 2.5](#). Click **Neo Trial** to access your trial account.

## 2 Development Environment Setup

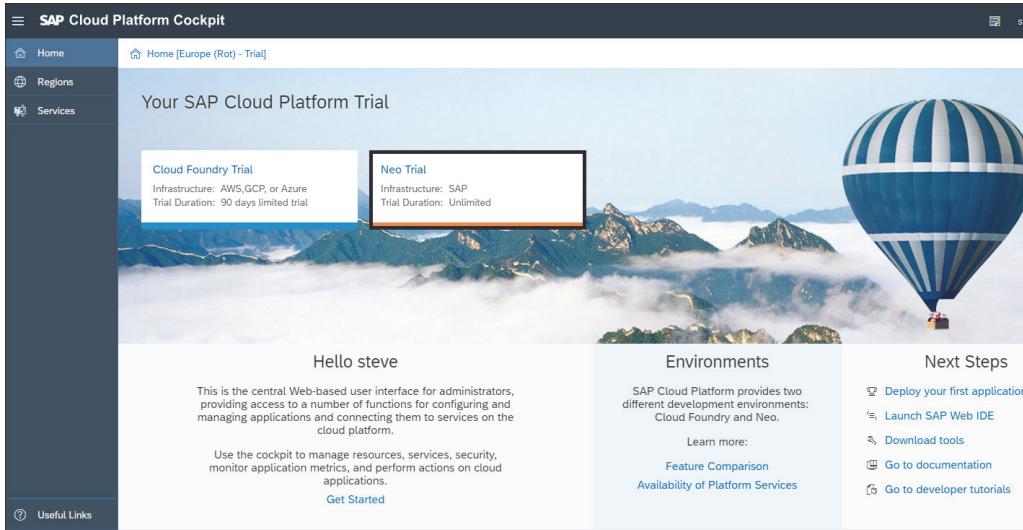


Figure 2.5 Trial Account Landing Page

5. In the cockpit for your trial account, note your trial account name. The account name starts with a letter: *d, i, c, s, or p*, according to your relationship to SAP. Several digits follow the letter, and the trial account name ends with *trial*. In this book, we write *p<xxxxx>trial* as a placeholder for this account name, as shown in [Figure 2.6](#).

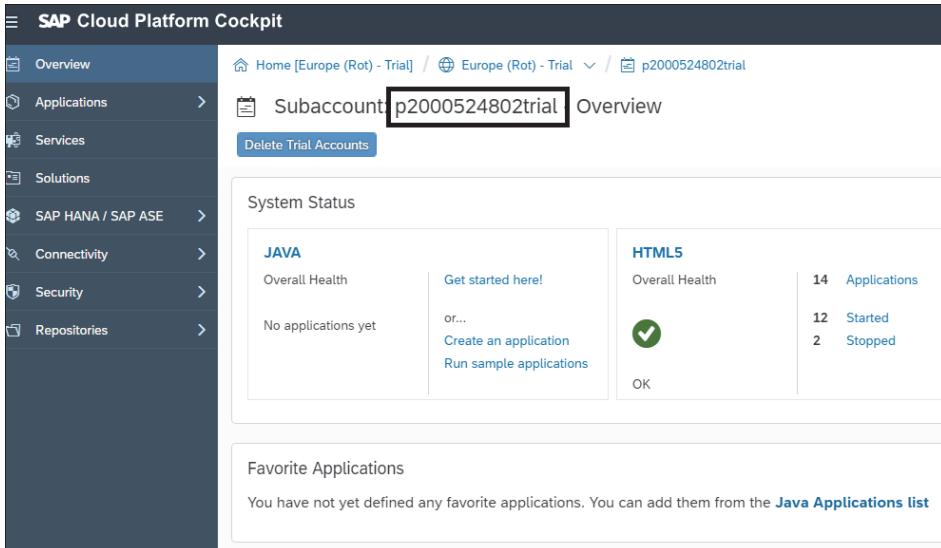


Figure 2.6 SAP Cloud Platform Cockpit for Neo Trial

You'll use your account name many times in this book and in your real-world work, so it's important to note down it in someplace so that you won't need to go back and to find it.

### Note

Remember to add this page to your bookmarks, giving it a descriptive name like *SAP Cloud Platform trial index*. You'll come back to this page later.

## 2.1.2 Open SAP Web IDE Full-Stack Version

In this section, you'll learn how to enter the SAP Web IDE full-stack version, which is the development tool we use in this book. To begin, follow these steps:

1. Click **Service**, then enter “web” in the search field. In the search results, click **SAP Web IDE Full-Stack**, as shown in Figure 2.7.

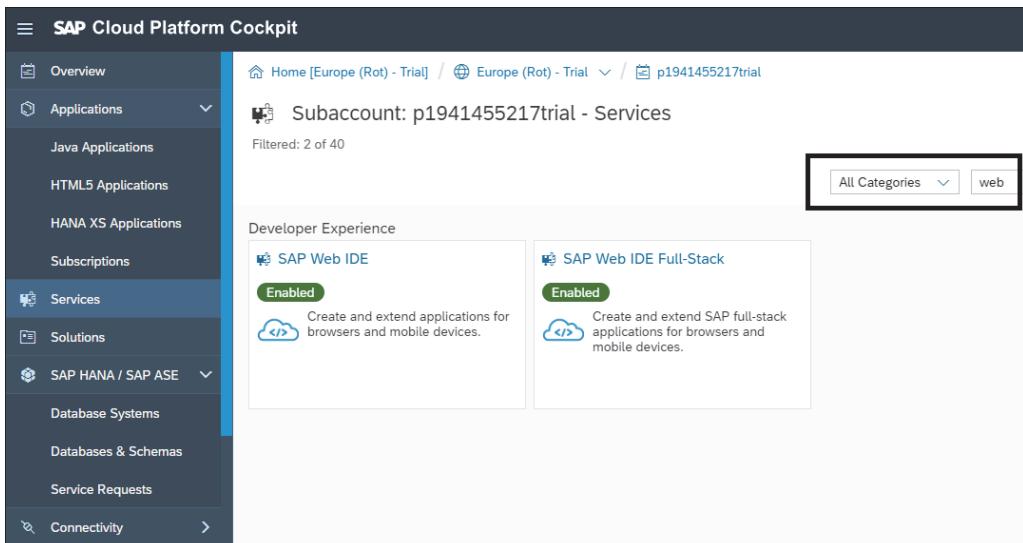


Figure 2.7 Locate Sap Web IDE Full-Stack in Services

2. On the next page, click **Go to Service**, as shown in Figure 2.8.

The screenshot shows the SAP Web IDE Full-Stack - Overview page. At the top left is a cloud icon followed by the text "Service: SAP Web IDE Full-Stack - Overview". Below this are two buttons: "Enabled" (green) and "Disable" (blue). A large text box titled "Service Description" contains the following text: "With SAP Cloud Platform Web IDE Full-Stack, you can easily develop, test, build, deploy, and extend role-based, consumer-grade apps for business users. Create applications rapidly and deliver an outstanding user experience. Developers can extend or build SAP Fiori apps, create new SaaS solutions, extend SAP S/4HANA cloud services, develop hybrid mobile applications, and build IoT apps for SAP Leonardo, using the UI development toolkit for HTML5 (SAPUI5) for desktop and mobile devices, the SAP HANA toolset, and Java programming language and technologies. Integrate with other services that run on SAP Cloud Platform, such as SAP Fiori Cloud apps, Git integration, mobile services, IoT services, and more." To the right of this is a sidebar titled "Availability" with sections for "Cloud Foundry" and "NEO:" showing icons for Brazil, EU, Japan, UAE, and USA.

**Take Action**

- [Configure Service](#)
- [Logs](#)
- [Go to Service](#)

**Additional Resources**

- [Documentation](#)
- [Getting Started](#)

Figure 2.8 SAP Web IDE Full-Stack Version Service Page

3. Click the second icon from the top in the left toolbar to enter your workspace page (see [Figure 2.9](#)). This is the main area for programming.

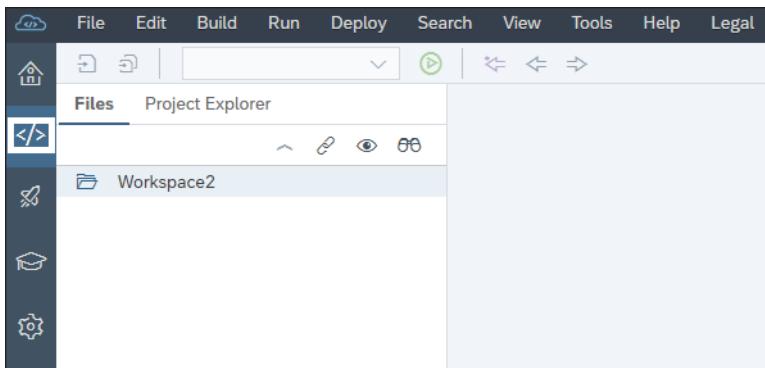


Figure 2.9 SAP Web IDE Workspace

**Note**

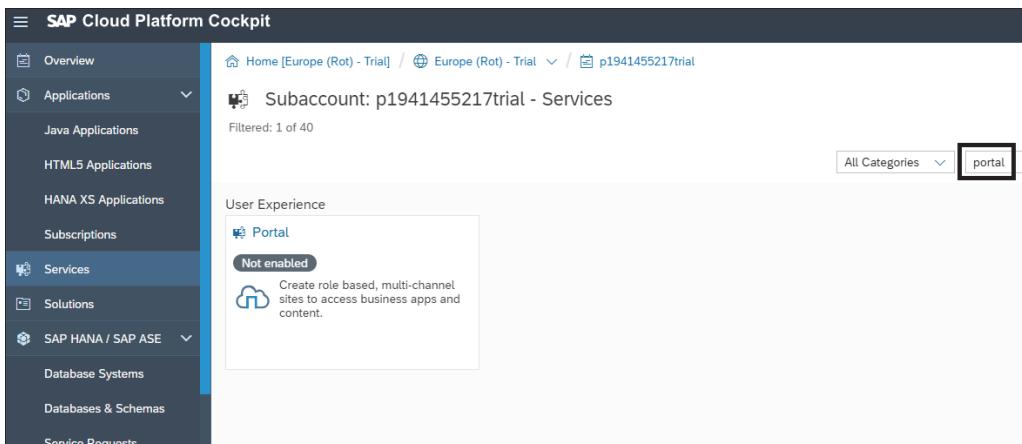
Remember to add this page to your bookmarks. Name it *SAP Web IDE full-stack version*. You'll use it a lot in this book.

### 2.1.3 Activate SAP Cloud Platform Portal Service

Although SAP Web IDE has an SAP Fiori launchpad sandbox environment for testing, it's better to test your code in a real environment. The SAP Fiori launchpad site in SAP Cloud Platform Portal will provide a cloud-based SAP Fiori launchpad environment that can be set up with very little effort.

By going through the following process, you'll learn how to activate the SAP Cloud Platform Portal service and how to create an SAP Fiori launchpad site for testing:

1. Open your SAP Cloud Platform trial index page from your bookmarks. Then select **Service** and search for "portal" via the search field. Click **Portal** under **User Experience** in the search results, as shown in [Figure 2.10](#).



**Figure 2.10** Locate Portal Service

2. The SAP Cloud Platform Portal service is disabled by default. Click **Enable** to activate it, as shown in [Figure 2.11](#).

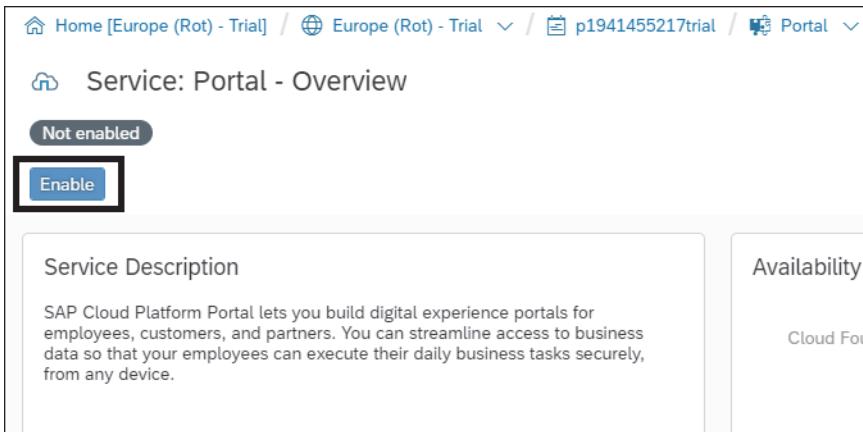


Figure 2.11 Enable SAP Cloud Platform Portal Service

3. After the service is enabled, click the **Configure Portal** link to configure authorization for SAP Cloud Platform Portal, as shown in [Figure 2.12](#).

The screenshot shows the SAP Cloud Platform Cockpit interface. The left sidebar has a dark theme with "Overview" selected. The main content area has a light theme. It shows the same "Service: Portal - Overview" page as Figure 2.11, but with "Enabled" instead of "Not enabled". The "Configure Portal" button in the "Take Action" section is highlighted with a red rectangle. The "Availability" section on the right shows "Cloud Foundry" and "NEO" with their respective status icons. The "Additional Resources" section on the right includes links to "Documentation", "Troubleshooting Guide", and "Portal Service Community".

Figure 2.12 Overview of Portal Service

4. Now you need to add your user to the predefined roles in the portal. Click **Roles**, then select the first role (**WEB\_CONTENT\_EDITOR**) and click **Assign** in the lower table, as shown in Figure 2.13.

The screenshot shows the SAP Cloud Platform Cockpit interface. The left sidebar has 'Destinations' and 'Roles' selected. The main content area is titled 'Service Configuration: Configure Portal - Roles'. It displays a table of roles with two entries:

|  | Name               | Type       |
|--|--------------------|------------|
|  | WEB_CONTENT_EDITOR | Predefined |
|  | TENANT_ADMIN       | Predefined |

Below the table, a note says 'WEB\_CONTENT\_EDITOR Predefined: Provisioned by the application'. At the bottom, there are buttons for 'Individual Users' (highlighted), 'Assign' (highlighted with a black box), and 'Unassign All'. A 'User ID' input field is present, and at the far right, there's a 'Actions' section.

**Figure 2.13** Roles Configuration for SAP Cloud Platform Portal

5. Enter your **User ID** and click **Assign**, as shown in Figure 2.14. Note that for a trial account, your user ID should be your account name without “trial” added to the end of it. You also need to note this user ID down for later reference.

The dialog box is titled 'Assign role "WEB\_CONTENT\_EDITOR" to user'. It has a 'User ID:' field containing 'pxxxxxxx' (with an asterisk) and a note below it stating 'Note: Changes will affect new sessions only.' At the bottom are 'Assign' and 'Cancel' buttons.

**Figure 2.14** Assign User ID to Role

### Example

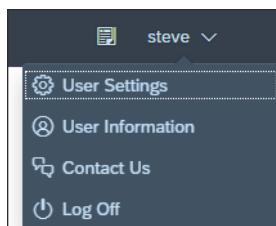
For instance, for account p12345678trial, the user ID should be p12345678.

6. Also assign your user to the **TENANT\_ADMIN** role.
7. Return to the portal service page by clicking **Portal** in the breadcrumb navigation, as shown in Figure 2.15.



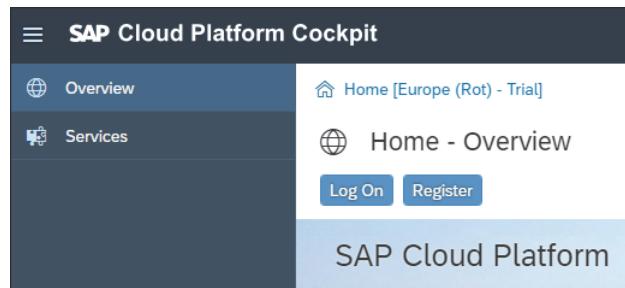
**Figure 2.15** Breadcrumb for Portal Service

8. To make the assignment valid, you need to log off and log on again. Click the arrow to the right of your user name and click **Log Off** in the drop-down menu, as shown in Figure 2.16.



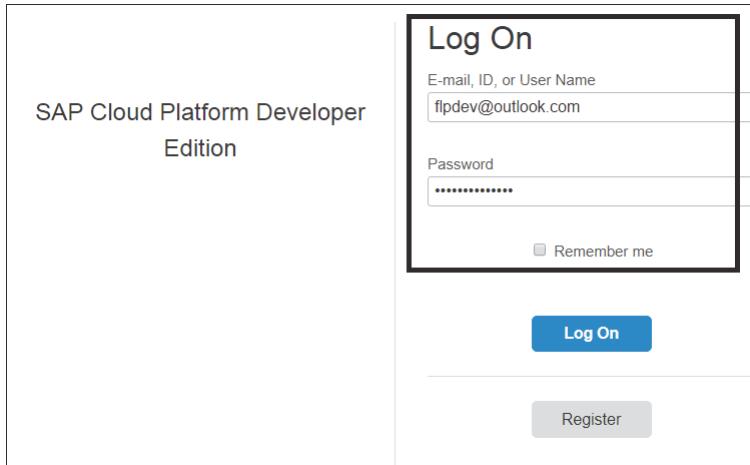
**Figure 2.16** Drop-Down for Logging Off

9. Log on again by clicking **Log On**, as shown in Figure 2.17.



**Figure 2.17** Home: Overview

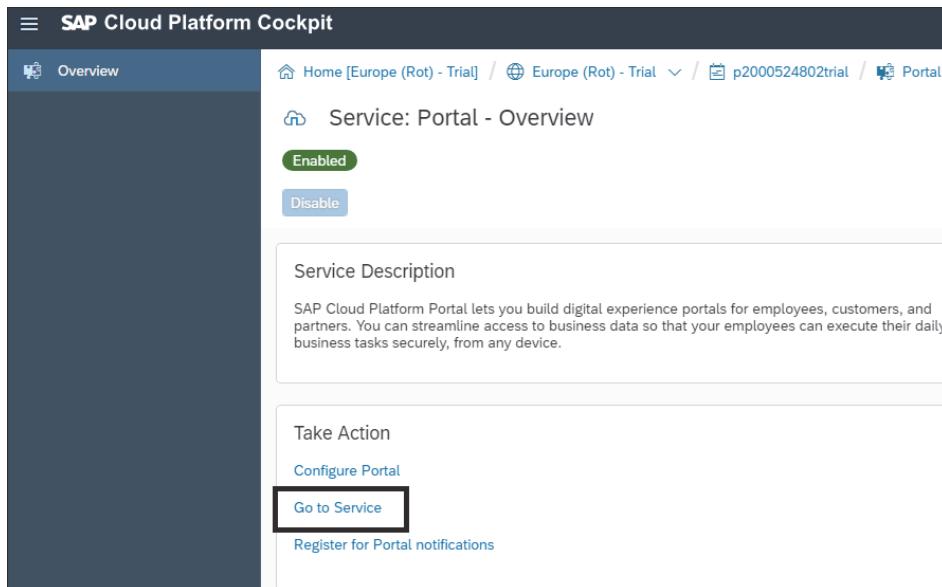
10. Input your user credentials and click **Log On**, as shown in [Figure 2.18](#).



The image shows the SAP Cloud Platform Developer Edition log-on page. It features a central 'Log On' form with fields for 'E-mail, ID, or User Name' containing 'flpdev@outlook.com' and 'Password' containing '\*\*\*\*\*'. There is a 'Remember me' checkbox and a 'Log On' button. Below the form is a 'Register' button.

**Figure 2.18** Log On Form for SAP Cloud Platform

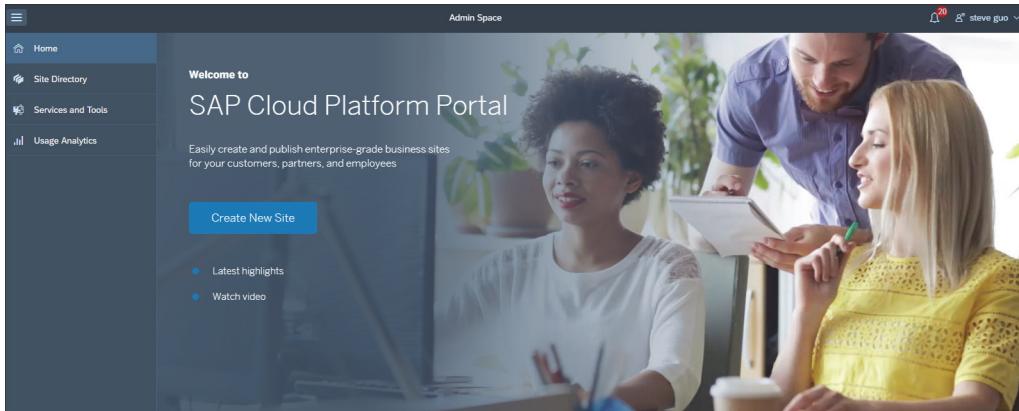
11. Navigate to the **Service: Portal—Overview** page again and click **Go to Service**, as shown in [Figure 2.19](#).



The image shows the SAP Cloud Platform Cockpit - Service: Portal - Overview page. The top navigation bar includes 'Overview', 'Home [Europe (Rot) - Trial]', 'Europe (Rot) - Trial', 'p2000524802trial', and 'Portal'. The main content area displays the service status as 'Enabled' with a 'Disable' button. Below this is a 'Service Description' section stating: 'SAP Cloud Platform Portal lets you build digital experience portals for employees, customers, and partners. You can streamline access to business data so that your employees can execute their daily business tasks securely, from any device.' At the bottom is a 'Take Action' section with 'Configure Portal', 'Go to Service' (which is highlighted with a black box), and 'Register for Portal notifications'.

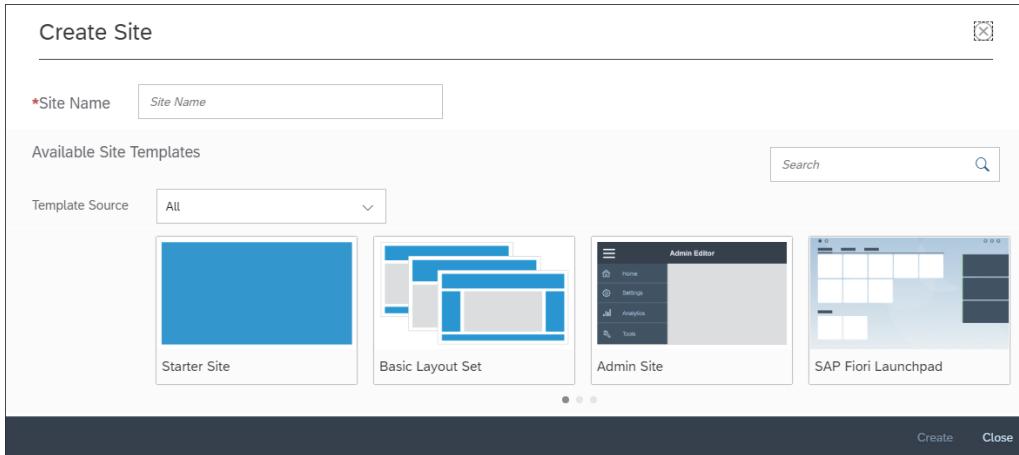
**Figure 2.19** Portal Service Overview Page

12. You'll enter the **Admin Space** index page of SAP Cloud Platform Portal. You need to create an SAP Cloud Platform Portal site by clicking on **Create New Site**, as shown in [Figure 2.20](#).



**Figure 2.20** SAP Cloud Platform Portal Admin Space Index Page

13. Provide a **Site Name**, and select **SAP Fiori Launchpad** as the site template, then click **Create**, as shown in [Figure 2.21](#).

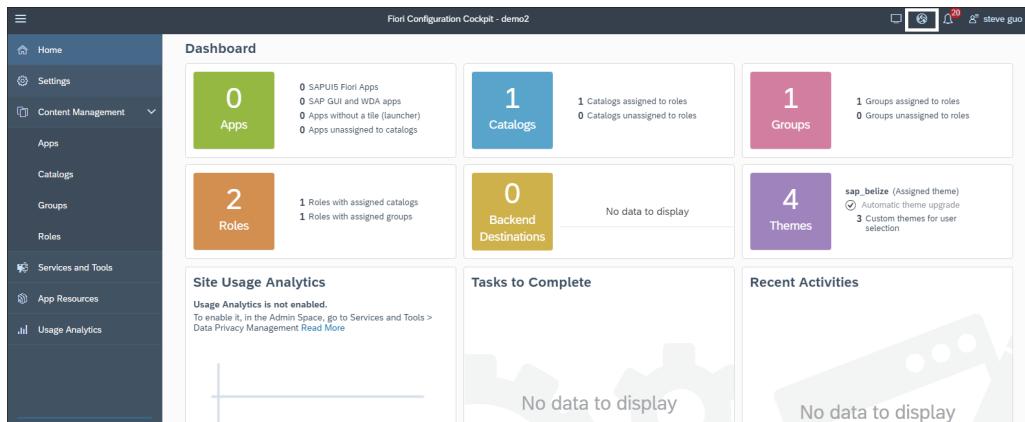


**Figure 2.21** Selecting Site Template

14. You'll enter the dashboard of your **SAP Fiori Configuration Cockpit**. For now, you don't need do anything in the dashboard. Just publish the site by clicking the globe icon at the end of header toolbar, as shown in [Figure 2.22](#).

### Note

Bookmark this page as *Manage SAP Fiori Launchpad site*. You'll access this page many times in this book.

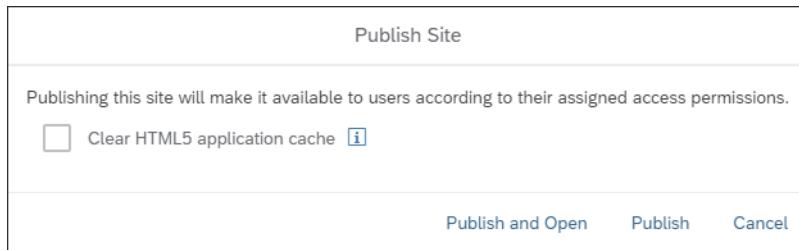


**Figure 2.22** Dashboard of SAP Fiori Configuration Cockpit

15. Click **Publish and Open**, as shown in [Figure 2.23](#).

### Note

Later in this book, you'll update your applications or configurations several times. You perform these tasks with the **Publish and Open** function as well. When updating settings, you need to check **Clear HTML5 Application Cache** if you want to see changes immediately.



**Figure 2.23** Publish Site

16. Now you'll see the initial state of your SAP Fiori launchpad site, as shown in [Figure 2.24](#). Nothing appears here now—but don't worry, you'll add your apps later in this book.

**Note**

Add this page to your bookmarks as *SAP Fiori launchpad on SAP Cloud Portal*. You'll use it to preview your applications later.



**Figure 2.24** Initial State of SAP Fiori Launchpad on SAP Cloud Platform

## 2.2 On-Premise Development Environment

The world is changing to a cloud-based world, but there's still a huge amount of on-premise implementation, especially for SAP's products. The development process and the source code are no different between cloud and on-premise; that's one of the purposes of SAP Fiori launchpad. But the deployment process is different on different platforms. As a development consultant, you need to know this process in the on-premise system as well.

In this section, we'll guide you through setting up a single-user development environment for SAP Fiori launchpad based on SAP NetWeaver AS ABAP and connecting it to the cloud-based SAP Web IDE full-stack version.

**Tip**

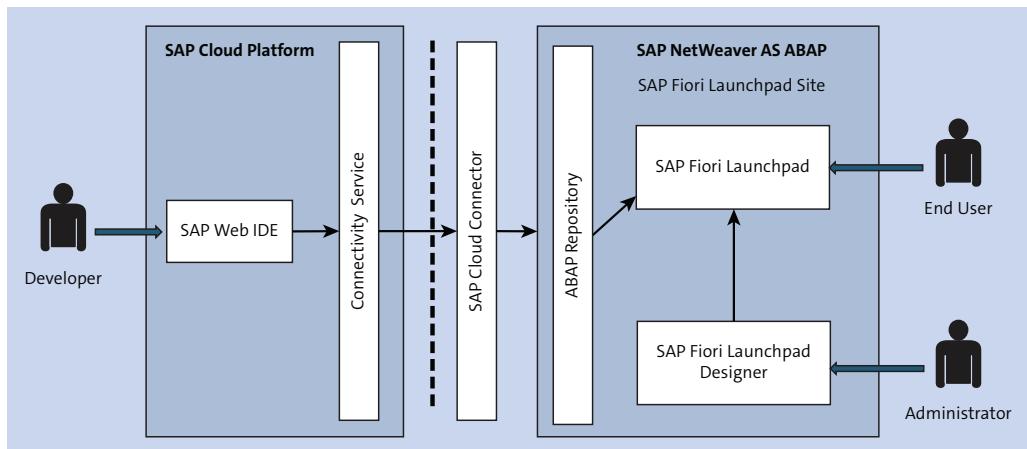
This environment is used in several sections of [Chapter 6](#) and [Chapter 7](#). It may take up to half a day to set up this environment. You can skip this section if you like and return to them when you're reading the associated sections.

### Warning

All software and processes in this guide are based on developer agreements and are for the purposes of creating an environment for self-learning and testing for developers who do not have access to commercially licensed software and are not specialized in system administration. Do not follow this guide if you want to build a production environment; it doesn't include the necessary security setup.

#### 2.2.1 Prerequisites

Before we jump into installation, we want to give you an overview of the architecture of the environment you'll be setting up, as shown in [Figure 2.25](#).



**Figure 2.25** Architecture of On-Premise Development Environment

This architecture contains the following parts:

- **SAP Web IDE full-stack**  
As in the cloud architecture, you use the cloud-based IDE for development.
- **SAP NetWeaver AS ABAP 7.52 developer version**  
The ABAP backend server.
- **VirtualBox**  
The virtual machine host operating system for SAP NetWeaver AS ABAP.
- **openSUSE Linux**  
Operating system for NetWeaver AS ABAP.

■ **SAP Cloud Connector**

Component for setting up a secure connection between your local system and SAP Cloud Platform.

Before going through this guide, you need to check that you meet the following hardware requirements:

- x86\_64 processor-based hardware
- Required: at least 4 GB of RAM plus about 8 GB of swap space
- Recommended: at least 8 GB of RAM plus about 8 GB of swap space
- About 100 GB of free disk space for server installation
- About 2 GB of free disk space for client installation

In preparation, download all the files for your installation, as discussed in the following sections.

### Download Oracle VirtualBox

Get the VirtualBox installation file from <https://www.virtualbox.org/wiki/Downloads> ([Figure 2.26](#)). You need VirtualBox to create a virtual machine to host your ABAP server.



**Figure 2.26** Download Page for Oracle VirtualBox

## Download openSUSE

We use openSUSE as the operating system. You can get it from <https://software.opensuse.org/distributions/leap> (Figure 2.27). Pick one of the links under **x86\_64** • DVD Image.

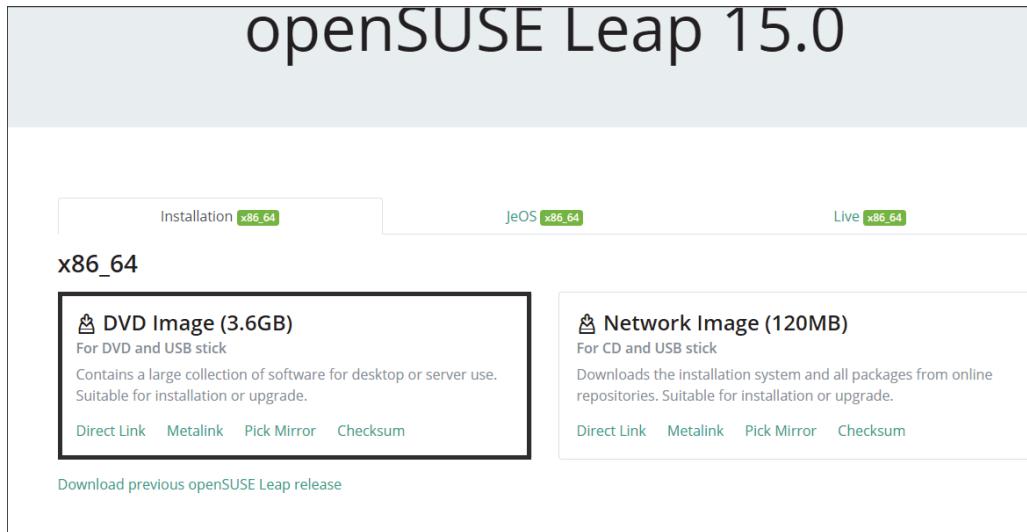


Figure 2.27 Download Page for openSUSE

## Download SAP NetWeaver AS ABAP 7.52 SP 01 Developer Version

SAP NetWeaver AS ABAP 7.52 SP 01 developer version is available for you to download, learn, and test. You can find it at [tools.hana.ondemand.com](https://tools.hana.ondemand.com) and download it as follows:

1. The download link can be accessed at <https://tools.hana.ondemand.com/#abap>. You'll find the link under **SAP NetWeaver AS ABAP Developer Edition**, as shown in Figure 2.28.
2. A blog page from SAP Community will appear. Scroll down to find the SAP Store link, as shown in Figure 2.29. The full link is <https://www.sap.com/developer/trials-downloads/additional-downloads/sap-netweaver-as-abap-developer-edition-sp01-7-52-15510.html>.

To install the front-end component of ADT, proceed as follows:

1. Get an installation of [Eclipse Photon](#) (e.g. [Eclipse IDE for Java Developers](#))
2. In Eclipse, choose in the menu bar **Help > Install New Software...**
3. For Eclipse Photon (4.8), add the URL <https://tools.hana.ondemand.com/photon>
4. Press **Enter** to display the available features.
5. Select **ABAP Development Tools** and choose **Next**.
6. On the next wizard page, you get an overview of the features to be installed. Choose **Next**.
7. Confirm the license agreements and choose **Finish** to start the installation.

**SAP NetWeaver AS ABAP Developer Edition**

If you want to try out the tools without having access to an ABAP Server, you can run your own one using the "SAP NetWeaver AS ABAP Developer Edition". You can download the Trial Version of [7.50 SP02](#) and [7.52 SP01](#) from the SAP Store.

**Enhanced SAP Change and Transport System (CTS+)**

With the enhanced Change and Transport System (CTS+) you can transport non-ABAP objects via the on premise ABAP server to a CTS+ transport request. With this ctsattach tool you are able to automate this attachment step or trigger it via command request can be triggered (without the need of the so called Transport Organizer Web UI). One use case for the usage of CTS+ is to transport objects between ABAP servers. For further details around CTS+, please see: [link](#).

Details:  
Further information can be found in the built in help of the tool by using the command 'ctsattach help'.  
In order to run the tool a Java 7 or higher JRE installation is required and the JAVA\_HOME environment variable must be set.

Figure 2.28 Download Page for ABAP-Related Developer Resources

services and SAPUI5 or you can get an overview on SAP's client/server technology.

Just like the versions we offer in the Cloud, this developer edition is preconfigured to run to run the Database Feature Gallery and the Enterprise Procurement Model programming examples out of the box.

It contains:

- SAP AS ABAP 7.52 SP01
- SAP GUI for the Java 7.5 and SAP GUI for Windows 7.50
- SAP Sybase ASE 16.0.1

The actual download (in the form of several .rar files), along with more information is available from the SAPStore: [SAP AS ABAP 7.52 SP01, developer edition](#).

Figure 2.29 Blog Post for SAP NetWeaver AS ABAP Developer Version

3. There are 10 RAR files to be downloaded, as shown in [Figure 2.30](#).

The screenshot shows the SAP website for the SAP NetWeaver AS ABAP 7.52 SP 01 edition. The top navigation bar includes links for Products, Industries, Support, Training, Community, Developer, Partner, About, a search icon, and user profile icons. Below the navigation is a dark header bar with the text "Additional Software". Underneath is a yellow bar with "Back to Finder". The main content area features a large title "SAP NetWeaver AS ABAP Developer Edition 7.52 SP01". A descriptive paragraph explains the application server's features, mentioning SAP Fiori Launchpad, SAP Cloud Connector, SAP Java Virtual Machine, and various SAP infrastructure components. Below the text is a list of ten RAR files: TD752SP01.part01.rar through TD752SP01.part10.rar, each preceded by a right-pointing arrow. The entire page has a clean, professional design with a white background.

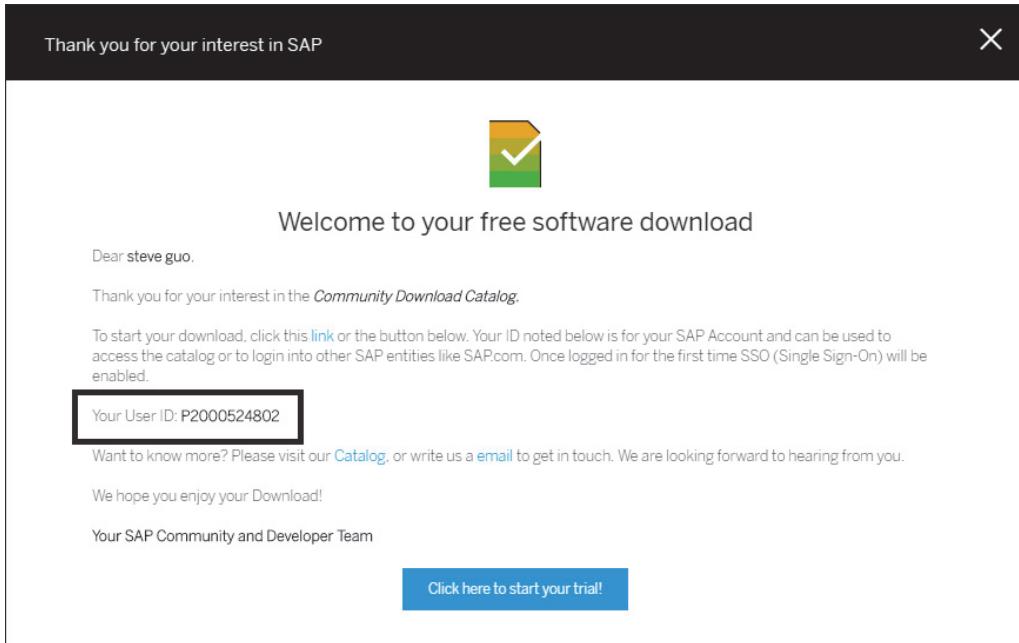
Figure 2.30 Download Page for SAP NetWeaver AS ABAP 7.52 SP 01

4. After clicking each file, you need to accept an agreement. Check the checkbox and click **Submit**, as shown in [Figure 2.31](#).

The pop-up window has a black header bar with the text "Thank you for your interest in SAP" and a close button (X) in the top right corner. The main content area contains a small icon of a clipboard with checkmarks. Below the icon is the text "End user license agreement". Underneath is a checkbox followed by the text "I have read and I accept the SAP Terms of Use". At the bottom is a blue "Submit" button. The overall design is simple and functional.

Figure 2.31 Pop-Up when Downloading Files for SAP NetWeaver AS ABAP 7.52 SP 01 Developer Version

5. Click **Click Here to Start Your Trial** and download the file. You may need to enter your user ID (displayed in the pop-up) and password again, as shown in Figure 2.32.



**Figure 2.32** Download Link for SAP NetWeaver AS ABAP 7.52 SP 01 Developer Version

6. After downloading all 10 files, you need to extract them.

### Download SAP Cloud Connector and SAP JVM

Enter <https://tools.hana.ondemand.com> again, and switch to the **Cloud** tab (see Figure 2.33). Scroll down to find **SAP Cloud Connector** and **SAP JVM**. You can choose the one that matches your system.

---

#### Tip

You can also use your own Java virtual machine; the required version is 7 or 8.

| Operating System*   | Architecture | Version | File Size | Download                                       |
|---------------------|--------------|---------|-----------|--|
| Linux               | x86_64       | 2.11.2  | 74.3 MB   | <a href="#">sapcc-2.11.2-linux-x64.zip</a>     |
| Linux (Portable)    | x86_64       | 2.11.2  | 77.7 MB   | <a href="#">sapcc-2.11.2-linux-x64.tar.gz</a>  |
| Mac OS X (Portable) | x86_64       | 2.11.2  | 77.7 MB   | <a href="#">sapcc-2.11.2-macosx-x64.tar.gz</a> |
| Windows             | x86_64       | 2.11.2  | 79.2 MB   | <a href="#">sapcc-2.11.2-windows-x64.msi</a>   |
| Windows (Portable)  | x86_64       | 2.11.2  | 77.2 MB   | <a href="#">sapcc-2.11.2-windows-x64.zip</a>   |

| Operating System* | Architecture | Version | File Size | Download                                       |
|-------------------|--------------|---------|-----------|--|
| Linux             | x86_64       | 7.1.059 | 113.8 MB  | <a href="#">sapjvm-7.1.059-linux-x64.zip</a>   |
| Linux             | x86_64       | 7.1.059 | 113.0 MB  | <a href="#">sapjvm-7.1.059-linux-x64.rpm</a>   |
| Linux             | x86_64       | 8.1.044 | 117.5 MB  | <a href="#">sapjvm-8.1.044-linux-x64.zip</a>   |
| Linux             | x86_64       | 8.1.044 | 119.4 MB  | <a href="#">sapjvm-8.1.044-linux-x64.rpm</a>   |
| Mac OS X          | x86_64       | 7.1.059 | 115.6 MB  | <a href="#">sapjvm-7.1.059-macosx-x64.zip</a>  |
| Mac OS X          | x86_64       | 8.1.044 | 119.4 MB  | <a href="#">sapjvm-8.1.044-macosx-x64.zip</a>  |
| Windows           | x86_64       | 7.1.059 | 130.0 MB  | <a href="#">sapjvm-7.1.059-windows-x64.zip</a> |
| Windows           | x86_64       | 8.1.044 | 136.3 MB  | <a href="#">sapjvm-8.1.044-windows-x64.zip</a> |

Figure 2.33 Download Page for Cloud Tools

## 2.2.2 Create a Virtual Machine

The first thing to do after downloading all files is to install Oracle VirtualBox. The installation process is simple: double-click the binary file, click **Next** a few times, and then click **Finish**.

After VirtualBox has been installed, you need to create a virtual machine for your development according to the following procedure:

1. Click **New** to create a new virtual machine, as shown in [Figure 2.34](#).

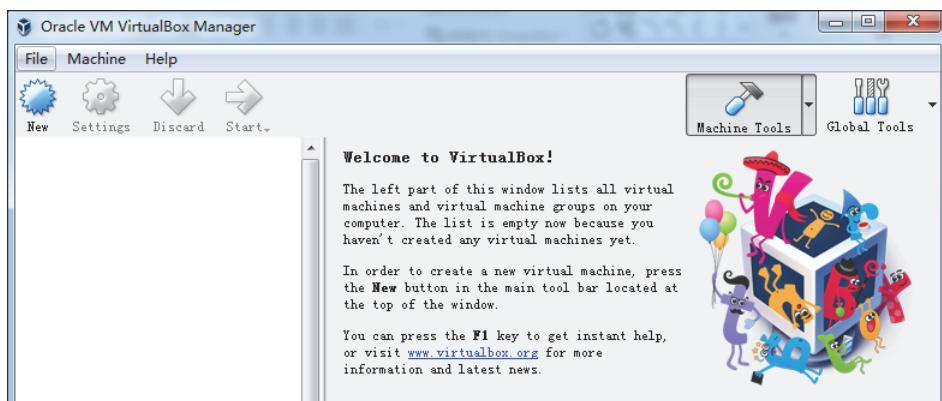
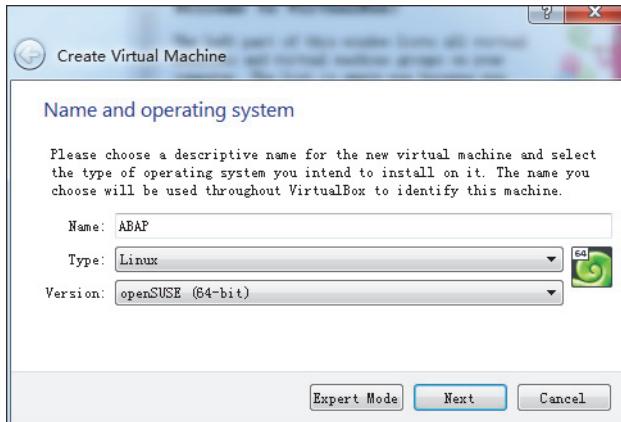


Figure 2.34 Startup UI for Oracle VirtualBox

2. Give the virtual machine a name (e.g., ABAP). For **Type**, choose **Linux**; for **Version** choose **openSUSE (64-bit)**. Then click **Next**, as shown in [Figure 2.35](#).

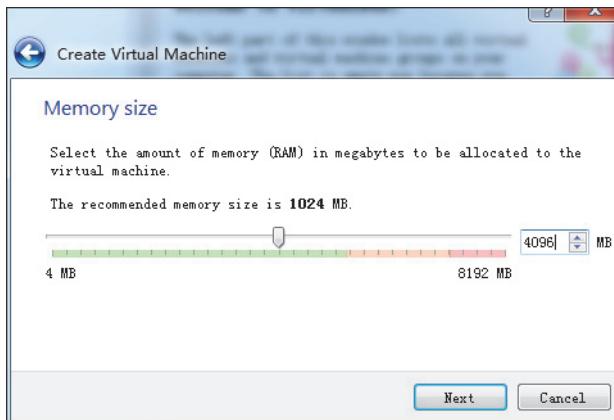


**Figure 2.35** Name and Operating System

3. Set the memory size using the slider, then click **Next**, as shown in [Figure 2.36](#).

### Note

The required size is 4096 MB and the recommend size is 8192 MB. Please also consider you should at least leave 4096 MB memory for your operating system.



**Figure 2.36** Memory Size

4. Choose **Create a Virtual Hard Disk Now** and then click **Create**, as shown in Figure 2.37.

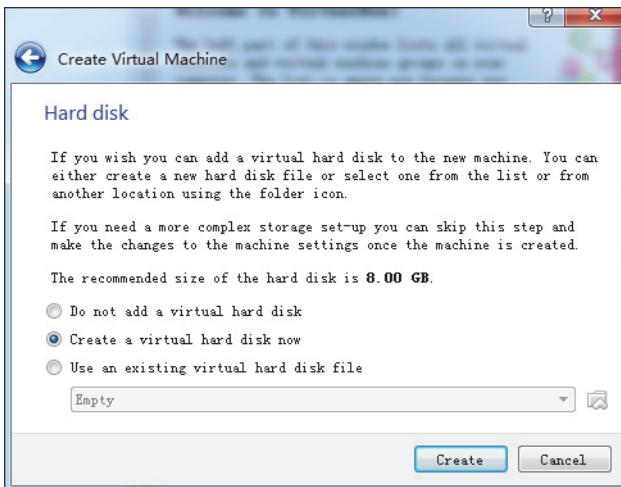


Figure 2.37 Hard Disk

5. Choose **VHD** as the hard disk file type and then click **Next**, as shown in Figure 2.38.

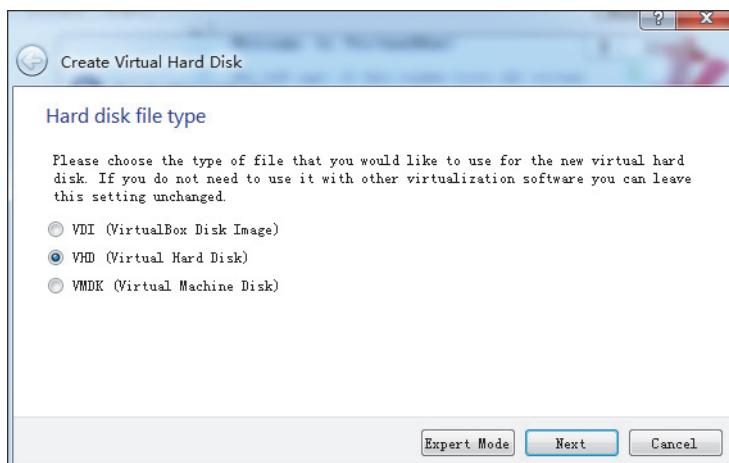


Figure 2.38 Hard Disk File Type

6. Choose **Dynamically Allocated** for the storage file and then click **Next**, as shown in Figure 2.39.

### Note

This option is important: it makes the VM consume less disk space.

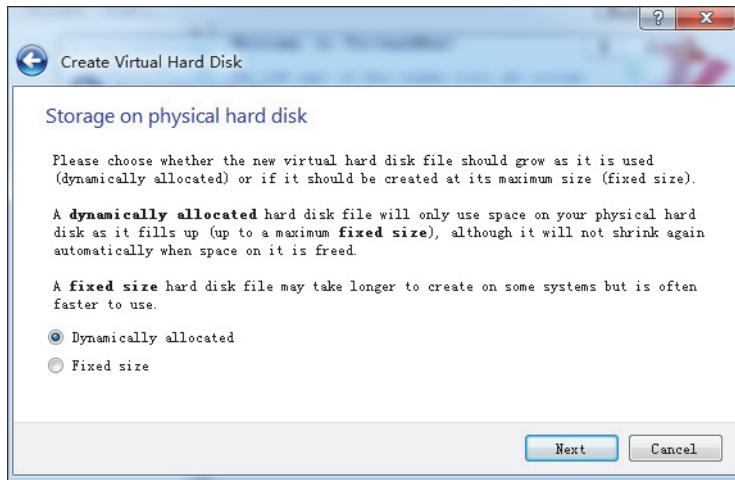


Figure 2.39 Storage on Physical Hard Disk

- Set the storage location and size for the virtual hard disk; it should be at least 100 GB. Then click **Create**, as shown in [Figure 2.40](#).

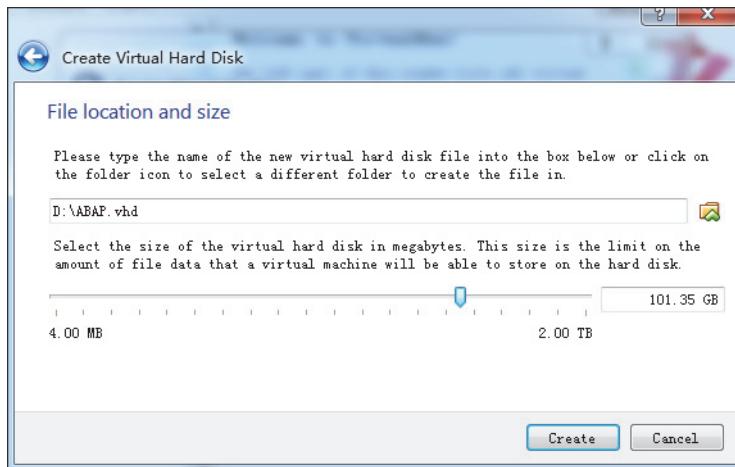


Figure 2.40 File Location and Size

**Tip**

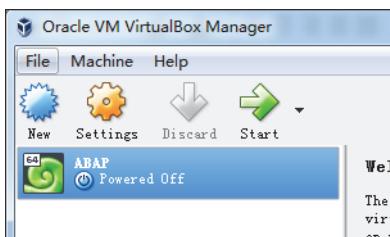
If the storage location is an SSD, then your ABAP system will run much faster.

Now you've successfully created a virtual machine.

### 2.2.3 Install the Operating System

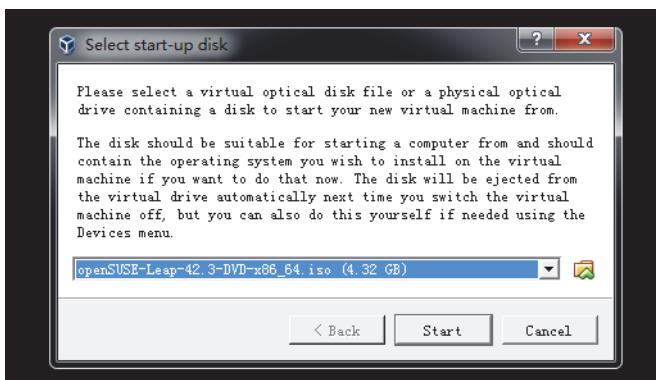
In this section, you'll install openSUSE Linux on the virtual machine, as follows:

1. Click **Start** to start your virtual machine, as shown in [Figure 2.41](#).



**Figure 2.41** Start Virtual Machine

2. Click the folder icon to choose your openSUSE Linux installation disk file and then click **Start**, as shown in [Figure 2.42](#).



**Figure 2.42** Choose Boot Disk File

3. Select **Installation** in the boot menu using the down arrow on your keyboard and then press **Enter**, as shown in [Figure 2.43](#).

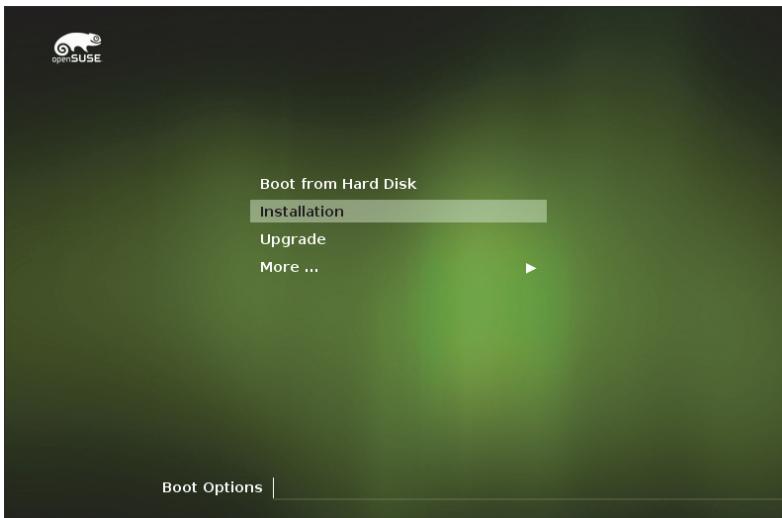


Figure 2.43 Boot Menu for openSUSE Linux Installation Disk

4. Keep the language settings as default and click **Next**, as shown in [Figure 2.44](#).

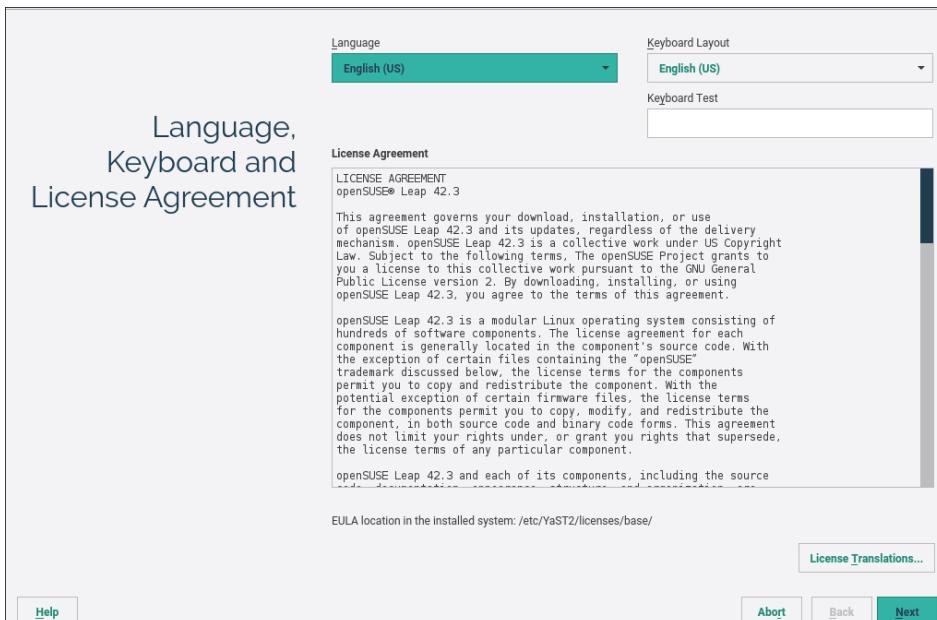
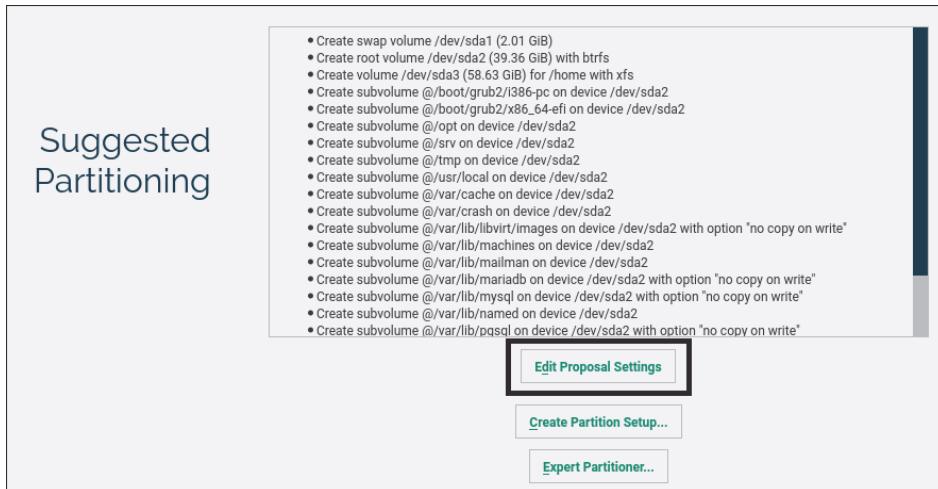


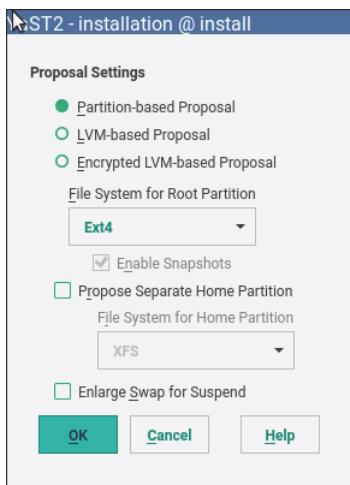
Figure 2.44 Language, Keyboard, and License Agreement

5. Choose **Edit Proposal Settings** to change the predefined partition settings, as shown in [Figure 2.45](#).



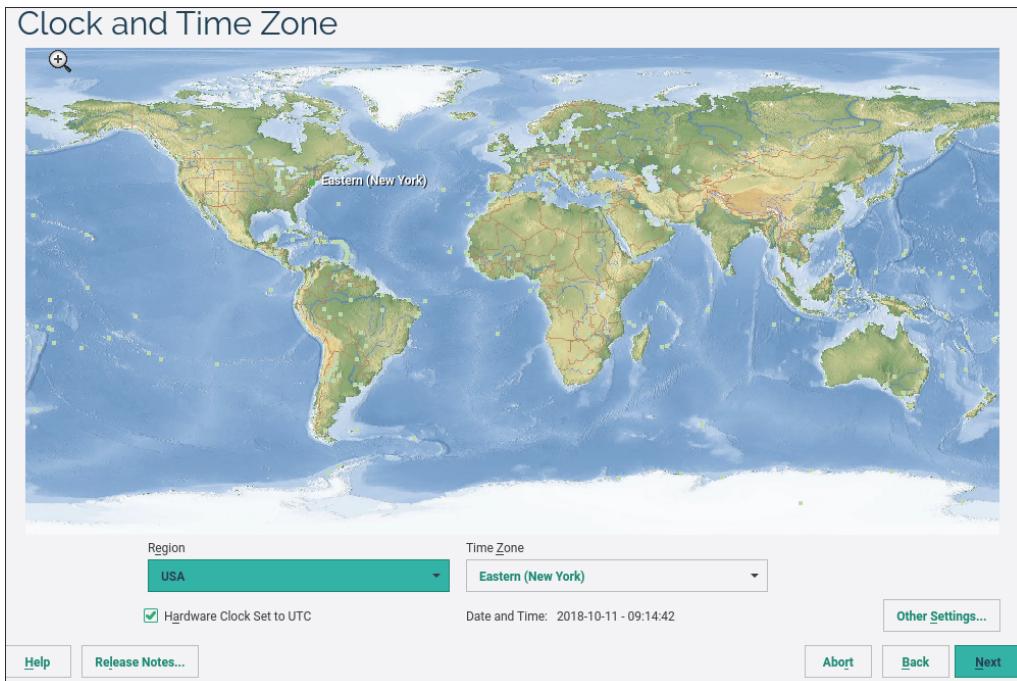
**Figure 2.45** Suggested Partitioning

6. Choose **Partition-Based Proposal** and then select **Ext4** as the **File System for Root Partition**. Uncheck **Propose Separate Home Partition** and **Enlarge Swap for Suspend**. Then click **OK**, as shown in [Figure 2.46](#). On the next page, click **Next** to continue.



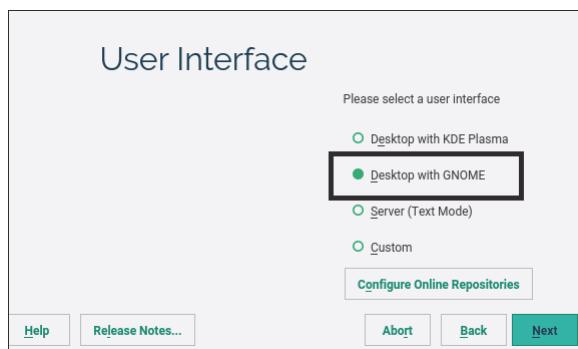
**Figure 2.46** Partition Settings

7. Choose your clock and time zone settings and click **Next**, as shown in [Figure 2.47](#).  
You can also leave all fields set to the default values.



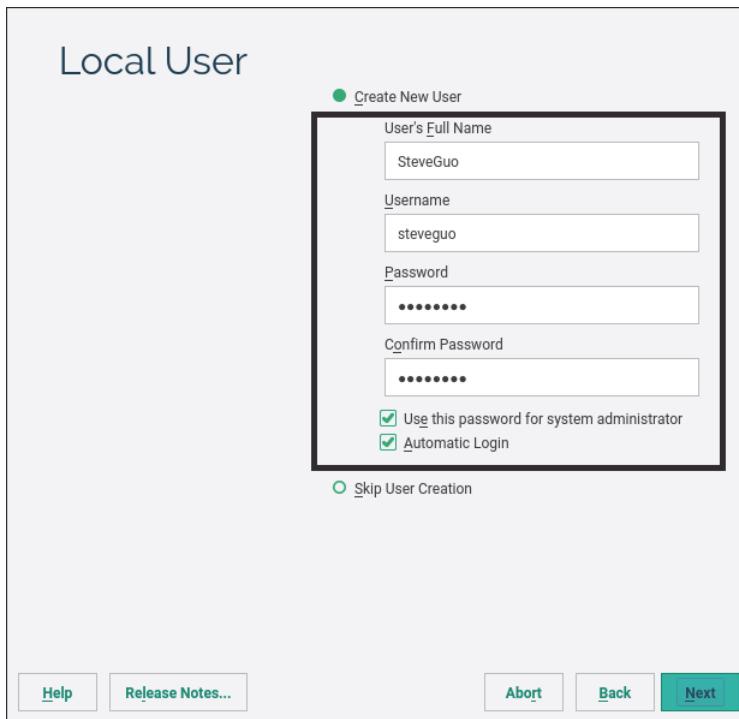
**Figure 2.47** Clock and Time Zone Settings

8. Select **Desktop with GNOME** as the user interface and click **Next**, as shown in [Figure 2.48](#).



**Figure 2.48** User Interface

9. Create a local user with a password, as shown in [Figure 2.49](#). Check **Use the Password for System Administrator** and **Automatic Login**. Then click **Next**.



**Figure 2.49** Local User

10. Review all the settings. Click the **Disable** link after **Firewall Will Be Enabled** in the **Firewall and SSH** section to disable the firewall. Then click the **Enable** link after **SSH Service Will Be Disabled** in the same section to enable SSH, as shown in [Figure 2.50](#). Then click **Install**.
11. On the next screen, click **Install** to start the installation process, as shown in [Figure 2.51](#). It should take about 20 minutes.

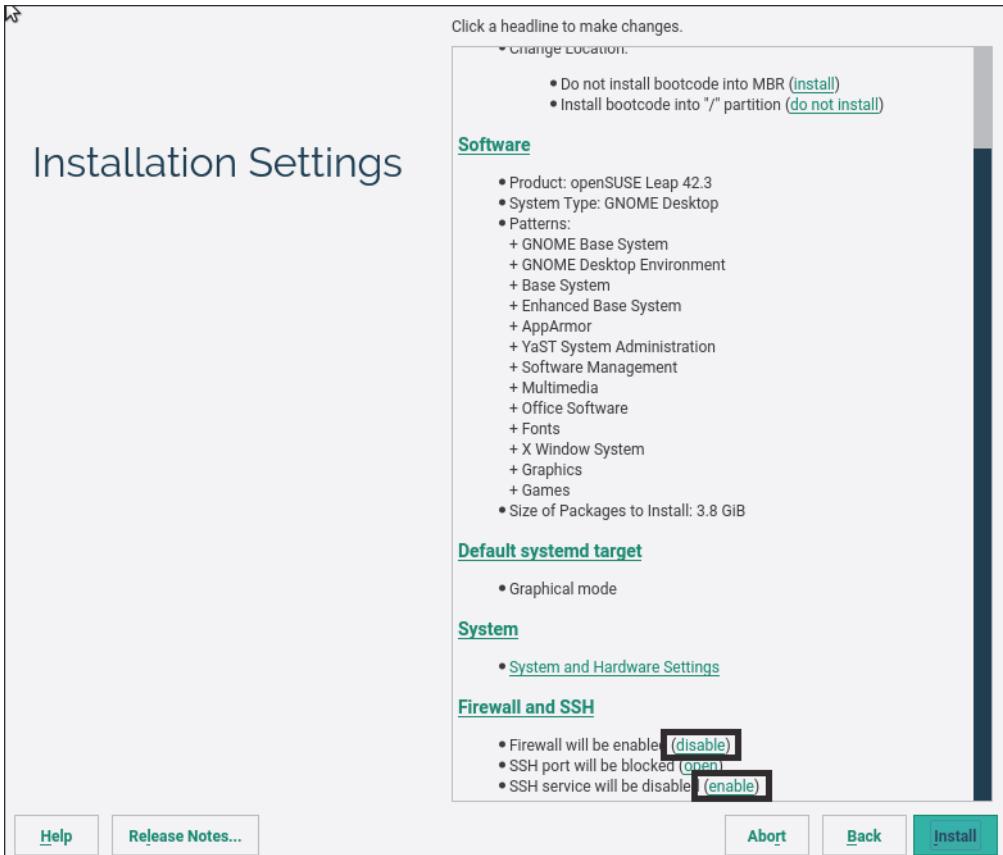


Figure 2.50 Installation Settings

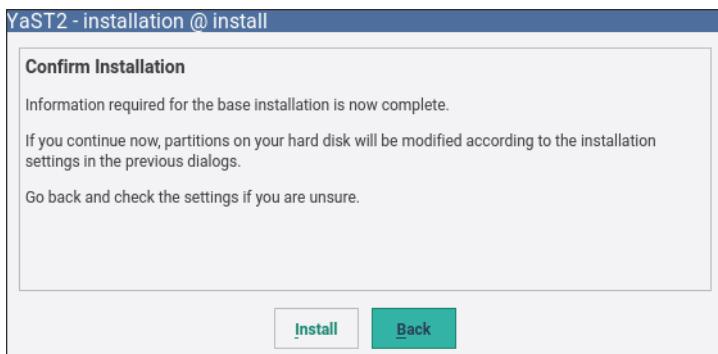


Figure 2.51 Confirm Installation

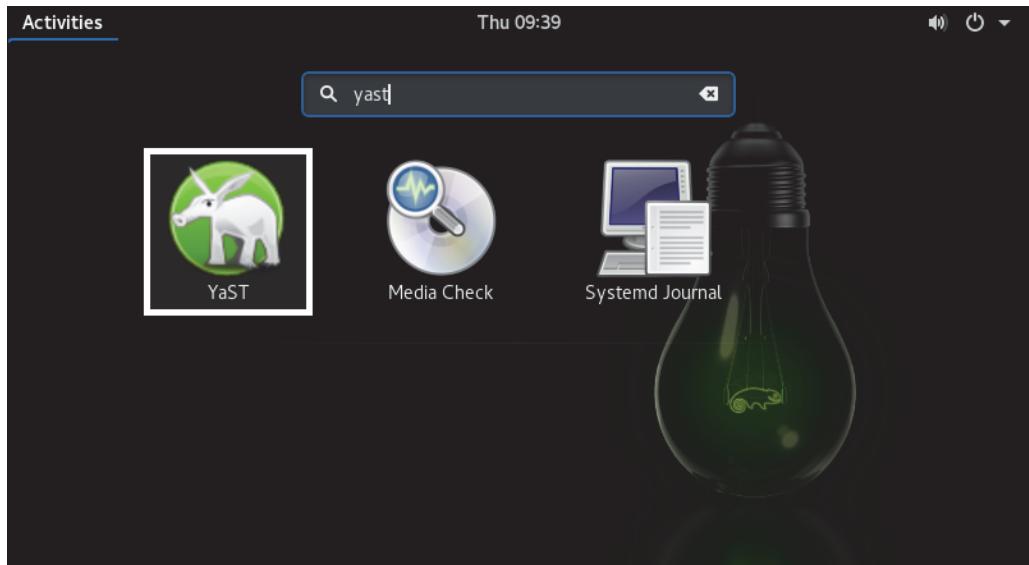
## 2.2.4 Prepare the Operating System for SAP NetWeaver AS ABAP

After installation of the operation system has been performed, the virtual machine will restart and finally enter the GNOME index page. Before installing SAP NetWeaver AS ABAP 7.52 developer version, you need to perform a set of operations to make the system ready for installation, each of which we'll discuss in the following subsections.

### Install uuidd

uuidd is a Linux-based service to generate universally unique identifiers (UUIDs). SAP NetWeaver requires this component. To install uuidd, follow these steps:

1. Click **Activities** and enter “yast” in the search field. Click **YaST** in the search results, as shown in [Figure 2.52](#).



**Figure 2.52** Enter Application in openSUSE Linux

2. Enter your Linux system password and click **Continue**, as shown in [Figure 2.53](#).

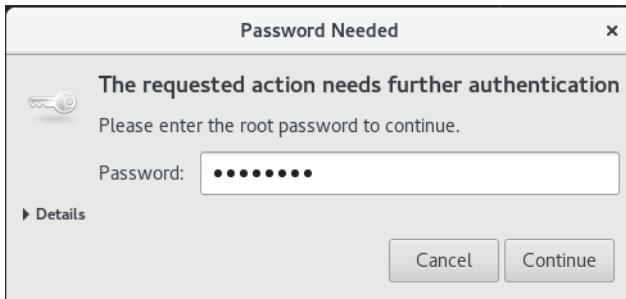


Figure 2.53 Provide Password

3. Click **Software** and then click **Online Update**, as shown in [Figure 2.54](#).

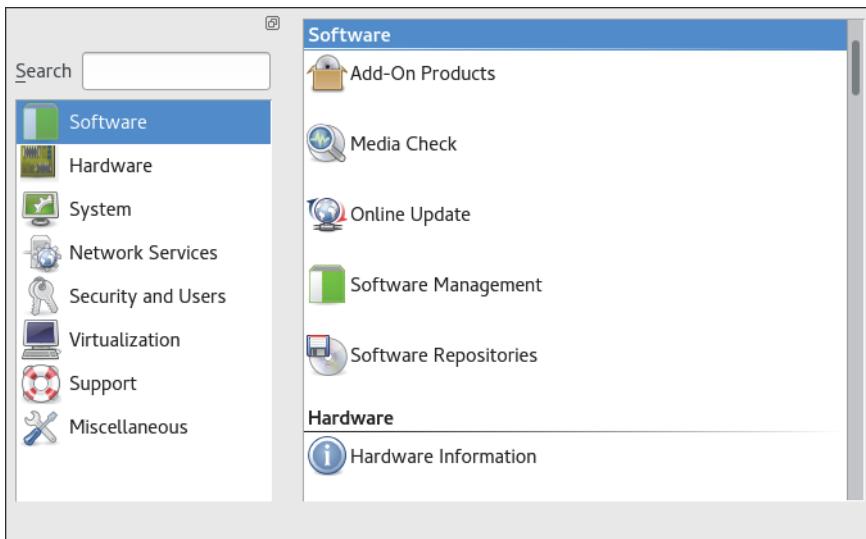


Figure 2.54 Yast UI

4. Enter “**uuidd**” in the search field and then check the checkbox to the left of **uuidd** in the search results. Finally, click **Accept**, as shown in [Figure 2.55](#).
5. Wait a few minutes and **uuidd** will install successfully. Click **OK** to close the pop-up shown in [Figure 2.56](#). Then click **Software** and **Online Update** again to install the next software.

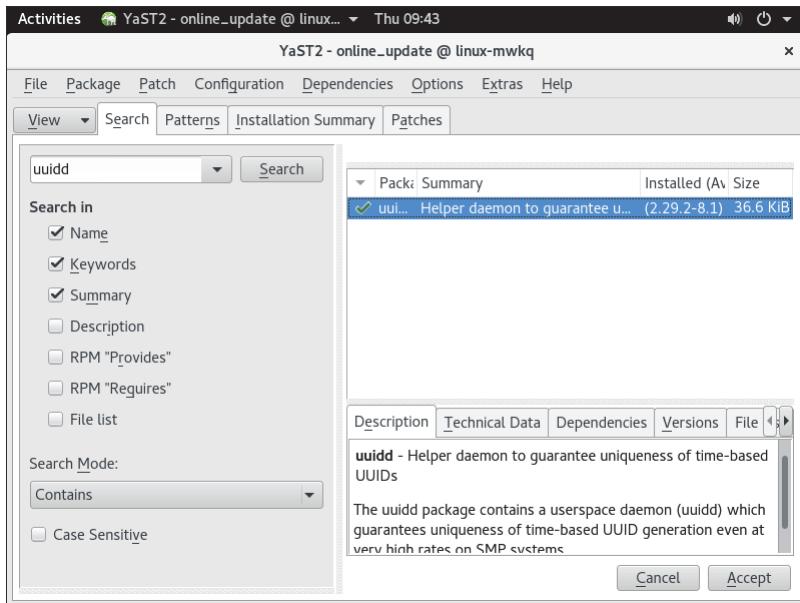


Figure 2.55 Install uidd

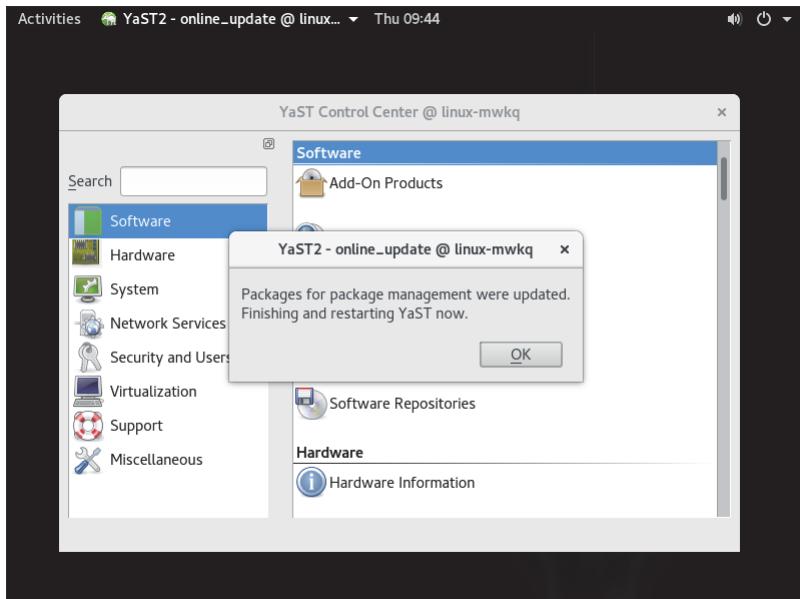
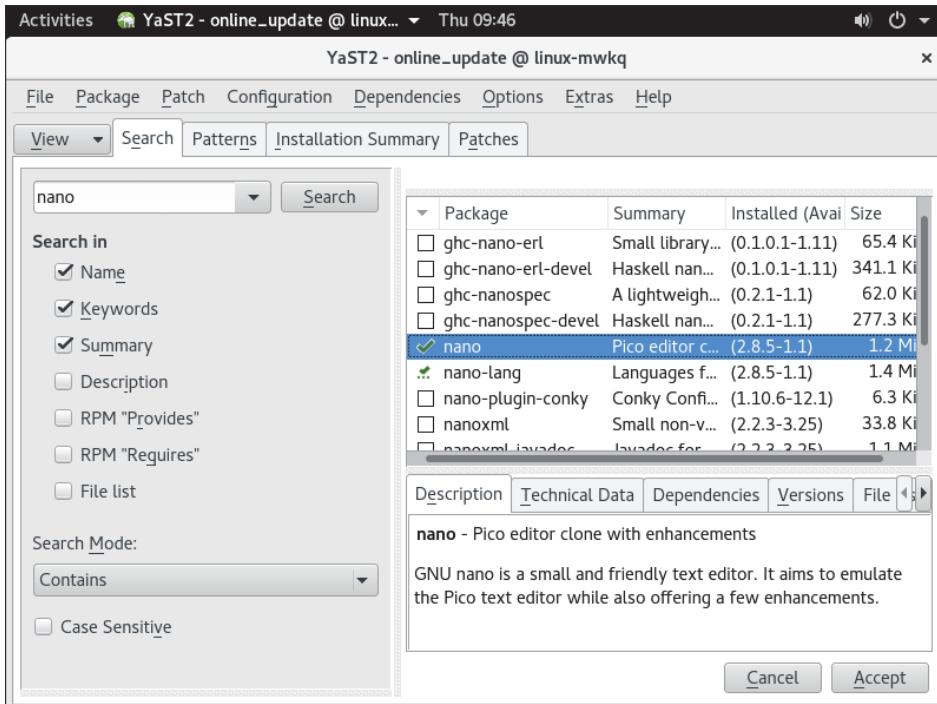


Figure 2.56 Successful Installation of uidd

### Install nano

The next step is to install nano, a text editor used to edit several text fields. To begin, follow these steps:

1. In the search field, enter “nano”, select **nano** in the results list, and then click **Accept**, as shown in [Figure 2.57](#).



**Figure 2.57** Install nano

2. In the confirmation dialog, click **Continue**, as shown in [Figure 2.58](#). Wait a few minutes, and nano will be installed successfully.

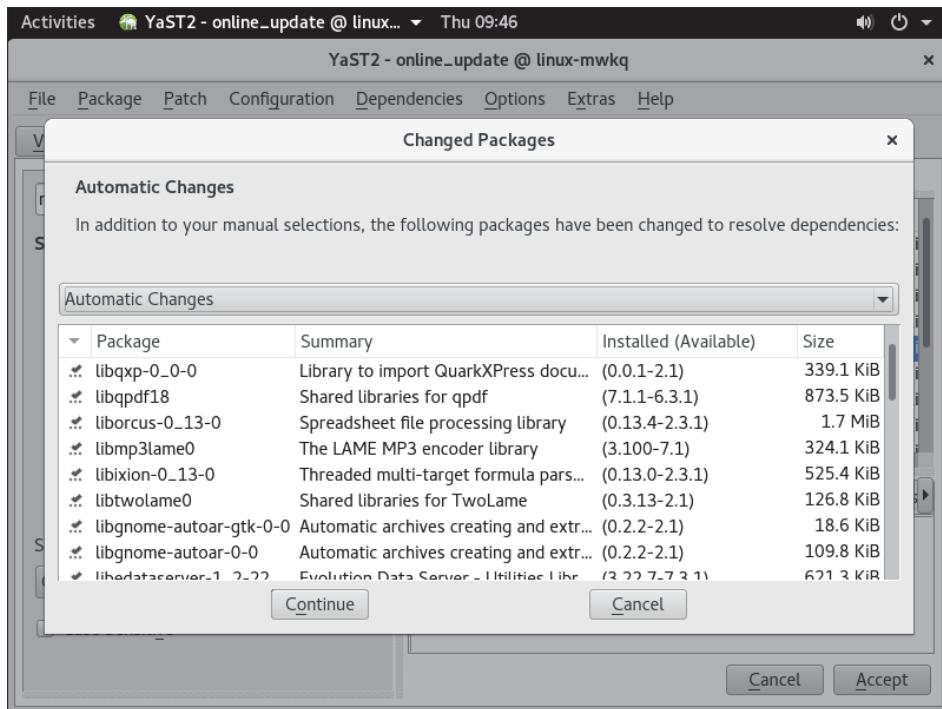


Figure 2.58 Confirmation Dialog when Installing nano

### Set Up a Shared Folder

Now you need to let your virtual host have access to your SAP NetWeaver installation files by setting up a shared folder, as follows:

1. Switch back to VirtualBox and click **Settings**, as shown in Figure 2.59.

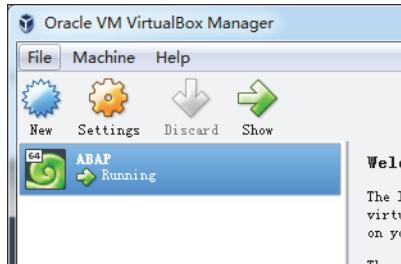


Figure 2.59 VirtualBox: Settings

2. Choose **Shared Folders**, then click the **Add** button on the right side. In the pop-up window, locate your install file path, name it “sapinstall”, then check the checkboxes to the left of **Auto-mount** and **Make Permanent**, as shown in Figure 2.60.

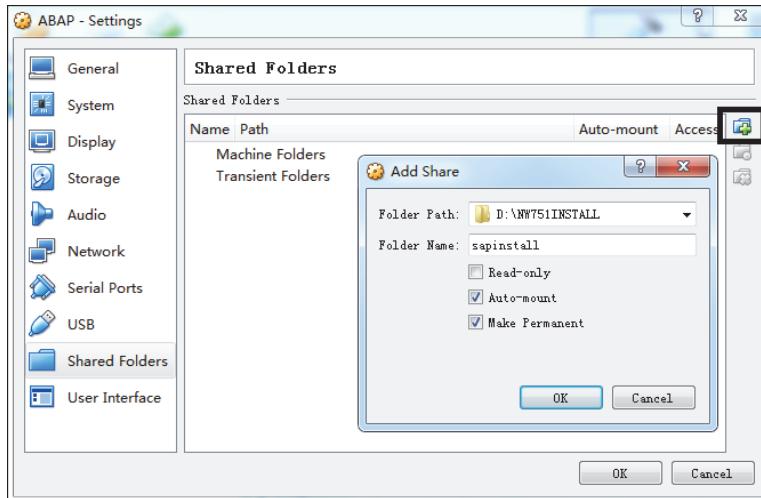


Figure 2.60 Adding Shared Folders

3. Figure 2.61 shows the results after you add the folder. Click **OK** to confirm this page.

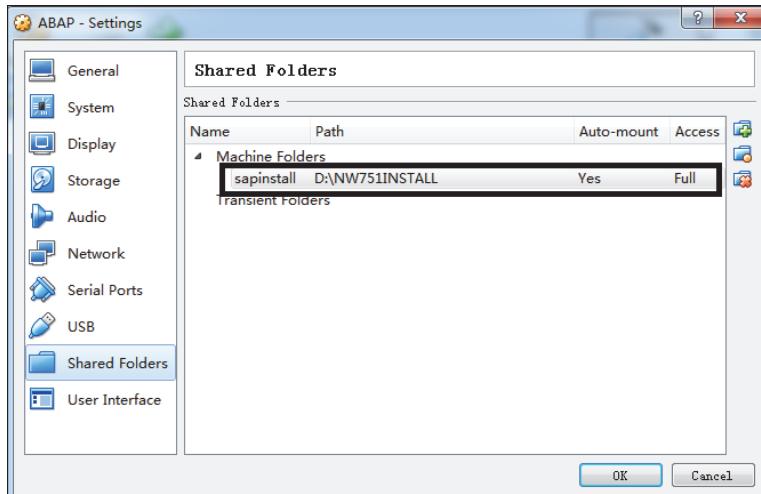
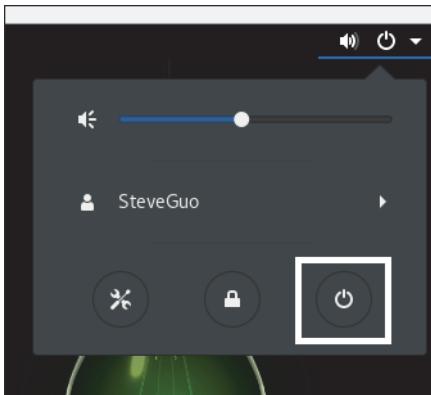


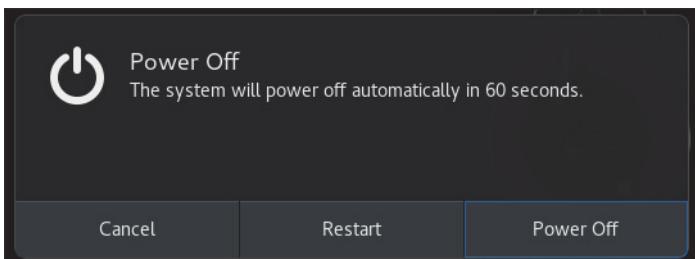
Figure 2.61 Result of Adding Shared Folder

4. Switch back to your Linux virtual machine, click the power icon at the top-right corner, and then click the round button with a power icon, as shown in [Figure 2.62](#).



**Figure 2.62** Reboot Menu for Virtual Machine

5. In the pop-up, choose **Restart** to restart your virtual machine, as shown in [Figure 2.63](#).



**Figure 2.63** Confirmation Dialog for Reboot

### Checking the `uuid` Service

In this section, you'll check if the `uuid` service is installed properly by following these steps:

1. Click **Activates**, enter “xterm” in the search field, and click **XTerm** in the search results, as shown in [Figure 2.64](#).

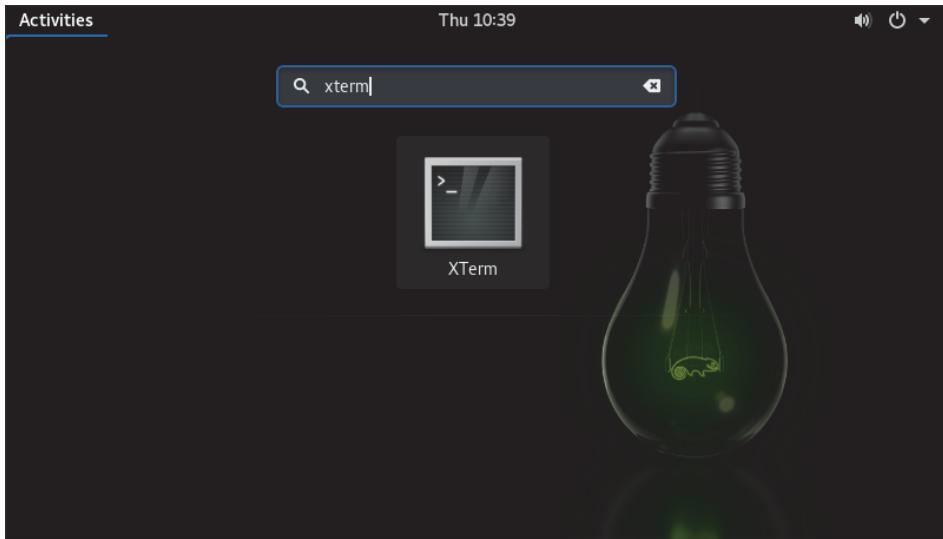


Figure 2.64 Searching for XTerm

2. Enter the following commands if the result contains `uidd`, as it does in [Figure 2.65](#):

- `sudo service uidd start`
- `sudo service --status-all | grep uidd`

The service is now installed successfully.

```
steveguo@linux-mukq:~$ sudo service uidd start
steveguo@linux-mukq:~$ sudo service --status-all | grep uidd
uidd.service          loaded active running Daemon for generating
uids
steveguo@linux-mukq:~$
```

A screenshot of a terminal window. The user has run the command `sudo service uidd start`. Then, they ran `sudo service --status-all | grep uidd`. The output shows the `uidd` service is loaded, active, and running. It is described as a "Daemon for generating UIDs".

Figure 2.65 Checking uidd

---

### Note

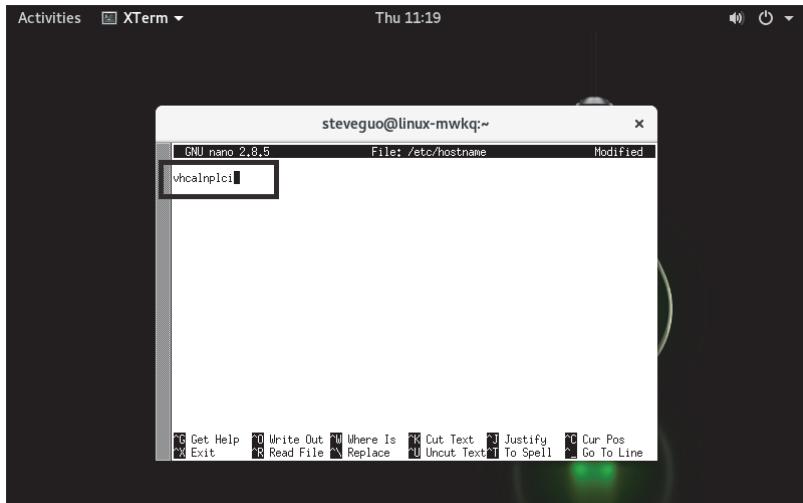
You may need to provide your administrator password.

---

### Change Network Settings

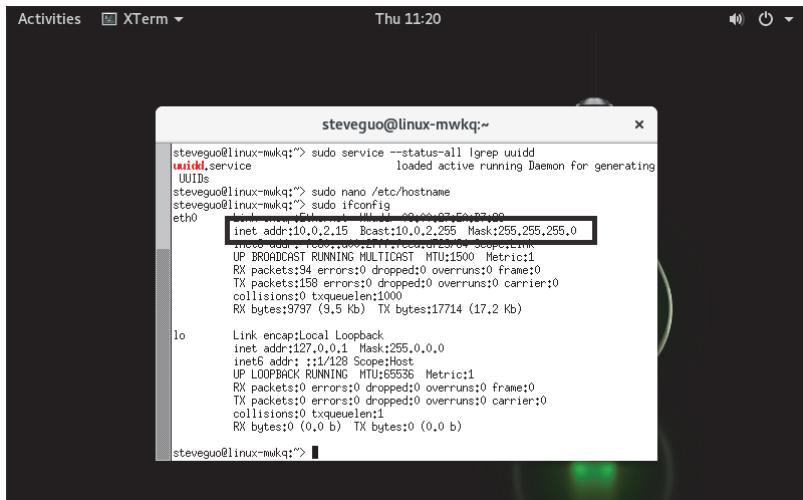
Several network settings need to be changed. The following steps will guide you to do so:

- In the command line, edit your host name using the `sudo nano /etc/hostname` command. Change the content to “`vhcalnplci`”, as shown in [Figure 2.66](#). Quit nano by pressing `[Ctrl]+[X]`, then press `[Y]`, and then press `[Enter]`.



**Figure 2.66** Changing `/etc/hostname`

- Check your current IP address by entering `sudo ifconfig`. Find the IP address in the output, as shown in [Figure 2.67](#).

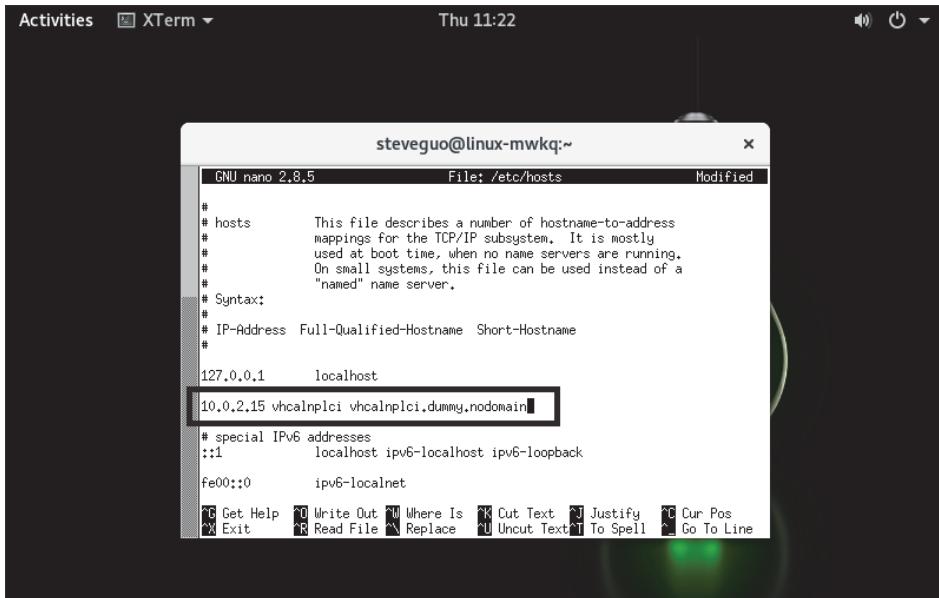


**Figure 2.67** Checking Local IP Address

**Note**

By default, the IP address of your VM should be 10.0.2.15.

3. Create a local DNS record in `/etc/hosts` by entering `sudo nano /etc/hosts` in the command line. Enter `10.0.2.15 vhcalnplci vhcalnplci.dummy.nodomain` in the file, as shown in [Figure 2.68](#). Then quit by pressing `[Ctrl]+[X]`, `[Y]`, and finally `[Enter]`.



**Figure 2.68** Changing `/etc/hosts`

4. Refresh the network service by entering `sudo rcnetwork restart` in the command line.

## 2.2.5 Install SAP NetWeaver AS ABAP Components

Everything's prepared! Now you can start the real installation process by following these instructions:

1. Switch to administration mode by entering the `sudo -i` command.
2. Jump to the shared folder you've just set in the settings by entering the `cd /media/sf_sapinstall` command.
3. Change the access rights for `install.sh` using the `chmod +x install.sh` command.

4. Execute the install script using the `./install.sh` command.

#### Note

Note that the `./install.sh` command starts with a period (.)

5. Hold [Enter] to scroll through and view a very long end user agreement, press [Q] to return to the terminal, and then enter “yes” to agree to it (Figure 2.69). It will take 20–60 minutes to install it.

#### Note

If it informs you that some libraries are missing, enter "yes" and then press [Enter].

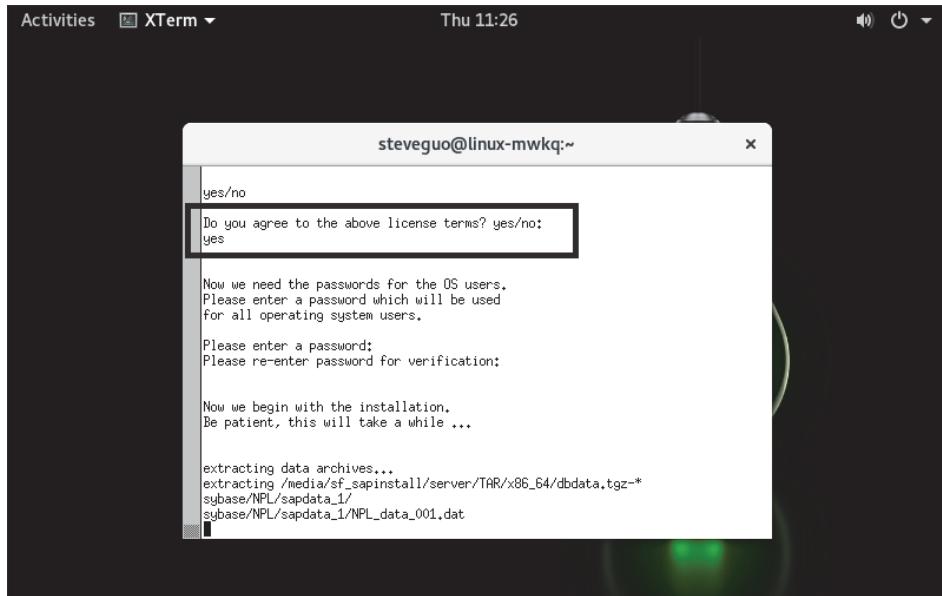


Figure 2.69 Install SAP NetWeaver AS ABAP

6. The console will show an **Installation of NPL Successful** message when it finishes, as shown in [Figure 2.70](#).

#### Tip

Here are some useful commands:

- To stop SAP NetWeaver AS ABAP:
  - su npladm
  - stopsap
- To start SAP NetWeaver AS ABAP:
  - su npladm
  - startsap
- To shut down the virtual machine:
  - sudo shutdown

The screenshot shows a terminal window titled "steveguo@linux-mwkq:~". The window contains the following text output:

```
Checking syb Database
Database is running
-----
Starting Startup Agent sapstartsrv
OK
Instance Service on host vhcalnplci started
-----
starting SAP Instance ASCS01
Startup-Log is written to /home/npladm/startsap_ASCS01.log
-----
/usr/sap/NPL/ASCS01/exe/sapcontrol -prot NI_HTTP -nr 01 -function Start
Instance on host vhcalnplci started
Starting Startup Agent sapstartsrv
OK
Instance Service on host vhcalnplci started
-----
starting SAP Instance D00
Startup-Log is written to /home/npladm/startsap_D00.log
-----
/usr/sap/NPL/D00/exe/sapcontrol -prot NI_HTTP -nr 00 -function Start
Instance on host vhcalnplci started
[Installation of NPL successful]
[redacted]
vhcalnplci:/media/sf_sapinstall #
```

Figure 2.70 Installation of SAP NetWeaver AS ABAP Finished

You also need to install SAP GUI as a client. The install file for SAP GUI is under `\client\SAPGUI4Windows` for your download file. You can install it with just a few clicks.

### 2.2.6 Post-Installation Steps

A series of post-installation steps should be performed to make SAP Fiori launchpad functional and to ensure it can be accessed via your client. You need to install a developer license to use this system legally. We'll walk through these steps in the following sections.

### Change Virtual Machine Network Settings

You need to change the network settings for your virtual machine. These changes enable access from your computer to the virtual machine, as follows:

1. In your virtual machine window, access **Machine • Settings...**, as shown in [Figure 2.71](#).

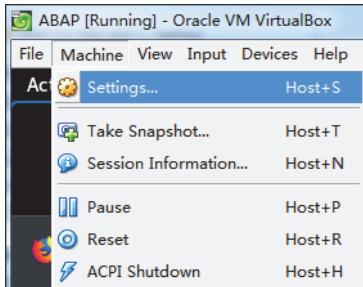


Figure 2.71 Setting Menu for Virtual Machine

2. In the settings window, choose **Network**, then click the triangle next to **Advanced** and click **Port Forwarding**, as shown in [Figure 2.72](#).

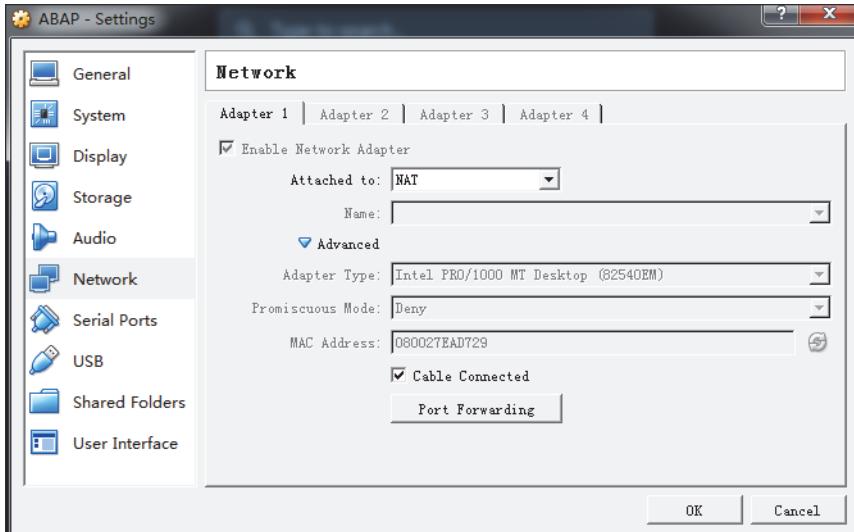
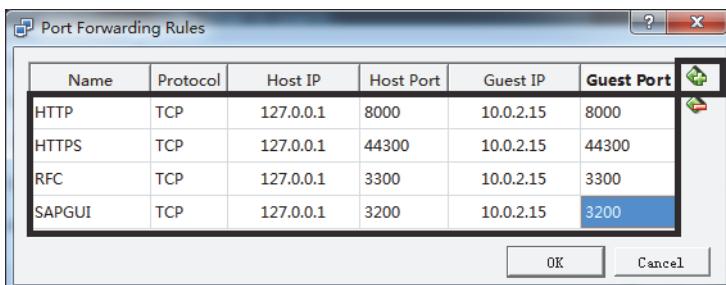


Figure 2.72 Network Settings Panel for Virtual Machine

3. Create four rules by clicking the plus button + on the right four times. Then fill in the content in the table in the **Port Forwarding Rules** window based on [Table 2.1](#). The result should look like [Figure 2.73](#). In general, these rules enable you to access the virtual machine using IP 127.0.0.1 through any client application installed on your operating system.

| Name   | Protocol | Host IP   | Host Port | Guest IP  | Guest Port |
|--------|----------|-----------|-----------|-----------|------------|
| HTTP   | TCP      | 127.0.0.1 | 8000      | 10.0.2.15 | 8000       |
| HTTPS  | TCP      | 127.0.0.1 | 44300     | 10.0.2.15 | 44300      |
| RFC    | TCP      | 127.0.0.1 | 3300      | 10.0.2.15 | 3300       |
| SAPGUI | TCP      | 127.0.0.1 | 3200      | 10.0.2.15 | 3200       |

**Table 2.1** Network Settings for Virtual Machine



**Figure 2.73** Port Fowarding Rules

4. Click **OK** to save. You don't need to restart for your changes to take effect.

### Change Hosts File in the Operating System

Normally, you access the system by a domain name, not an IP address. For this you need to change your hosts file. Add the following record (see [Figure 2.74](#)):

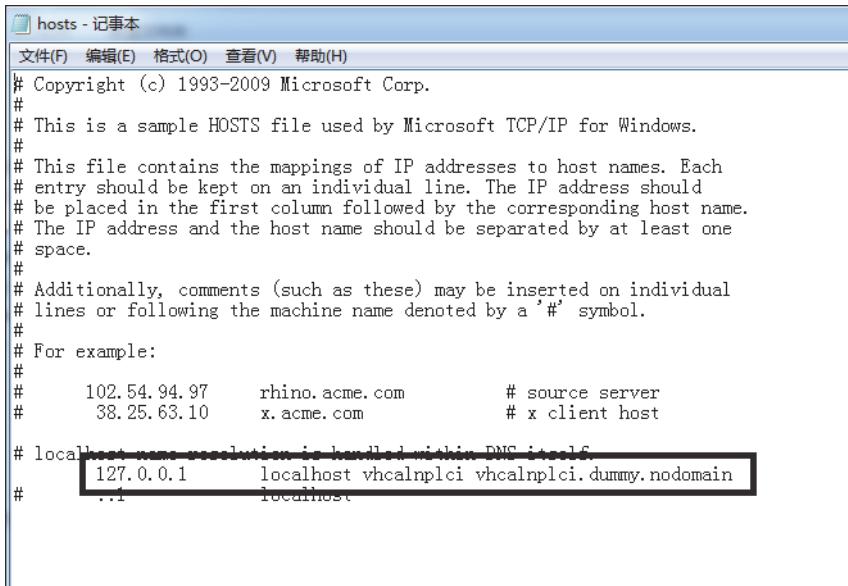
127.0.0.1 vhcalnplci vhcalnplci.dummy.nodomain

---

#### Note

In Windows, this file is located at <windowsdir>\systems32\etc.

In OS X, the file is located at /etc/hosts.



```

hosts - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97      rhino.acme.com          # source server
#      38.25.63.10      x.acme.com               # x client host
#
# localhost resolution is handled within DMC itself
#      127.0.0.1      localhost vhcalsplci vhcalsplci.dummy.nodomain
#      ...1      localhost

```

Figure 2.74 Host File in Windows System

### Log On to the System and Apply for a Developer License

You can obtain a free developer license from SAP. It's valid for 90 days, and you can extend the validity period when it expires. To begin, follow these steps:

1. Open SAP Logon, choose **Connections**, and click **New** to create a new connection. It should look like [Figure 2.75](#).

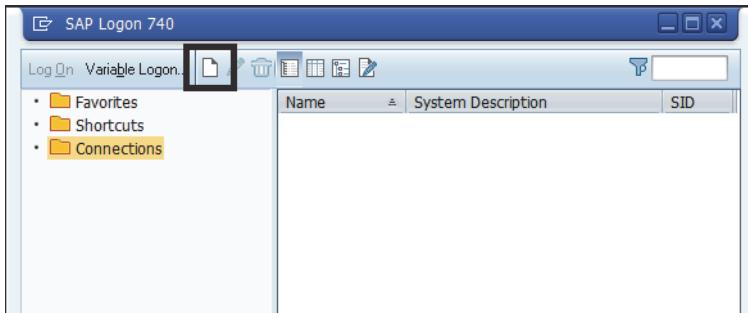


Figure 2.75 SAP Logon

2. Click **Next** on the first screen shown in [Figure 2.76](#).

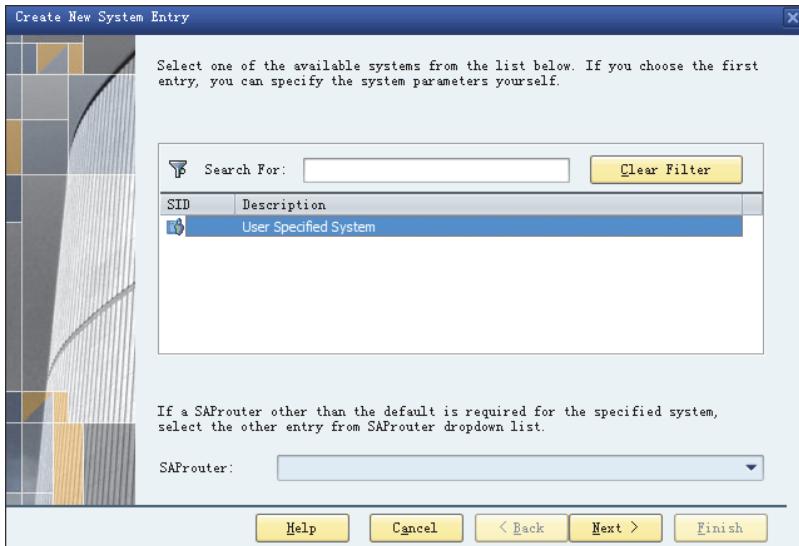


Figure 2.76 Create New System Entry

3. Enter a **Description** (e.g., “NPL”). For **Application Server**, use “vhcalnplci”; for **Instance Number**, use “OO”; and for **System ID**, use “NPL”. Then click **Finish**, as shown in [Figure 2.77](#).

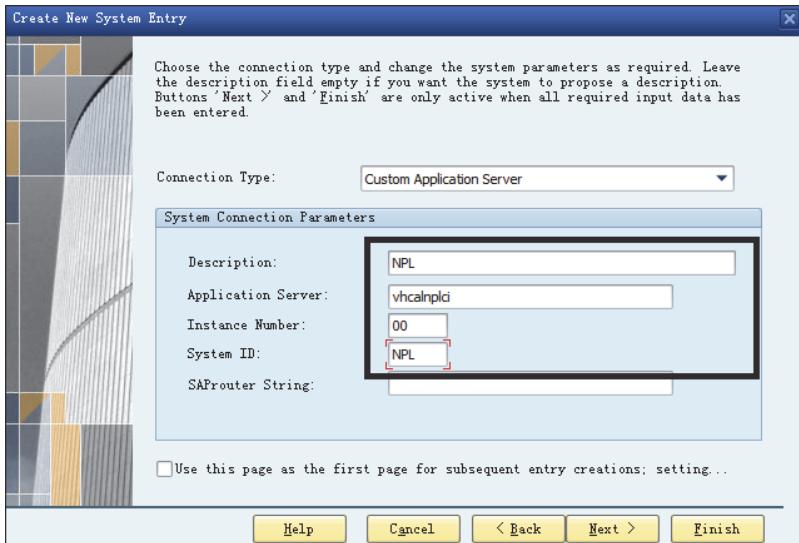
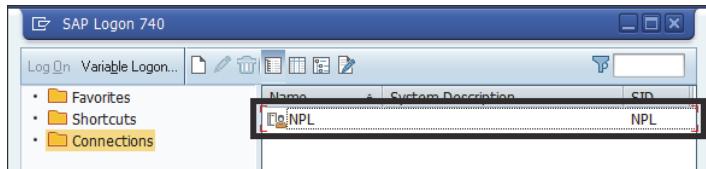


Figure 2.77 Connection Settings

4. The result of the SAP GUI logon should look like [Figure 2.78](#). Now double-click **NPL** to logon to the system.



**Figure 2.78** SAP Logon after Connection Has Been Added

5. On the next screen, change **Client** to “000”. For **User**, use “SAP\*”; and for **Password**, use “Download”, as shown in [Figure 2.79](#). Press **Enter** to log on.

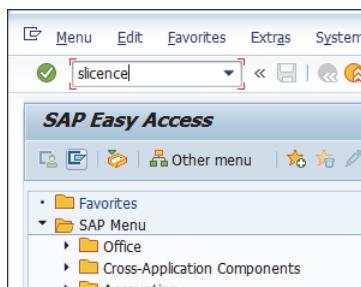
#### Note

The fifth character of the password is the number 1 (one), not the letter I (L).

The screenshot shows the SAP logon dialog box. It has fields for Client (000), User (sap\*), and Password (\*\*\*\*\*). Other fields include New password, Logon Language (EN), and a note about the password character.

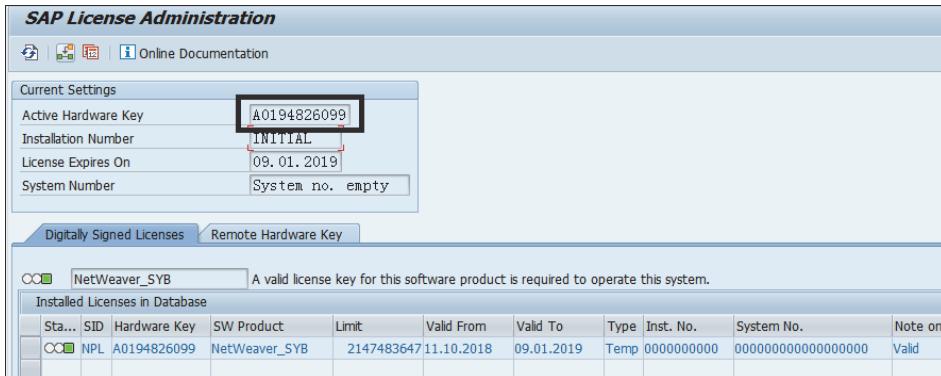
**Figure 2.79** Logon UI for System NPL

6. Enter “license” in the input field and press **Enter**, as shown in [Figure 2.80](#).



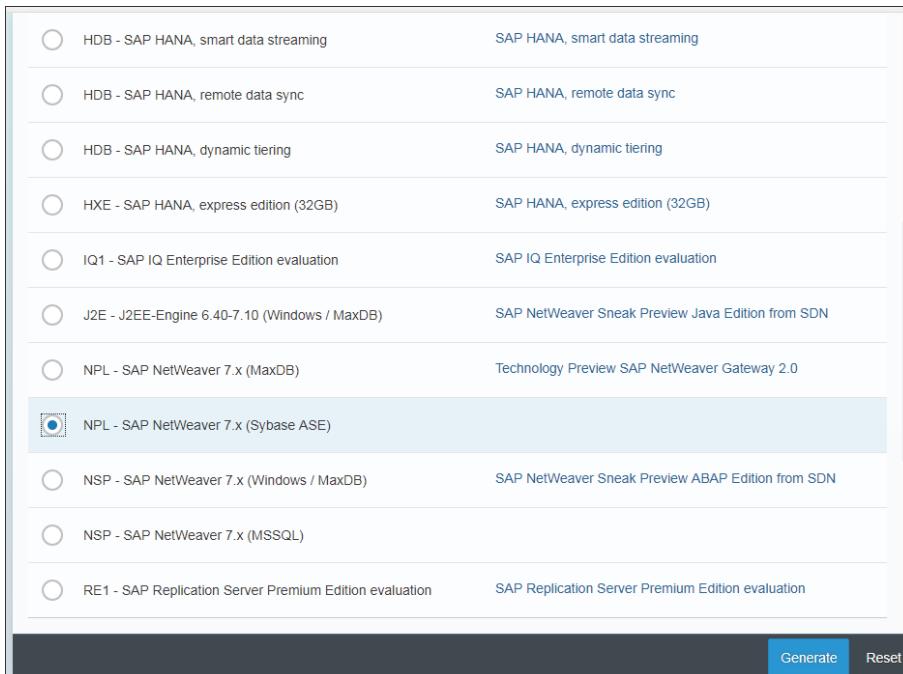
**Figure 2.80** SAP Easy Access Menu

7. The **SAP License Administration** screen will appear. Copy the content of the **Active Hardware Key** field, as shown in [Figure 2.81](#).



**Figure 2.81** SAP License Administration

8. Go to <https://go.support.sap.com/minisap/#/minisap>. Scroll down and choose **NPL—SAP NetWeaver 7.x (Sybase ASE)**, as shown in [Figure 2.82](#).



**Figure 2.82** License Application Website

9. Continue to scroll down, fill in the form with your personal information, and paste the hardware key you copied previously into the **Hardware Key** field. Then click **Generate**, as shown in Figure 2.83.

The screenshot shows a SAP application window titled "Personal Data & System Info". A large black rectangular box highlights the data entry section. Inside this box, the following fields are visible:

- Salutation\*:  Mr  Ms
- First Name\*: Steve
- Last Name\*: Guo
- Email Address\*: steve.guo@outlook.com
- Hardware Key\*: A0194826099

Below this section is a "License Agreement" panel with a checkbox labeled "I Agree\*" and a link "Click here to View the License Agreement".

\* Marked fields are required

At the bottom right are "Generate" and "Reset" buttons.

Figure 2.83 License Application Form

10. A file called *NPL.txt* will be downloaded, as shown in Figure 2.84.

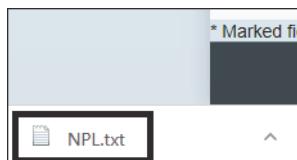


Figure 2.84 Downloaded License Key File

11. Back in SAP GUI, click the the **Install** button shown in Figure 2.85. Choose **NPL.txt** in the file browser.
12. A pop-up like that in Figure 2.86 will inform you that the license has been installed successfully.
13. Figure 2.87 shows the status after you've installed the license. Note that the **Install Number** has changed to **DEMO SYSTEM**, and **License Expires On** has extended to 90 days after the current date.



Figure 2.85 Install License Button

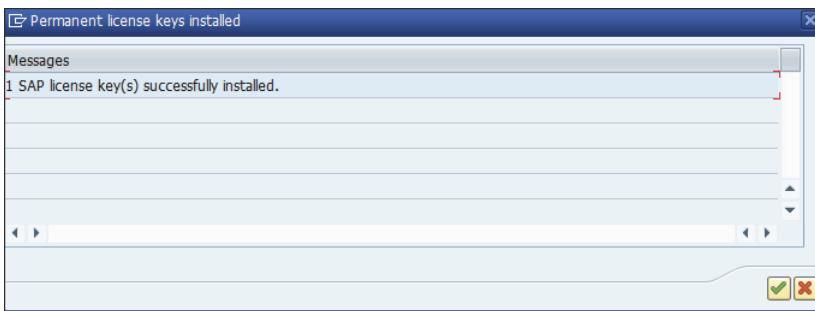


Figure 2.86 Notification for License Installation

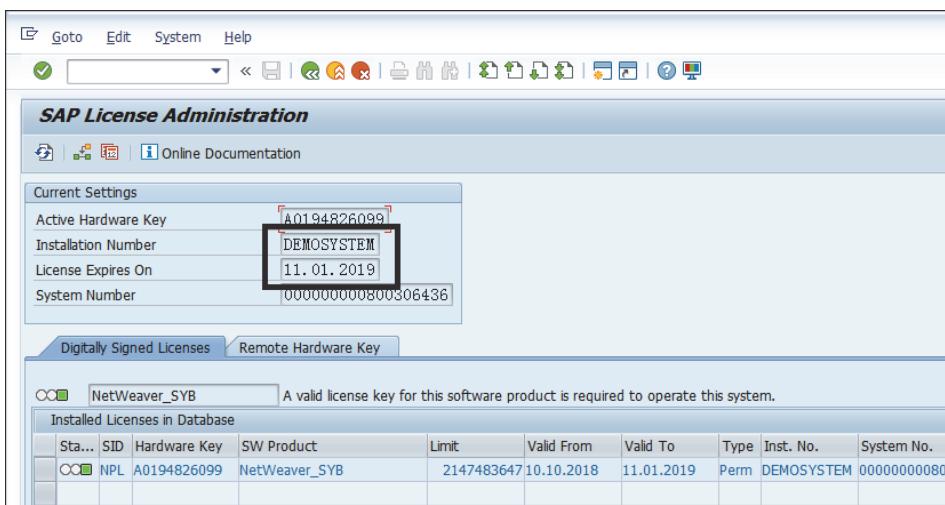
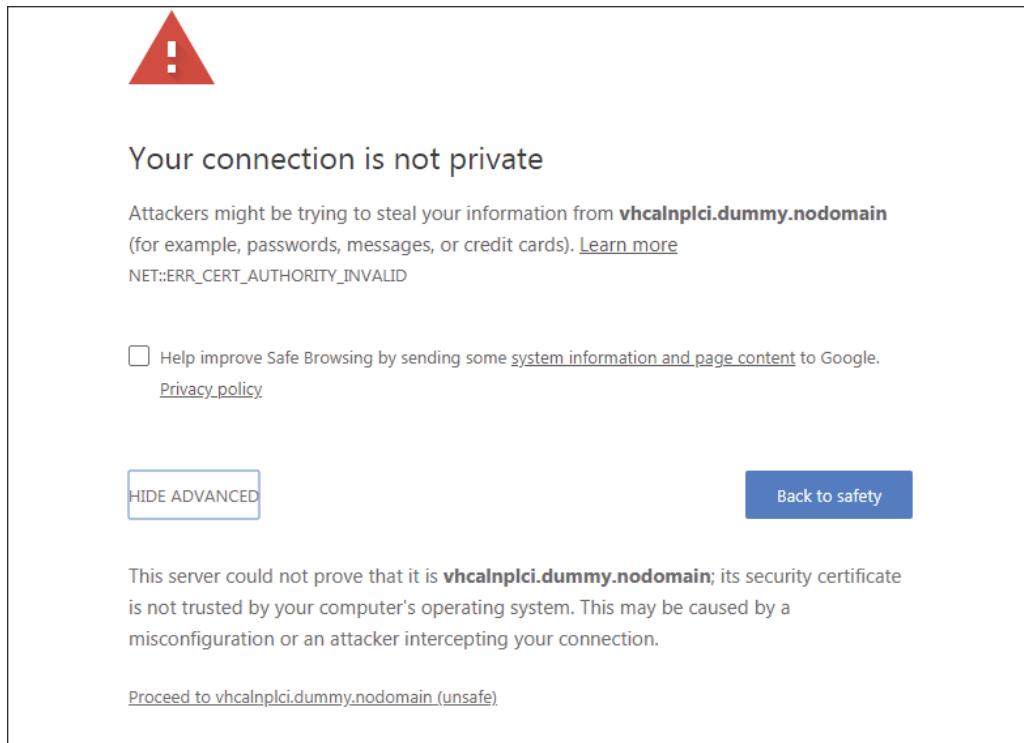


Figure 2.87 SAP License Administration after License Has Been Installed

### Test SAP Fiori Launchpad Sites

You're almost done! Now let's test of SAP Fiori launchpad-associated sites and add them to your bookmarks, as follows:

1. Open your browser and enter `https://vhcalnplci.dummy.nodomain:44300/sap/bc/ui2/flp`. You'll see a warning page like the one shown in [Figure 2.88](#) because your HTTPS certificate isn't certified. You can ignore this warning by clicking **Advanced** and then **Proceed to vhcalnplci.dummy.nodomain (unsafe)**.



**Figure 2.88** Warning Page when Accessing NPL Server

2. The logon page for SAP Fiori launchpad will appear. Enter your predefined **User**, "developer", and **Password**, "Download". Then click **Log On**, as shown in [Figure 2.89](#).

### Note

Remember, the fifth character of the password is the number 1 (one).

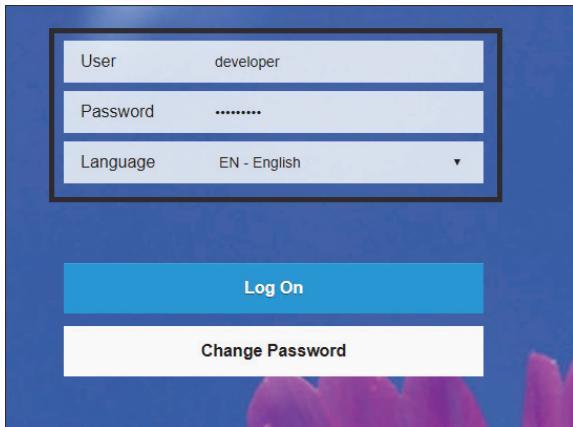


Figure 2.89 SAP Fiori Launchpad Logon Page

3. An SAP Fiori launchpad with lots of predefined tiles will appear, as shown in [Figure 2.90](#). In some versions, there are no tiles to display. This is fine for the purpose of the demos in this book.

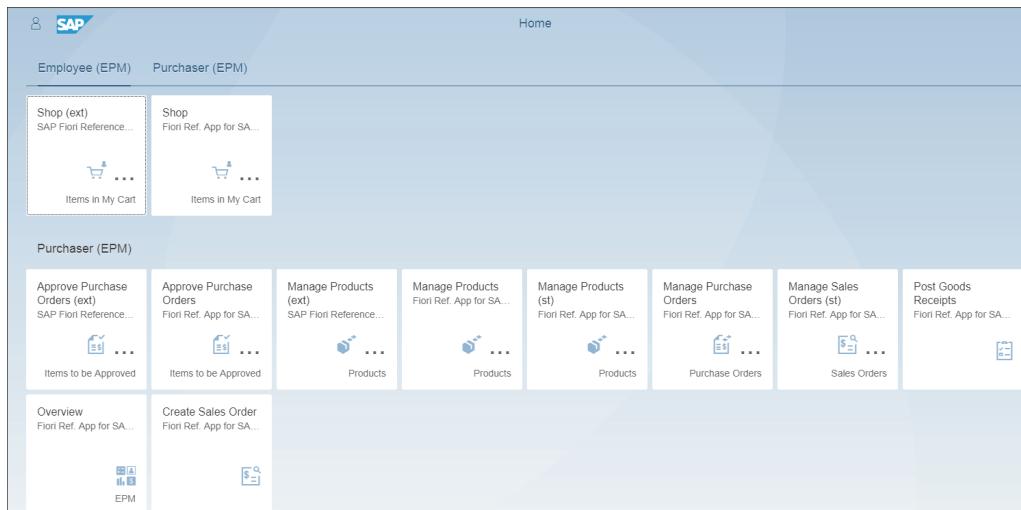


Figure 2.90 SAP Fiori Launchpad On-Premise

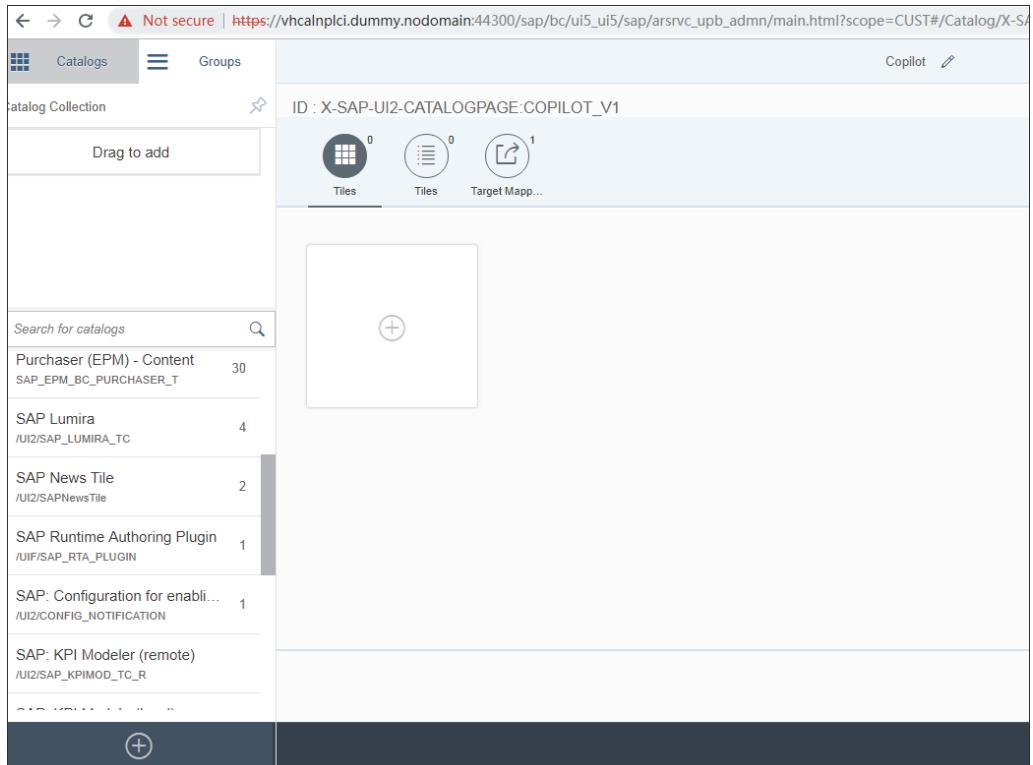
**Note**

Bookmark this page as *SAP Fiori launchpad local*. You'll use it in later chapters.

4. Change the URL to [https://vhcalnplci.dummy.nodomain:44300/sap/bc/ui5\\_ui5/sap/arsvc\\_upb\\_admn/main.html?scope=CUST](https://vhcalnplci.dummy.nodomain:44300/sap/bc/ui5_ui5/sap/arsvc_upb_admn/main.html?scope=CUST). You'll navigate to SAP Fiori launchpad designer, which is for configuring catalogs and tiles, as shown in [Figure 2.91](#).

**Note**

Bookmark this page as *SAP Fiori launchpad designer local*. You'll use this page in later chapters.

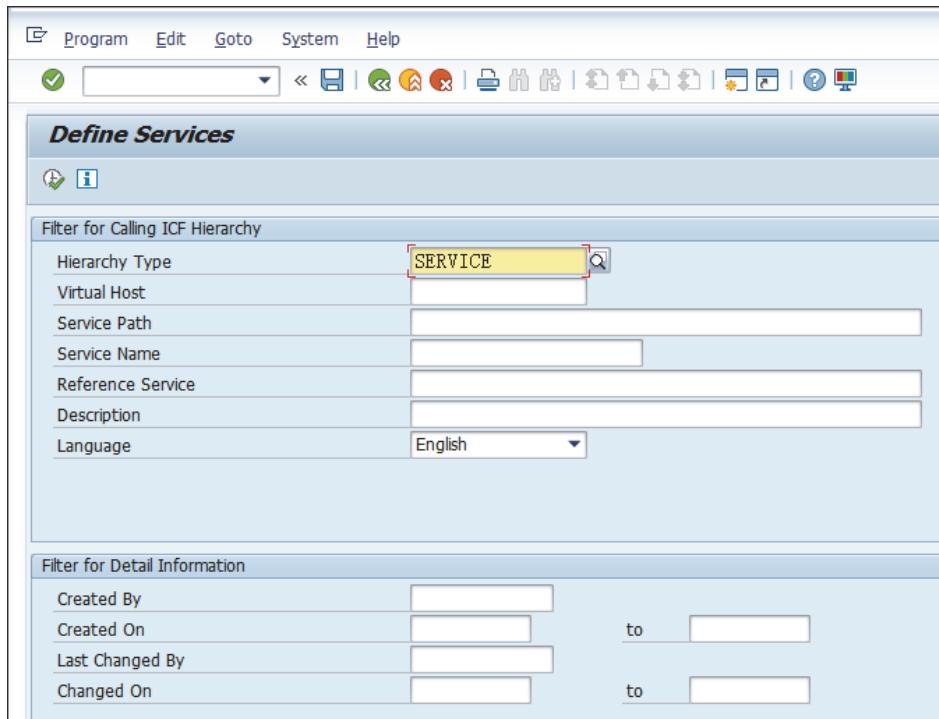


**Figure 2.91** SAP Fiori Launchpad Designer On-Premise

### Activate ADT Service

ABAP Development Tools (ADT) is a service in SAP NetWeaver AS ABAP to support HTTP access for development objects. It will be used to import and deploy SAPUI5 apps to SAP NetWeaver AS ABAP. By default, the service is disabled. You need to enable it by following these steps:

1. Return to SAP GUI and enter Transaction SICF to enter the HTTP Service Management application. Click **Execute** (see [Figure 2.92](#)).



**Figure 2.92** UI for Transaction SICF

2. Expand **sap • bc • adt** to find that the **adt** node is grayed out. Right-click it and choose **Activate Service** in the context menu, as shown in [Figure 2.93](#). If you cannot find **Activate Service** in the context menu, that means it is already activated, and you can proceed directly to [Section 2.3](#).
3. In the pop-up window, choose **Yes**, as shown in [Figure 2.94](#).

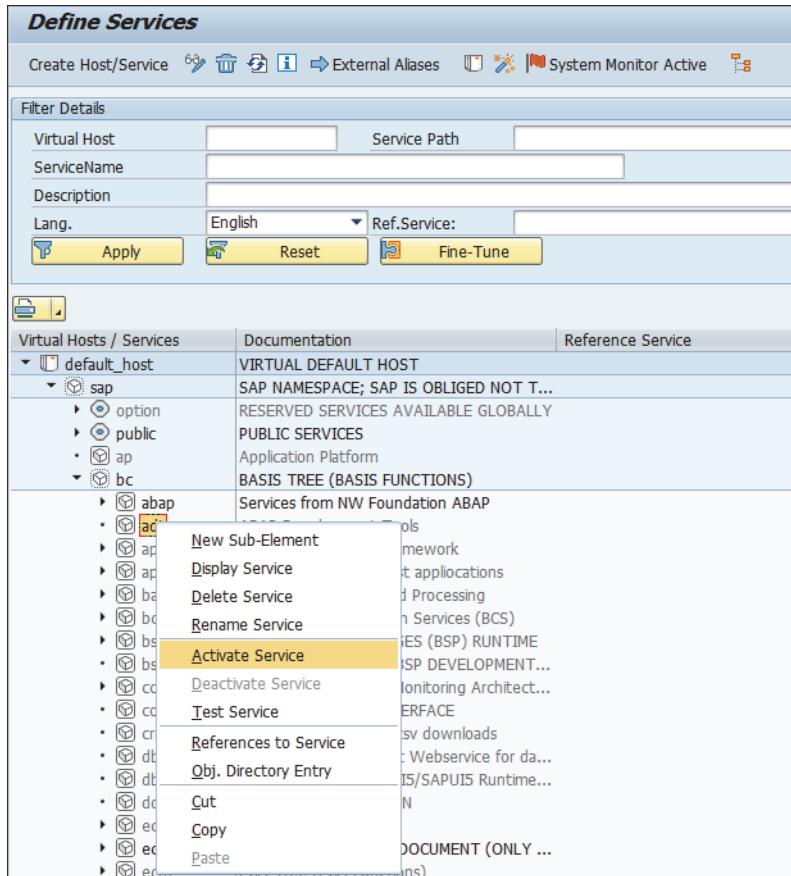


Figure 2.93 Activate adt Service

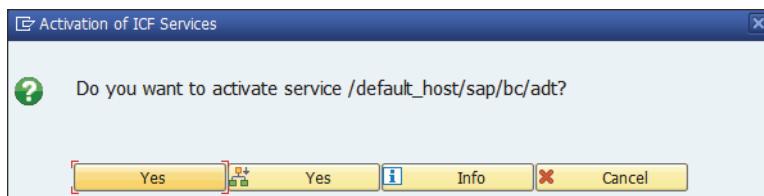


Figure 2.94 Confirm Dialog to Activate adt Service

Now you've finished all the settings in SAP NetWeaver AS ABAP. You can close SAP GUI, but don't close the virtual machine.

## 2.3 Connect a Cloud Environment to an On-Premise Environment

Because development is done in the cloud, you need to set up a secure connection between the cloud-based SAP Web IDE and your on-premise SAP NetWeaver AS ABAP system. You need SAP Cloud Connector to build a secure bridge between those two environments. Then connectivity service settings in SAP Cloud Platform need to be made to connect to your backend system. We'll perform all the necessary activities for this connection in the following sections.

### 2.3.1 Install SAP Cloud Connector

Before installing SAP Cloud Connector, you need to have a Java Platform, Standard Edition Development Kit (JDK) at version 7 or 8. You can also use the SAP JVM downloaded from the same place with SAP Cloud Connector.

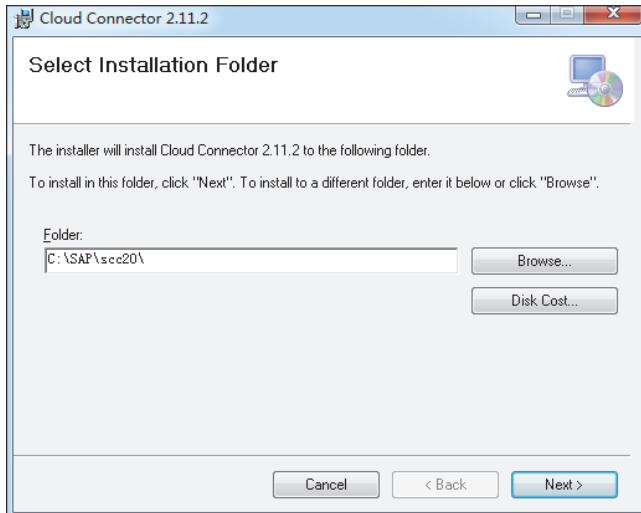
After you've installed JVM, you can start installing SAP Cloud Connector with the following steps:

1. On the welcome screen, click **Next**, as shown in [Figure 2.95](#).



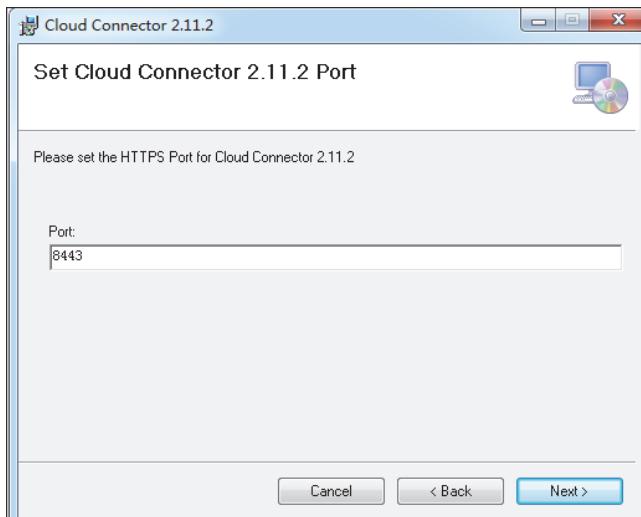
**Figure 2.95** Installing SAP Cloud Connector: Welcome Page

2. On the **Select Installation Folder** screen, choose a place to install SAP Cloud Connector and click **Next**, as shown in [Figure 2.96](#).



**Figure 2.96** Installing SAP Cloud Connector: Choose Install Path

3. Set the port number; the default number is 8443. Click **Next**, as shown in [Figure 2.97](#).

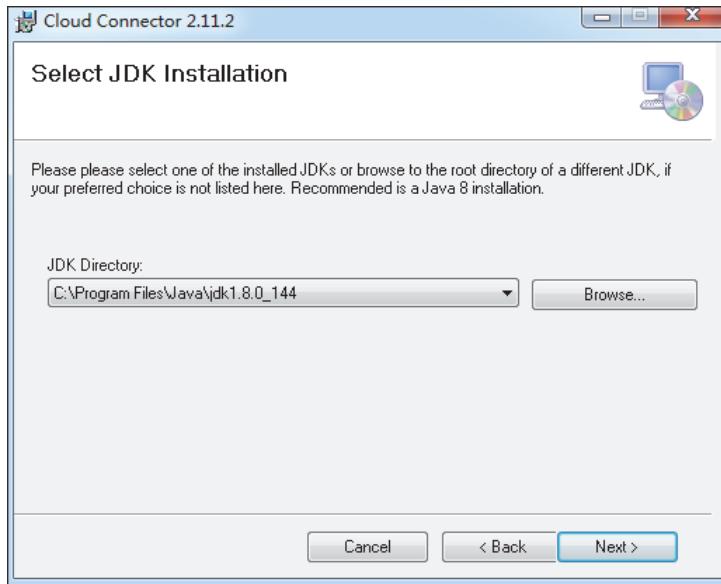


**Figure 2.97** Installing SAP Cloud Connector: Choose Port

**Note**

Most startup failure cases are caused by the port being in use by other applications—and 8443 is a really common port, often used by other software. It's better to change it to a rarer one, such as 8888, to avoid startup failures.

4. Choose your **JDK Directory** from the available list of JDKs. In some cases, you may need to browse to the JDK path manually. Then click **Next**, as shown in [Figure 2.98](#).



**Figure 2.98** Installing SAP Cloud Connector: Choose JDK

5. Check **Start Cloud Connector after Finishing the Setup** and click **Next**, as shown in [Figure 2.99](#).
6. Click **Next** to confirm the installation, as shown in [Figure 2.100](#).

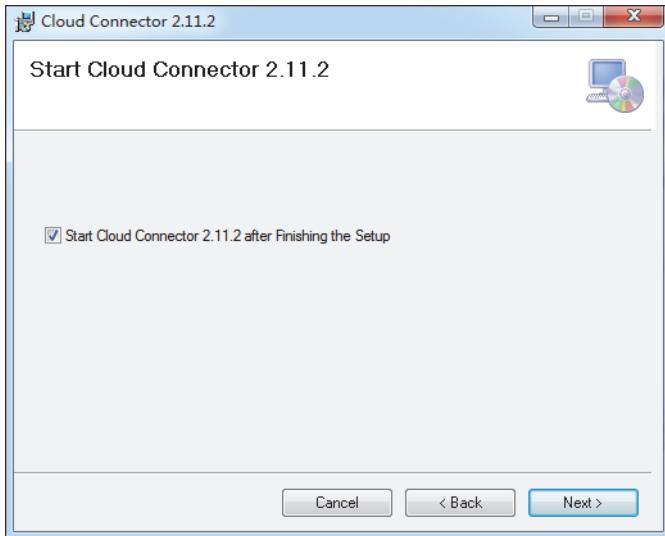


Figure 2.99 Installing SAP Cloud Connector: Start SAP Cloud Connector

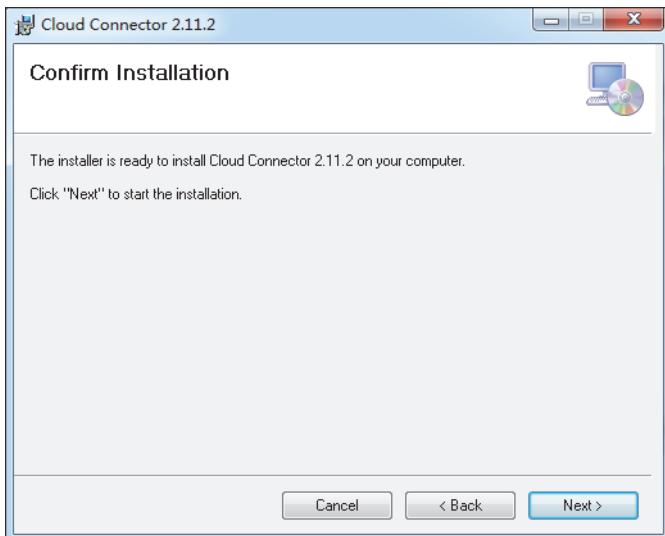
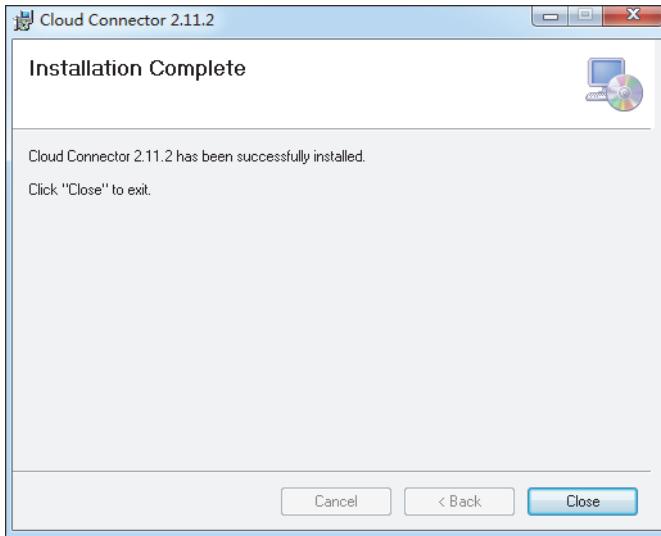


Figure 2.100 Installing SAP Cloud Connector: Confirm

7. Click **Close** to finalize the installation, as shown in [Figure 2.101](#).



**Figure 2.101** Installing SAP Cloud Connector: Installation Complete

8. Two shortcuts will be placed on your desktop, one for starting and one for stopping SAP Cloud Connector, as shown in [Figure 2.102](#).



**Figure 2.102** Shortcuts for SAP Cloud Connector

### 2.3.2 Set Up SAP Cloud Connector

SAP Cloud Connector is used to setup a secure connection between your computer and SAP Cloud Platform. In this section, you'll initialize SAP Cloud Connector, connect it to SAP Cloud Platform, and finally enable the access to your local SAP NetWeaver AS ABAP system from the cloud, as follows:

1. Access <https://localhost:8443> or use the port you've defined. Use the predefined **User Name** "Administrator" and **Password** "manage" and then click **Logon**, as shown in [Figure 2.103](#).



Figure 2.103 SAP Cloud Connector: Logon Page

2. First you need to change the default password to a new one. You may also need to set a proxy to connect to the internet, as shown in [Figure 2.104](#).

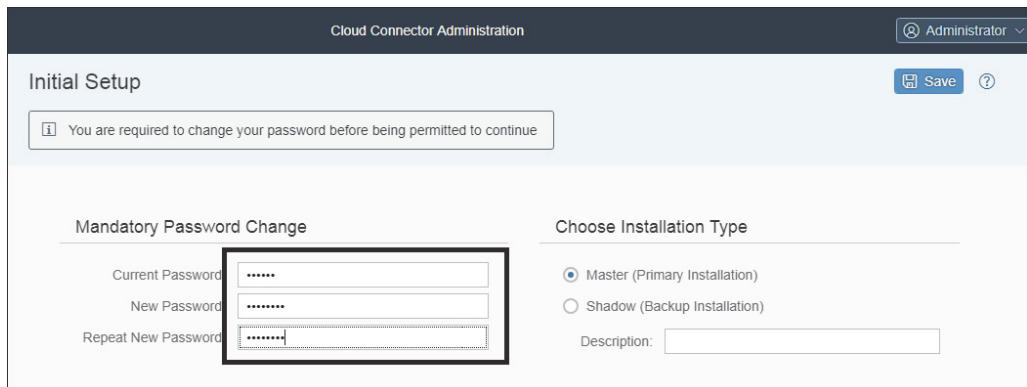


Figure 2.104 SAP Cloud Connector: Initial Setup

3. Now you'll connect SAP Cloud Connector to your SAP Cloud Platform trial account. Here you need to select **Europe (Rot)-Trial** for **Region**. Fill in your account name, which looks like "p<xxxxx>trial", in the **Subaccount** field. Also enter your **Account User** (*pxxxxxx*) and **Password**. Then click **Save** to save this configuration, as shown in [Figure 2.105](#).

## 2 Development Environment Setup

Define Subaccount

Cloud Connector is not configured and remains inoperative unless you provide the following settings

Save ?

First Subaccount

\*Region: Europe (Rot) - Trial

\*Subaccount: p2000524802trial

Display Name: Steve

\*Subaccount User: p2000524802

\*Password:  .....

Location ID: Enter location ID to overwrite default

Description:

HTTPS Proxy

Host:

Port:

User:

Password:

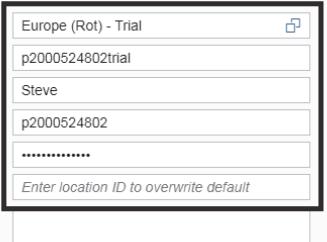


Figure 2.105 SAP Cloud Connector: Define Subaccount

4. Figure 2.106 shows the status when you've connected to SAP Cloud Platform. You'll see a green success message.

Subaccount: Steve

Steve

Disconnect Subaccount Certificate

Connected since Oct 12, 2018 1:05:53 AM — no active resources available (check Cloud To On-Premise/Access Control)

Connector State

Region: Europe (Rot) - Trial Subaccount: p2000524802trial

Region Host: hanatrial.ondemand.com Subaccount User: p2000524802

HTTPS Proxy: ◇ Location ID:

System Certificate: ◇ Description:

Disaster Recovery Subaccount

Feature not available for trial subaccounts



Figure 2.106 SAP Cloud Connector: Account Connected

5. Click **Cloud To On-Premise** in the left menu, then click the **Add** button on the right side, as shown in [Figure 2.107](#).

The screenshot shows the SAP Cloud Connector interface. On the left, there's a sidebar with a user profile (Steve) and several menu items: Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, Cloud To On-Premise (which is highlighted in blue), On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main content area is titled "Cloud To On-Premise" and contains tabs for ACCESS CONTROL, COOKIE DOMAINS, APPLICATIONS, and PRINCIPAL PROPAGATION. Below these tabs is a section titled "Mapping Virtual To Internal System" with columns for Status, Virtual Host, Internal Host, Check Result, Protocol, and Back-end Type. A message "No data" is displayed. In the top right corner of the main content area, there are "Add" and "Actions" buttons. The "Add" button is highlighted with a red box.

[Figure 2.107](#) SAP Cloud Connector: Cloud to On-Premise Settings

6. Choose **SAP Gateway** as the **Backend Type**, then click **Next**, as shown in [Figure 2.108](#).

The screenshot shows a modal dialog titled "Add System Mapping". Inside, there's a text input field with a placeholder "Select back-end type of on-premise system" and a dropdown menu labeled "Back-end Type:" containing the option "SAP Gateway". At the bottom of the dialog are buttons for "Previous", "Next", and "Cancel".

[Figure 2.108](#) Add System Mapping: Choose Type

7. Choose **HTTPS** as the **Protocol**, then click **Next**, as shown in [Figure 2.109](#).

## Add System Mapping

**Figure 2.109** Add System Mapping: Choose Protocol

8. For **Internal Host**, enter “vhcalnplci.dummy.nodomain”, and enter “44300” for **Internal Port**. Then click **Next**, as shown in Figure 2.110.

## Add System Mapping

i Enter internal (on-premise) host and port

\*Internal Host:

\*Internal Port:

[Previous](#) [Next](#) [Cancel](#)

**Figure 2.110** Add System Mapping: Internal Host and Internal Port

9. Change **Virtual Host** to “npl”, then click **Next**, as shown in Figure 2.111.

**Note**

The purpose of the virtual host is to hide your real address in the cloud. The virtual host and virtual port are only things you can see in the cloud.

Add System Mapping

Optional change virtual names (used on cloud-side)

\*Virtual Host: npl

\*Virtual Port: 44300

Previous   Next   Cancel

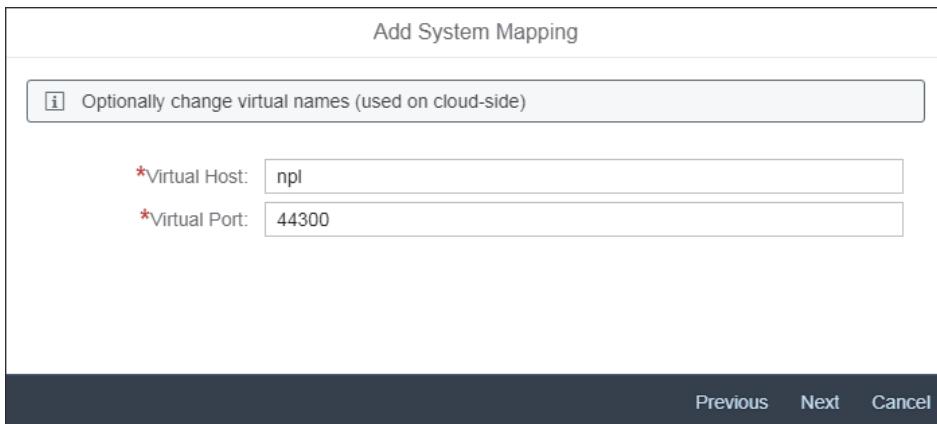


Figure 2.111 Add System Mapping: Virtual Host and Virtual Port

10. Select **None** for **Principal Type**, then click **Next**, as shown in Figure 2.112.

Add System Mapping

Select principal type

Principal Type: None

Previous   Next   Cancel

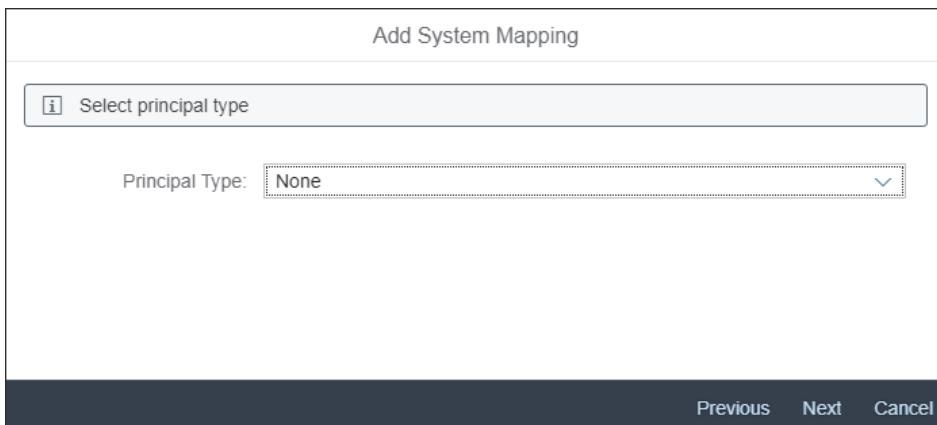
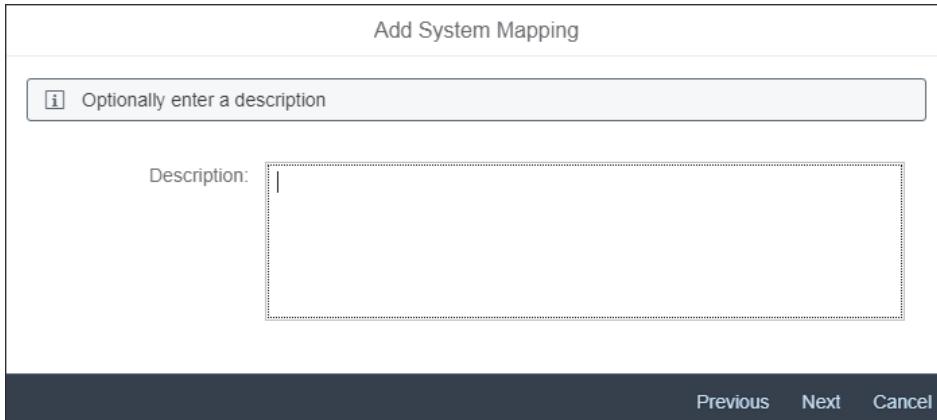


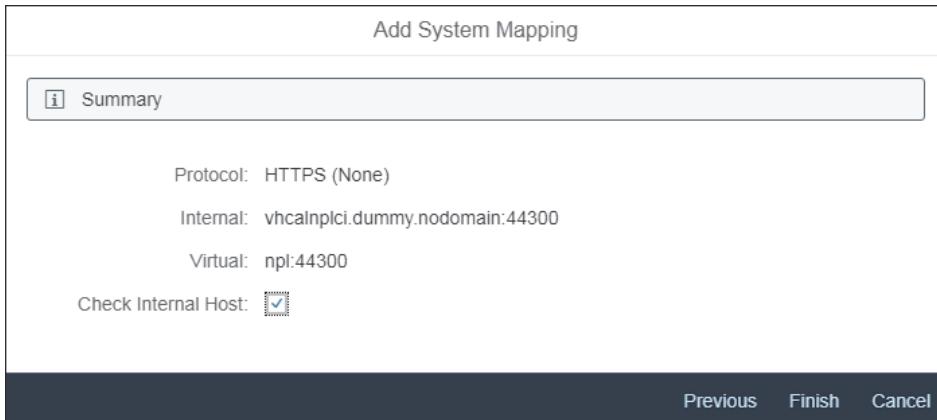
Figure 2.112 Add System Mapping: Principal Type

11. Optionally, add a **Description**. Click **Next**, as shown in [Figure 2.113](#).



**Figure 2.113** Add System Mapping: Description

12. Check the checkbox for **Check Internal Host**, then click **Finish**, as shown in [Figure 2.114](#). This option will test if your SAP NetWeaver AS ABAP service is accessible from your computer.



**Figure 2.114** Add System Mapping: Finish

13. You should now see a row in the first table **Reachable** listed in green under the **Check Result** column. Click the **Plus** button in the lower table, as shown in [Figure 2.115](#).

The screenshot shows the SAP Cloud Connector interface for 'Cloud To On-Premise'. At the top, there are tabs for ACCESS CONTROL, COOKIE DOMAINS, APPLICATIONS, and PRINCIPAL PROPAGATION. Below these, a table titled 'Mapping Virtual To Internal System' lists a single entry: 'Status' (npl:44300), 'Virtual Host' (vhcalnplci.dummy.nodomain:44300), 'Internal Host' (vhcalnplci.dummy.nodomain:44300), 'Check Result' (Reachable), 'Protocol' (HTTPS), and 'Back-end Type' (SAP Gateway). A blue plus sign icon is located in the 'Actions' column for this row. Below this table is another section titled 'Resources Accessible On npl:44300' with a table showing 'No data'. A blue plus sign icon is also present here.

**Figure 2.115** SAP Cloud Connector: Cloud to On-Premise with One Mapping

14. In the **Add Resource** pop-up, enter “/sap” for **URL Path**, then select the **Path and all-sub-paths** radio button for **Access Policy**. Then click **Save**, as shown in [Figure 2.116](#).

The screenshot shows the 'Add Resource' dialog box. It has fields for 'URL Path' (containing '/sap'), 'Enabled' (checkbox checked), 'Access Policy' (radio button selected for 'Path and all sub-paths'), and a 'Description' text area. At the bottom are 'Save' and 'Cancel' buttons.

**Figure 2.116** Add HTTP Resource for System Mapping

15. [Figure 2.117](#) shows the final status of SAP Cloud Connector.

The screenshot shows the SAP Cloud Connector interface under the 'Cloud To On-Premise' section. It displays two main tables:

- Mapping Virtual To Internal System:** A table with columns: Status, Virtual Host, Internal Host, Check Result, Protocol, Back-end Type, and Actions. One row is shown: npl:44300, vhcainipci.dummy.nodomain:44300, Reachable, HTTPS, SAP Gateway.
- Resources Accessible On npl:44300:** A table with columns: Enabled, Status, URL Path, Access Policy, and Actions. One row is shown: checked, /sap, Path and all sub-paths.

Figure 2.117 SAP Cloud Connector, Final Status

### 2.3.3 Create a Destination in SAP Cloud Platform

Now that you've installed SAP Cloud Connector, you need to add a destination in your SAP Cloud Platform account, as follows:

1. Open SAP Cloud Platform Cockpit for your neo trial, which you bookmarked previously.
2. Click **Cloud Connectors** under **Connectivity**. You'll see the status of SAP Cloud Connector. You can also see the virtual Host **npl:44300** under **Exposed Backend Systems**, as shown in [Figure 2.118](#).

The screenshot shows the SAP Cloud Platform Cockpit interface. The left sidebar navigation includes:

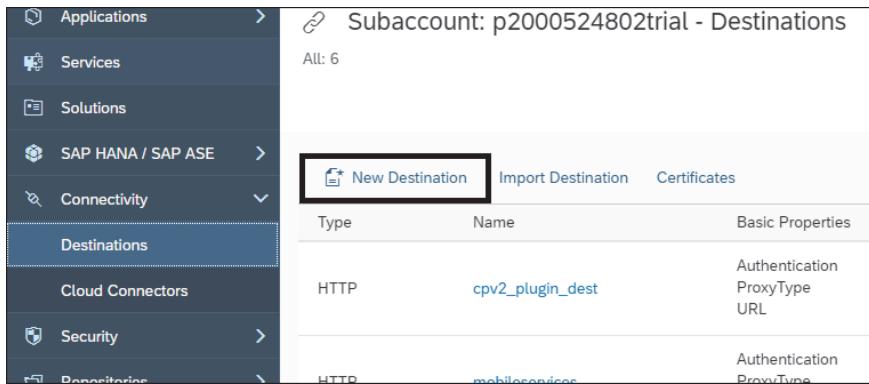
- Overview
- Applications
- Services
- Solutions
- SAP HANA / SAP ASE
- Connectivity
- Destinations
- Cloud Connectors** (highlighted)
- Security
- Repositories

The main content area shows:

- Master Instance:** Connector ID: 6D5ACE30CD7711E881BAC3F9C0A83801, Connected since: 11.10.2018 17:05:53, \*Initiated by: p2000524802, Version: 2.11.2, Java Version: 1.8.0\_144 (Oracle Corporation), High Availability: Inactive.
- Exposed Back-End Systems:** A table with columns: Host, Protocol, Back-End Type, and Resources. One row is shown: npl:44300, HTTP, SAP Gateway, Available.

Figure 2.118 Checking Connection to SAP Cloud Connector in SAP Cloud Platform Cockpit

3. Click **Destinations** under **Connectivity**. Then click **New Destination** to create a new destination, as shown in Figure 2.119.



**Figure 2.119** Manage Destinations

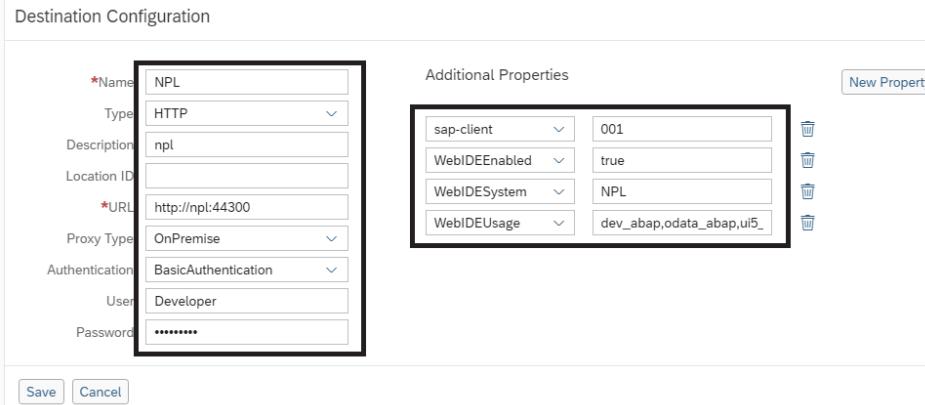
4. Fill in the form according to Table 2.2. Note that for the **sap-client**, **WebIDEEEnabled**, **WebIDESystem**, and **WebIDEUsage** fields, you need to add these fields through the **New Property** button and choose a parameter name from the dropdown list. The result should look like Figure 2.120.

| Field          | Value                                     |
|----------------|---|
| Name           | NPL                                       |
| Type           | HTTP                                      |
| Description    | NPL                                       |
| URL            | http://npl:44300                          |
| Proxy Type     | OnPremise                                 |
| Authentication | Basic                                     |
| User           | Developer                                 |
| Password       | <Your password for SAP NetWeaver AS ABAP> |
| sap-client     | 001                                       |
| WebIDEEEnabled | true                                      |

**Table 2.2** Information for Destination NPL

| Field        | Value                                |
|--------------|--------------------------------------|
| WebIDESystem | NPL                                  |
| WebIDEUsage  | dev_abap,odata_abap,ui5_execute_abap |

Table 2.2 Information for Destination NPL (Cont.)



The screenshot shows the SAP Fiori-style Destination Configuration form. On the left, there is a main configuration area with the following fields:

- \*Name: NPL
- Type: HTTP
- Description: npl
- Location ID:
- \*URL: http://npl:44300
- Proxy Type: OnPremise
- Authentication: BasicAuthentication
- User: Developer
- Password: \*\*\*\*\*

On the right, there is an "Additional Properties" section containing the following entries:

| Property      | Value                                |
|---------------|--------------------------------------|
| sap-client    | 001                                  |
| WebIDEEnabled | true                                 |
| WebIDESystem  | NPL                                  |
| WebIDEUsage   | dev_abap,odata_abap,ui5_execute_abap |

At the bottom left are "Save" and "Cancel" buttons.

Figure 2.120 Destination Configuration Form

5. After filling in the form, click **Save**, then click **Check Connection**. A pop-up like the one in Figure 2.121 will appear.

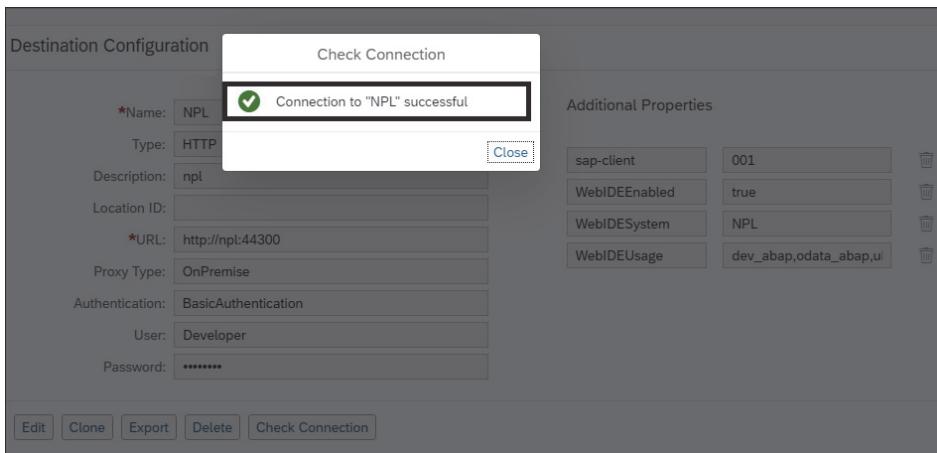


Figure 2.121 Checking Results of Destination Configuration

## 2.4 Summary

In this chapter, you prepared all the environments needed for this book. For cloud development, you need a free SAP Cloud Platform account and need to activate the SAP Web IDE full-stack version and SAP Cloud Platform Portal. For the on-premise environment, you still need SAP Web IDE full-stack for development, and you also need to install a bunch of components, including a virtual machine, an operating system, SAP NetWeaver AS ABAP, and SAP Cloud Connector.

All demos and examples will work for both environments, but in some cases the deployment processes and code are slightly different. For simplicity, we will always write and test code in the cloud environment, and we'll callout any differences in the on-premise environment.

In the next chapter, you'll gain a deep understanding of how SAPUI5 applications are embedded in SAP Fiori launchpad and how to communicate among SAPUI5 applications.



# Chapter 3

## SAPUI5 Applications in SAP Fiori Launchpad

*How did SAPUI5 end up embedded in SAP Fiori launchpad? What are the benefits of the SAP Fiori launchpad architecture? How do SAPUI5 applications in SAP Fiori launchpad communicate with each other? This chapter will explain everything you need to take care of when your SAPUI5 application is running in SAP Fiori launchpad.*

You already have some experience creating SAPUI5 applications and running them standalone or in SAP Fiori launchpad, but you must curious about the difference between running in SAP Fiori launchpad and standalone. In this chapter, we'll explore how SAPUI5 applications are loaded into SAP Fiori launchpad. You'll also learn about the most significant service provided by SAP Fiori launchpad: navigating and communicating between SAPUI5 applications.

### 3.1 Architecture

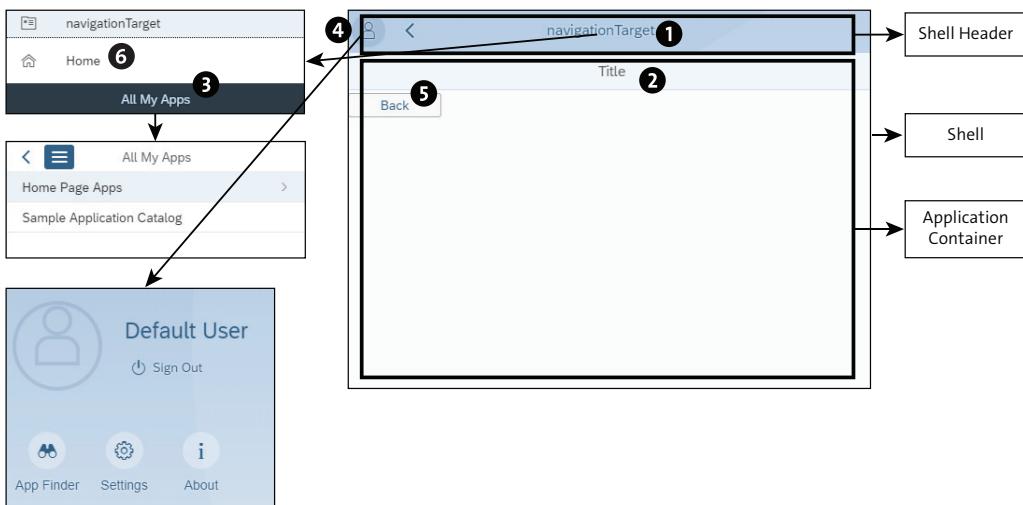
First, you need to know how SAPUI5 applications are loaded into SAP Fiori launchpad. SAP Fiori launchpad has several types of runtime and supports highly flexible configurations and personalizations. Loading an SAPUI5 application is much more complex than just navigating to a web page.

When you create an SAPUI5 application in SAP Web IDE, an *index.html* file is generated for test purposes. If you click the **Run** button, the default behavior is to open *index.html*—but that's not the case when you run it in SAP Fiori launchpad.

In SAP Fiori launchpad, all applications, regardless of their type (SAPUI5, Web Dynpro, or SAP Screen Personas), are loaded by an application container. For SAPUI5 applications, the container creates a component container (`sap.ui.core.ComponentContainer`), then loads the `Component.js` file as a component (`sap.ui.core.Component`) into it.

Therefore, though the end user uses SAP Fiori launchpad like a website, internally the system isn't navigating from one page to another. It loads applications into the UI area and prevents the end user from accessing the “real” URL of your SAP Fiori application.

In SAP Fiori launchpad, your application doesn't run in full-screen mode. A container is rendered for providing some fundamental functions, like navigation to home or displaying the title of your app. We call this container a *shell*. [Figure 3.1](#) illustrates the relationship between the shell and the application container.



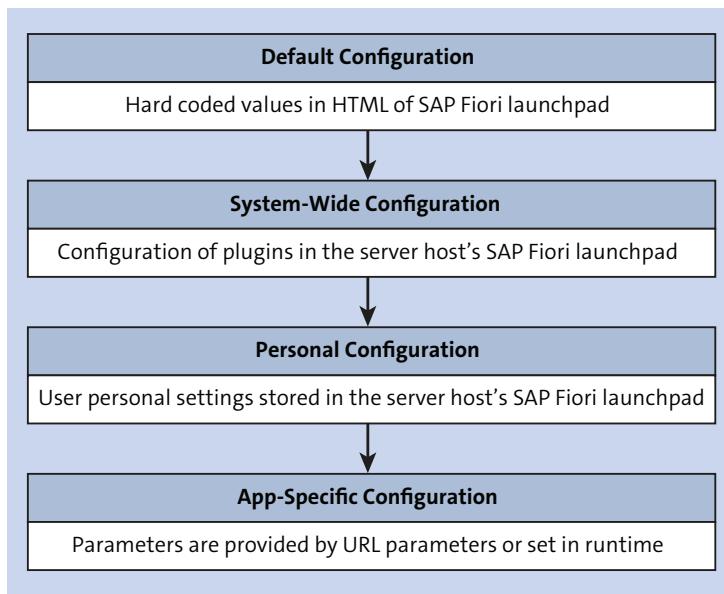
**Figure 3.1** SAPUI5 Application Running in Shell

The following are most important features provided by the shell control:

- ❶ The title of your page control is provided for your own SAPUI5 application.
- ❷ A title for your application is displayed in the shell container.
- ❸ Clicking **All My Apps** opens a pop-up to help users navigate to other apps.
- ❹ Navigation is provided to the Me Area.
- ❺ A **Back** button behaves like the **Back** button of the browser. It's useful when SAP Fiori launchpad is running in full-screen mode—such as when running in the SAP Fiori client on a mobile device.
- ❻ Navigate is provided to the home page.

In addition to running SAPUI5 applications, the shell is also responsible for rendering the SAP Fiori launchpad home page, which holds many groups and tiles.

Various configurations can affect the result of shell rendering. The determination of values for configuration parameters is complex because the parameters can be set at different levels. In addition, the manner of configuration is not identical in different versions of SAP Fiori launchpad. In general, there are four levels of configuration, regardless of the platform SAP Fiori launchpad is running on, as illustrated in [Figure 3.2](#). The settings in a lower level will override settings in a higher level.



**Figure 3.2** Possible Levels of Configuration

To separate the problem of rendering a shell and determining the effective value of configuration parameters, SAP isolated the job of rendering the shell. The *shell renderer* is responsible for rendering the shell according to effective parameters. Another component called the *shell container* is responsible for determining the effective parameters and loading the shell renderer using these parameters.

SAP Fiori launchpad also provides useful JavaScript APIs to help SAPUI5 developers perform common tasks centered on communicating with SAP Fiori launchpad or the backend system.

The purposes of these services are as follows:

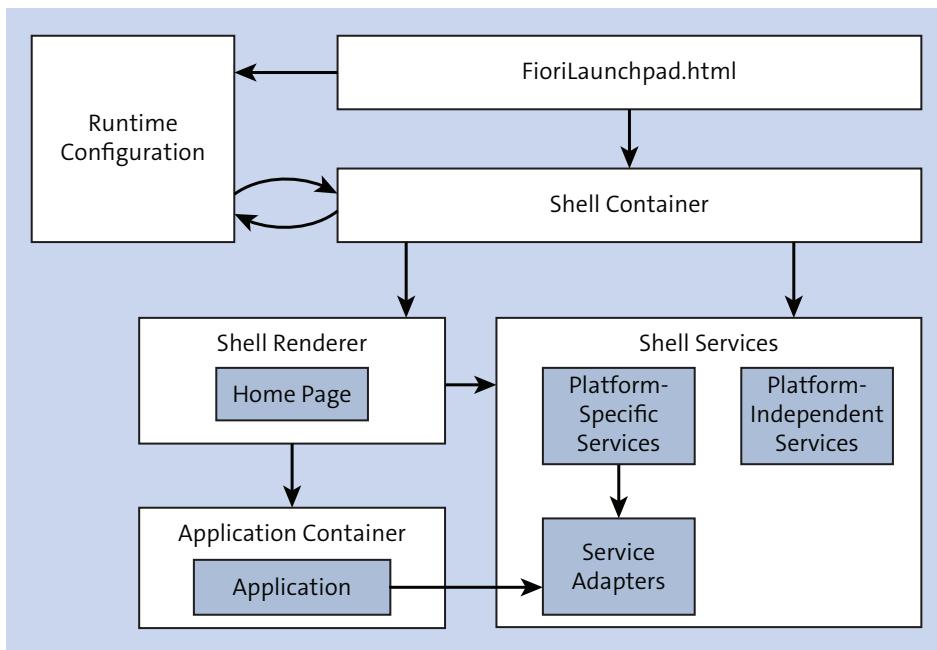
- Simplify the consumption of common functions, especially those that need support from the backend
- Provide a unified API for SAPUI5 developers, regardless the type of backend system

Some of these services are designed for all versions of SAP Fiori launchpad, and others are for a specific version.

#### Note

We'll discuss services and their availability in [Chapter 4](#).

[Figure 3.3](#) shows a summary of the architecture. It represents the whole journey of your SAPUI5 app displayed on a user's screen when it's running in SAP Fiori launchpad.



**Figure 3.3** Building Blocks of SAP Fiori Launchpad

## 3.2 Intent-Based Navigation

As we've discussed before, all SAPUI5 apps are loaded into SAP Fiori launchpad. From the end user's perspective, he always accesses the web address for SAP Fiori launchpad. What happens when user clicks on a tile in SAP Fiori launchpad? How does SAP Fiori launchpad know which app the user wants to access and load it into the shell?

The answer is intent. In SAP Fiori launchpad, most tiles are hyperlinks to an intent. *Intent* is the order a user sends to SAP Fiori launchpad. SAP Fiori launchpad will load the correct application according to *target mapping*, which sets the link between an intent and the technical information for an application.

An intent is composed of two segments: semantic object and action. A dash (-) is used to separate them. A typical intent should look like that in Figure 3.4.

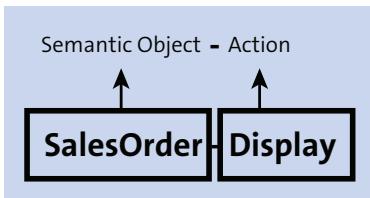


Figure 3.4 Example Intent

### Tip: Naming Conventions for Intents

Technically, you can use any string as an intent. But as a best practice, you should use an “Object-action” pattern that will make the intent meaningful. The object should start with an uppercase letter, and the action should start with a lowercase one.

To access an intent, add a hash sign (#) after the URL of SAP Fiori launchpad's index page, followed by the intent.

Here's an example of the URL to access an application through an intent:

`https://<address>:<port>/sap/bc/ui2/flp#SalesOrder-Create`

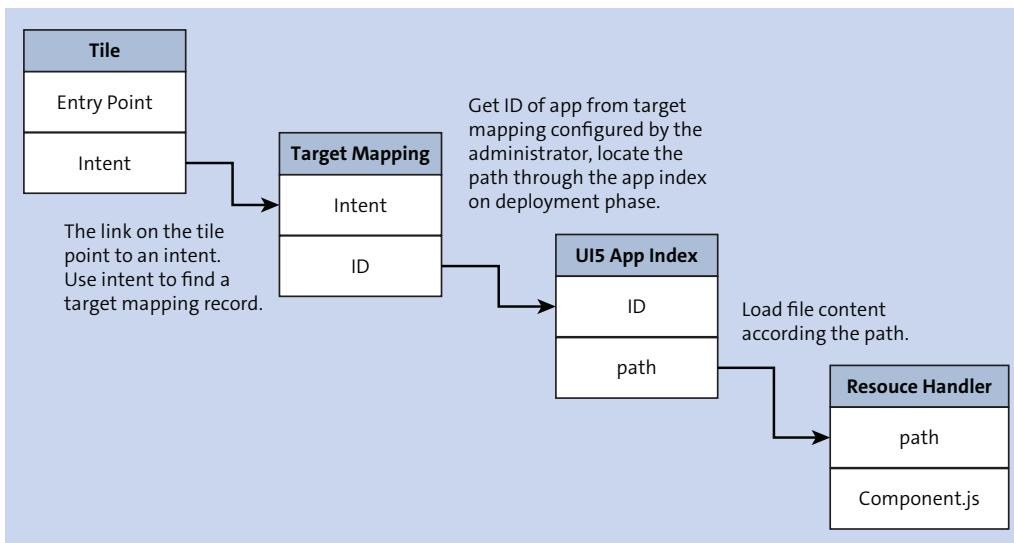
To locate your app via the intent, a target mapping needs to be created. The target mapping defines the runtime information for an intent. For an SAPUI5 application, you need to provide the namespace of your app, which you define during the creation of your SAPUI5 project. You can find it in the **ID** field in the **General** panel of application descriptor (“sap.app”/ID in *manifest.json*).

**Note**

Don't change the ID. It's used everywhere in your project. Use **Search • Advanced Repository Search** to replace all occurrences in your project. To change the ID, you need detailed knowledge of SAPUI5.

A link between the ID of an SAPUI5 app and its address is created automatically when the app is deployed on the target system. In general, you don't need to maintain it manually.

Figure 3.5 describes the whole process of intent-based navigation.



**Figure 3.5** Intent-Based Navigation Process

**Note**

In a real-world scenario, when lots of apps run in SAP Fiori launchpad, more concepts, like groups and catalogs, need to be introduced for better management of SAP Fiori apps. For detailed knowledge of those concepts, we recommend reading [Chapter 4](#) of *SAP Fiori Implementation and Development*, published by SAP PRESS.

After figuring out how intent-based navigation works, you might wonder why SAP made this solution so complex. Isn't it enough to just provide the link to the app in the tile? However, there are some benefits provided by intent-based navigation:

- It makes the entry point of an application stable even if the physical address of the application file changes.
- Target mapping is device-dependent, so you can provide different applications for different devices even if the end users access the same URLs.
- If you're developing an app that displays sales order items, you need to navigate to material master data when a user clicks the material's link; you may know nothing about the material app, or even if the material app is finished yet, so how can you finish your app regardless of the status of the material app? How can you manage the dependencies among thousands of apps in SAP S/4HANA? Intent-based navigation can separate a project or system into many simple SAPUI5 applications. Those applications are connected loosely by intent, and that's essential for complex SAP Fiori projects.

## 3.3 Embedding SAPUI5 Applications

In this section, we'll walk through several hands-on exercises to present the full process of developing and testing an SAPUI5 application for SAP Fiori launchpad. You'll first develop a simple SAPUI5 application and test it in an SAP Fiori launchpad sandbox environment. Then you'll learn how to propose tile information and intent for it when deploying it in the SAP Cloud Platform Portal service.

### 3.3.1 Testing SAPUI5 Apps

To deploy your SAPUI5 application into the target system and run it in SAP Fiori launchpad, you need to complete lots of configuration steps according to the target platform. For convenience of testing your app in SAP Fiori launchpad, you can run your app in a sandbox environment provided by SAP Web IDE.

Now, let's create an SAPUI5 application and run it in an SAP Fiori launchpad sandbox:

1. Enter SAP Web IDE full stack and switch into the **Development** section.
2. In the menu, go to **File • New • Project from Template**.
3. Choose **SAPUI5 Application**, then click **Next** (see [Figure 3.6](#)).

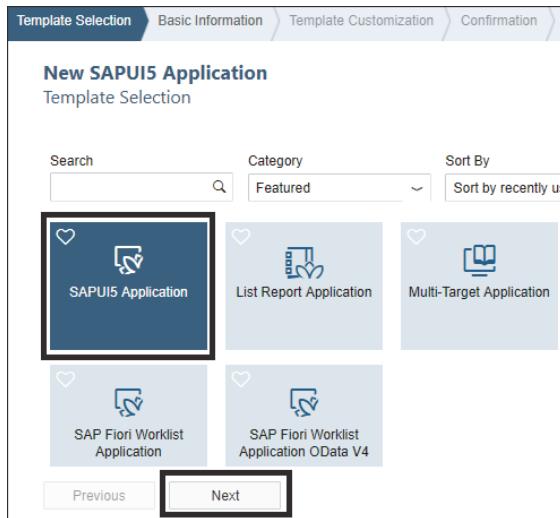


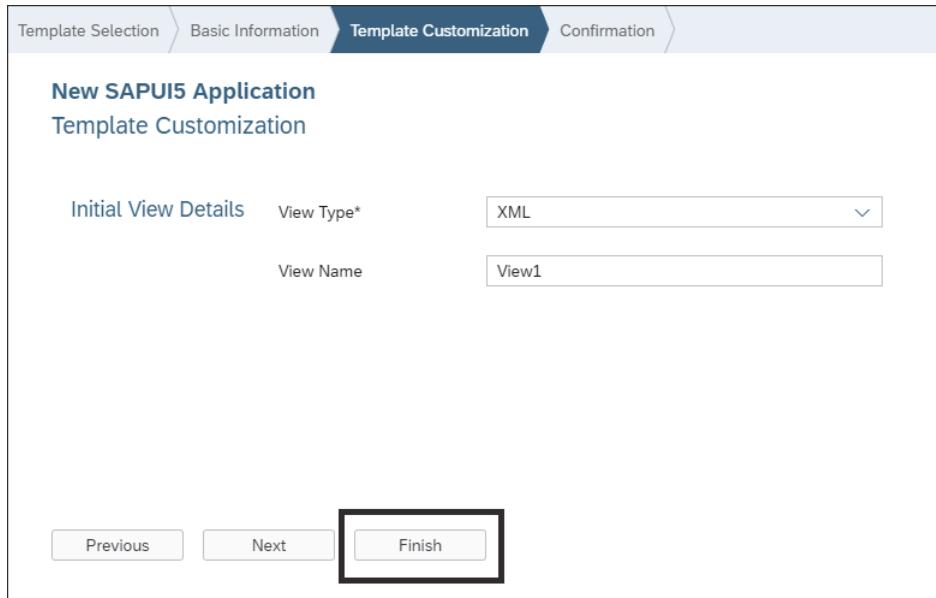
Figure 3.6 Template Selection

4. For **Project Name**, enter “UI5InFLP”. For **Namespace**, enter “flpdev”. Click **Next** (see Figure 3.7).

The screenshot shows the 'Basic Information' step of creating a new SAPUI5 application. The tabs at the top are 'Template Selection', 'Basic Information' (active), 'Template Customization', and 'Confirmation'. The title 'New SAPUI5 Application' and the sub-section 'Basic Information' are displayed. The 'Project Name\*' field contains 'UI5InFLP'. The 'App Descriptor Data' section shows the 'Namespace\*' field containing 'flpdev'. Navigation buttons 'Previous' and 'Next' are at the bottom.

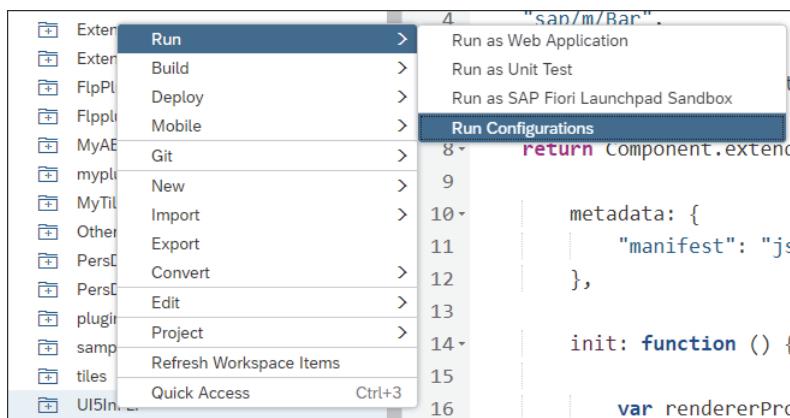
Figure 3.7 Basic Information

5. On the **Template Customization** page, leave the default values and click **Finish** (see [Figure 3.8](#)).



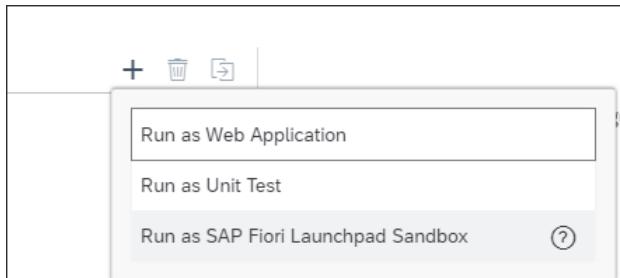
**Figure 3.8** Template Customization

6. Right-click **UI5InFLP** and choose **Run • Run Configurations...** from the context menu (see [Figure 3.9](#)).



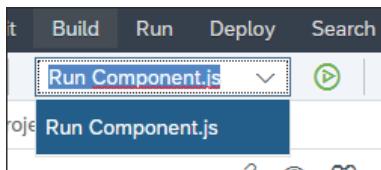
**Figure 3.9** Context Menu: Run Configurations

7. Click **+** and choose **SAP Fiori Launchpad Sandbox**, then click **OK** to save the settings (see [Figure 3.10](#)).



**Figure 3.10** Run as SAP Fiori Launchpad Sandbox

8. Now the **Run Component.js** option is available in the dropdown list on the toolbar when your project is selected, as shown in [Figure 3.11](#). You can also access the configuration through the **Run** options in the context menu. Click to run your SAPUI5 app.



**Figure 3.11** Run Configuration List and Run Button

9. Now you can see your SAPUI5 application running in SAP Fiori launchpad.

### 3.3.2 Provisioning Application Title and Description

The title of your app will be displayed on the title bar in the shell control as well as in the title bar of the browser. A description may appear on the tile as well. The values are determined according the configuration set by your administrator. However, you should provide a default value. The value you've provided in the app will be used as the default value when you deploy it on SAP Cloud Platform. If you deploy the app on SAP NetWeaver AS ABAP, it will not have any effect. The administrator can check the value you provided and configure SAP Fiori launchpad accordingly.

In the application descriptor (*manifest.json*), you will find the **Title** and **Description** fields in the **General** tab. You can also find this information in the source code under "sap.app", as shown in [Figure 3.12](#).

The diagram illustrates the connection between the SAPUI5 application descriptor (*manifest.json*) and the SAP Fiori Launchpad's **Description Editor**. On the left, the **Source Code** pane shows the JSON structure of *manifest.json*, specifically the `sap.app` section. A red box highlights the `title` and `description` properties. On the right, the **Description Editor** pane shows the corresponding configuration in the **General** tab. A red box highlights the **Title** and **Description** fields. A double-headed arrow indicates the bidirectional relationship between the two representations.

```
{
  "_version": "1.8.0",
  "sap.app": {
    "id": "flpdev.ExtensionDemo",
    "type": "application",
    "i18n": "i18n/i18n.properties",
    "applicationVersion": {
      "version": "1.0.0"
    },
    "title": "{{appTitle}}",
    "description": "{{appDescription}}",
    "sourceTemplate": {
      "id": "ui5template.basicSAPUI5ApplicationProject",
      "version": "1.40.12"
    }
  }
},
```

**Figure 3.12** Title and Description in *manifest.json*

Notice that these two fields are pointing to a field in *i18n/i18n.properties*; that means they are translatable. The simplest way of modifying them is to change *i18n/i18n.properties*, as shown in [Figure 3.13](#).

The screenshot shows the SAP Fiori Launchpad's **Resource Bundle** editor for the `i18n.properties` file. A red box highlights the three entries: `title=Title`, `appTitle=UI5InFLP`, and `appDescription=App Description`.

```
1 title=Title
2 appTitle=UI5InFLP
3 appDescription=App Description
```

**Figure 3.13** App Title and Description in Resource Bundle

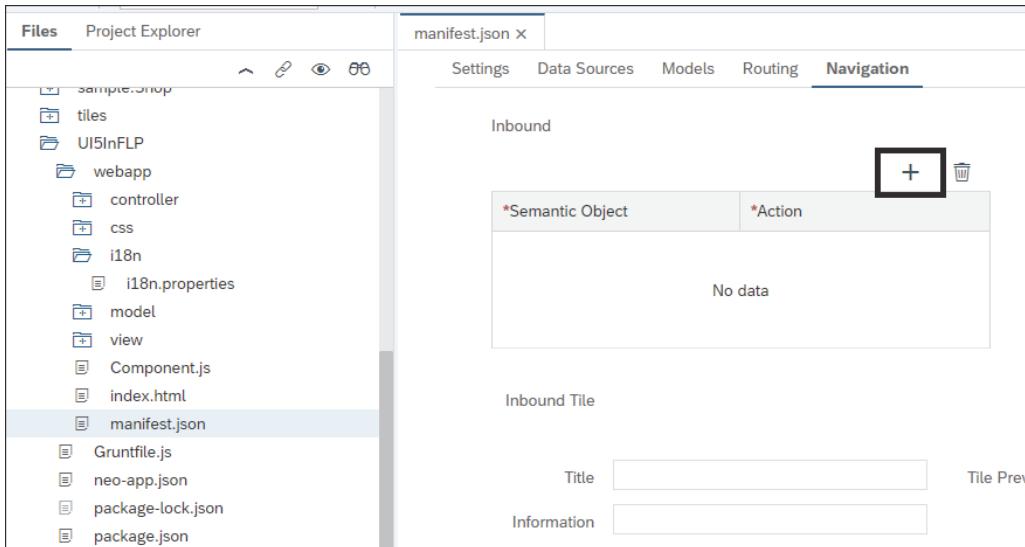
### 3.3.3 Proposing an Intent

The intent and other information, like titles on tiles, are determined at the configuration level. However, as a developer, you can create several suggestions (though normally you would create just one) at development time in the application descriptor (*manifest.json*). The value you suggest will be used in the testing sandbox environment and provide a default value when deploying your app on SAP Cloud Platform.

Currently, SAP NetWeaver AS ABAP can't extract intent information you provide at the time of deployment. However, it can be a guide for the administrator to decide which intent should be used for your app.

Now, switch back to project UI5InFLP. Follow these steps to add intent information for this app and check it by deploying it on SAP Cloud Platform:

1. Open the application descriptor by double-clicking **manifest.json**, then switch to the **Navigation** section. Click the **+** button under **Inbound** (see [Figure 3.14](#)).

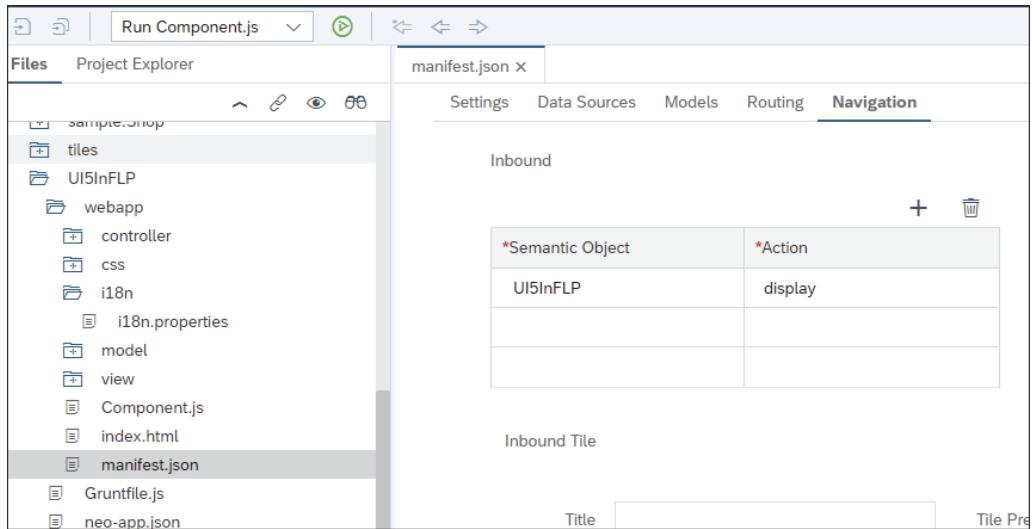


**Figure 3.14** Navigation Configuration in Application Descriptor

2. A new line has been added to the table under the plus sign; override the default values according to [Table 3.1](#). Click or press **Ctrl**+**S** to save the file (see [Figure 3.15](#)).

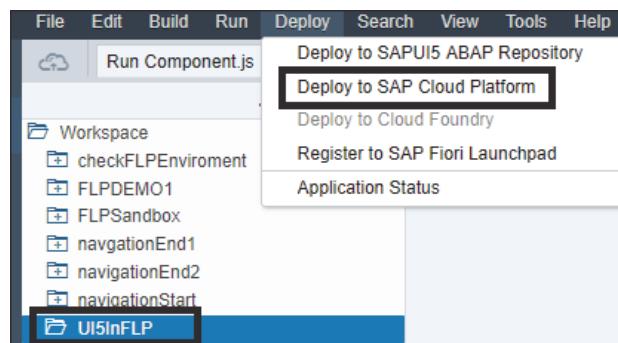
| Field           | Value    |
|-----------------|----------|
| Semantic Object | UI5InFLP |
| Action          | display  |

**Table 3.1** Parameters for Intent



**Figure 3.15** Add Inbound Navigation

3. Deploy your app to SAP Cloud Platform by choosing **Deploy • Deploy to SAP Cloud Platform** (see [Figure 3.16](#)). Make sure the root folder of your project is selected.



**Figure 3.16** Deploy Menu

4. Leave all fields set to their defaults and click **Deploy** (see [Figure 3.17](#)).

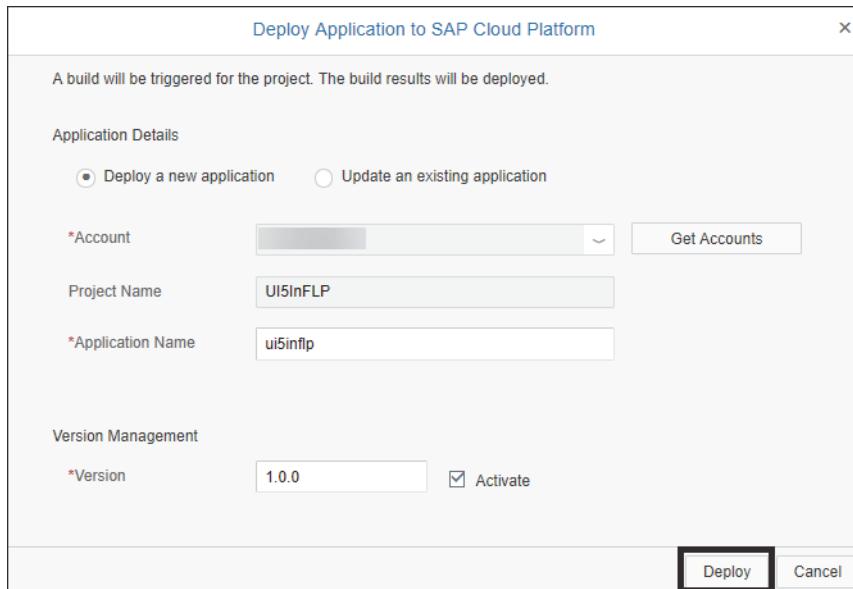


Figure 3.17 Deploy Application to SAP Cloud Platform

5. Wait about one minute for the deployment to process, then click **Register to SAP Fiori Launchpad** (see Figure 3.18).

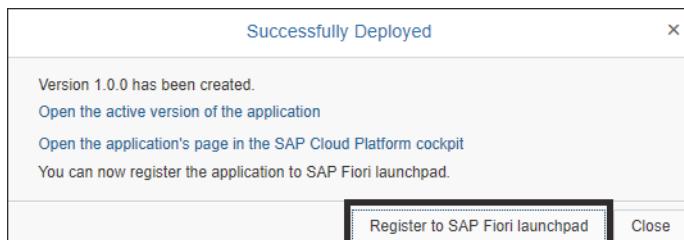


Figure 3.18 Register to SAP Fiori Launchpad

6. In the first registration step, note that the intent you've provided in the application descriptor is set to the default value (see Figure 3.19). You can also change it in this step. Once you're satisfied with the value, click **Next**.

The screenshot shows the 'General Information' step of a configuration wizard titled 'Register to SAP Fiori Launchpad'. The 'Assignment' and 'Confirmation' steps are visible in the breadcrumb navigation. The form contains fields for 'Provider Account' (set to 'trial (flpportal)'), 'Application Name' (set to 'flpdev.UI5InFLP.UI5InFLP'), and a 'Description' field which is empty. Below these, there is a table for 'Intent' configuration:

|                          | Semantic Object | Action  |
|--------------------------|-----------------|---------|
| <input type="checkbox"/> | UI5InFLP        | Display |

A link 'More information on intent properties.' is present below the table. At the bottom are 'Previous' and 'Next' buttons.

Figure 3.19 Intent Configuration

7. Keep the default values for the following windows, clicking **Next** until you reach the final window, where you'll click **Finish**. Click the **Open SAP Fiori Launchpad** link in the pop-up shown in [Figure 3.20](#).

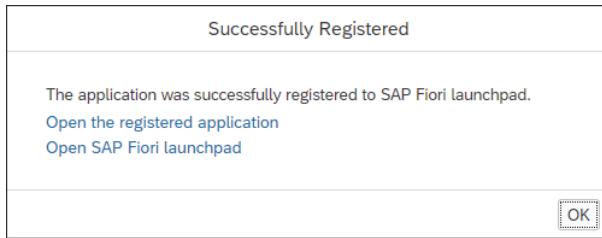


Figure 3.20 Successfully Registered

### Tip

If you provided tile information in the navigation section of the application descriptor, that tile information will be displayed as the default values in this wizard. But for now, let's focus on intent.

8. In the SAP Fiori launchpad, click the tile for your app. After you enter the app, notice that the intent, prefixed by a hash sign (#), is appended to the end of the URL in the address bar of your browser, as shown in [Figure 3.21](#).



Figure 3.21 Intent in URL Address

## 3.4 Navigation between SAPUI5 Applications

As you've seen, you access SAPUI5 apps in SAP Fiori launchpad by intent, and this also works when you navigate from app to app.

Of course, you can provide a link address, as we've discussed before, but the best way to navigate using intent is to use the cross-application navigation service provided by SAP Fiori launchpad. Besides navigation, it provides a set of supportive functions like URL resolution, backward navigation, and environment checks.

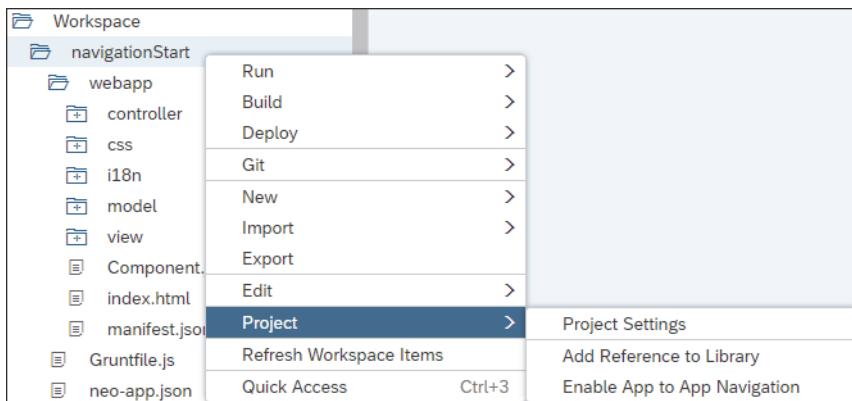
You already know how to test a single SAPUI5 application in the SAP Fiori launchpad sandbox, so now let's cover how to embed multiple apps in one sandbox environment. Then, you'll learn not only how to navigate between apps using the navigation service provided in SAP Fiori launchpad, but also how to write robust code when your app isn't running in SAP Fiori launchpad or you're navigating to an intent that doesn't exist. Also, you'll learn how to better manage and use your navigation targets in the app descriptor file.

### 3.4.1 Set Up Test Environment for Cross-Application Navigation

Before writing any code, you need to create a standalone SAP Fiori launchpad sandbox that can hold more than one SAPUI5 application. The following steps import two SAPUI5 applications and set up an SAP Fiori launchpad sandbox:

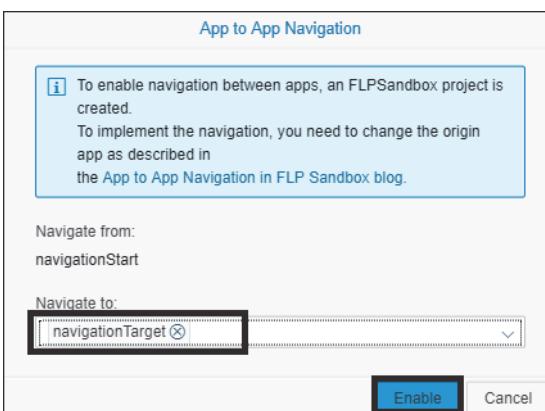
1. Import the `navigationStart` and `navigationTarget` projects from the resources available on this book's webpage ([www.sap-press.com/4556](http://www.sap-press.com/4556)). You can import a project using the menu path **File • Import • File or Project**.
2. Examine the source code of the `navigationStart` project. In the **Navigation** panel of the application descriptor, you'll see that the default intent for this project is `NavigationStart-display`. In `view/Main.view.xml` you'll find a button with text `Go`. Also in `controller/Main.controller.js` you'll find an `onGo` method without any code. You'll add your cross-application navigation service call in this method.

3. Examine the source code of the navigationEnd project. In the **Navigation** panel of application descriptor, you will see the default intent for this project is NavigationEnd-display. In view/Main.view.xml you'll find a **Back** button. Also in controller/Main.controller.js you'll find an **onBack** method without any code. You'll add your code for backward navigation in this method.
4. Right-click the **navigationStart** project. Choose **Enable App to App Navigation** in the context menu, as shown in [Figure 3.22](#).



[Figure 3.22](#) Enable App to App Navigation

5. From the **Navigation To** dropdown list, select the **navigationTarget** project and click **Enable**, as shown in [Figure 3.23](#).



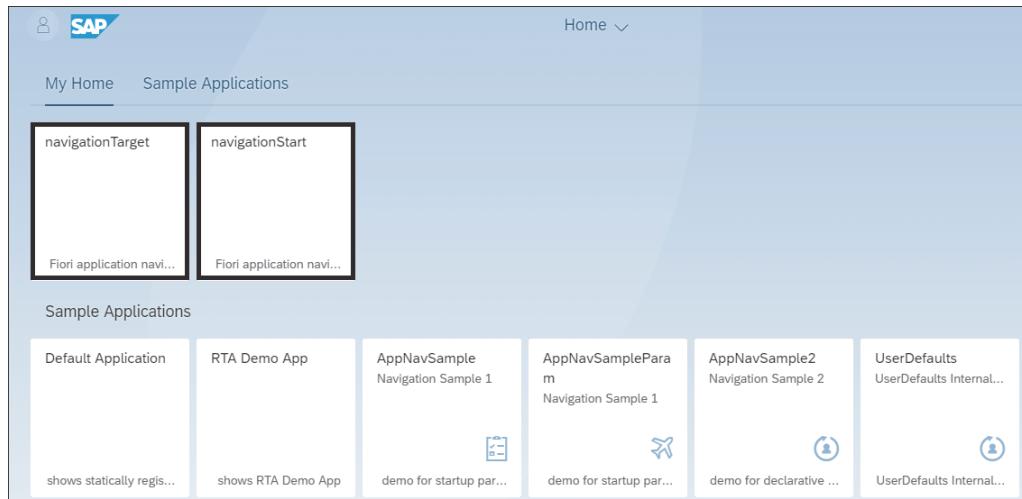
[Figure 3.23](#) Choose Navigation Target

6. Wait for the request to be processed, and a new project called **FLPSandbox** will be created for you.
7. Open *appconfig/fioriSandboxConfig.json* and examine the code generated in this file. The file contains intents and runtime information for the two projects you've added, as shown in [Listing 3.1](#).

```
"NavigationTarget-display": {  
    "additionalInformation": "SAPUI5.Component=flpdev.navigationTarget",  
    "applicationType": "URL",  
    "url": "../../navigationTarget",  
    "description": "Fiori application navigat...",  
    "title": "Demo:Target of Navigation"  
},
```

**Listing 3.1** Navigation Start: App Configuration in SAP Fiori Launchpad Sandbox Environment

8. Select the **FLPSandbox** project and click  on the toolbar. An SAP Fiori launchpad containing the two apps will appear, as shown in [Figure 3.24](#).



**Figure 3.24** SAP Fiori Launchpad with Two SAPUI5 Apps

9. Click each tile in turn and make sure the two apps look like those in [Figure 3.25](#).

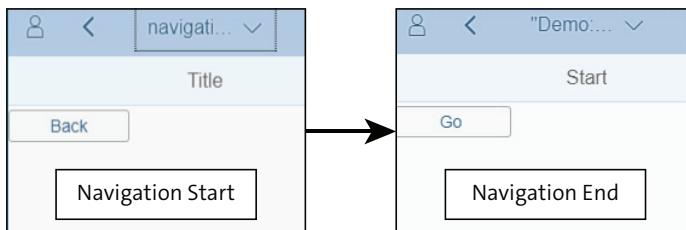


Figure 3.25 Navigation Start and End Apps

### Warning

Although the menu option you used to create an SAP Fiori launchpad sandbox was **Enable App to App Navigation**, it did nothing navigation-wise. You need to implement all cross-application navigation operations manually.

## 3.4.2 Calling Navigation Services

To call navigation services, you need to create an instance of a cross-application navigation service. Use following code to do so:

```
var service = sap.ushell.Container.getService("CrossApplicationNavigation")
```

After getting the reference of the service, call the `toExternal` method to navigate to an intent. You need to pass a JSON object with a property `target` to specify the intent you want navigate to, as shown in [Listing 3.2](#).

```
service.toExternal({
  target:{
    semanticObject:<An Semantic Object>
    action:<action>
  }
})
```

### [Listing 3.2](#) Navigate to Intent

Now, let's implement cross-application navigation in the test environment, as follows:

1. Locate the **navigationStart** project. Under **webapp • controller**, open **Main.controller.js** (see [Figure 3.26](#)) by double-clicking it.



Figure 3.26 Structure of navigationStart Project

2. Implement the `onGo` method as shown in [Listing 3.3](#).

```
onGo: function(oEvent) {  
    //This code was generated by the layout editor.  
    //Get instance of cross application navigation service  
    var service = sap.ushell.Container.getService("CrossApplicationNavigation");  
  
    //Call external intent  
    service.toExternal({  
        target: {  
            semanticObject:"navigationTarget",  
            action:"display"  
        }  
    });  
},
```

[Listing 3.3](#) navigationStart: Calling Cross-Application Navigation Service

3. Save your changes by clicking the  button on the toolbar.
4. Select the **FLPSandbox** project and run it by clicking on .
5. Click the **Demo: Start Point of Navigation** tile, as shown in [Figure 3.27](#).

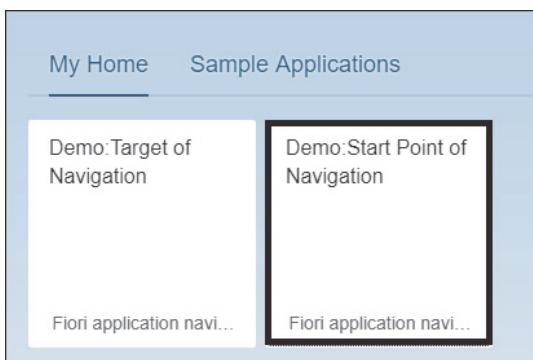
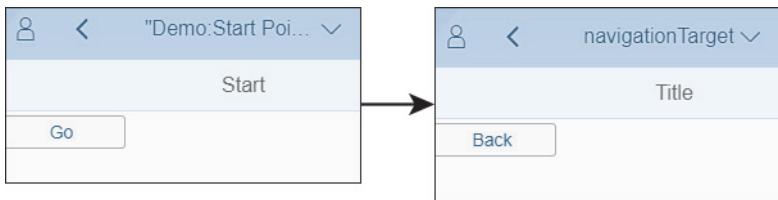


Figure 3.27 SAP Fiori Launchpad

6. Click the **Go** button. You should see the page for the navigationEnd project, as shown in [Figure 3.28](#).



**Figure 3.28** navigationEnd Project Page

### 3.4.3 Test Supportability

To make sure the intent is reachable, you need to check it before navigation. The cross-application navigation service provides a `isNavigationSupported` method to check if intent navigation to a specific target is valid. That will help you prevent a user navigating to a page that doesn't exist.

The method accepts an array of intents and returns a JavaScript deferred object. In this deferred object, you can get an array for supportability for each intent you've provided.

[Listing 3.4](#) shows the sample code for testing availability of intents.

```
service.isNavigationSupported([
    {
        semanticObject: "<semanticObject1>",
        action: "action1"
    },
    {
        semanticObject: "<semanticObject2>",
        action: "<action2>"
    }
]).done(function(result) {
    //deal with result
    //The result is something like [ {supported:true},{supported:false}]
}).fail(function() {
    //Some fatal error occurred. All intents can not be resolved
});
```

**Listing 3.4** Checking Availability of Intent

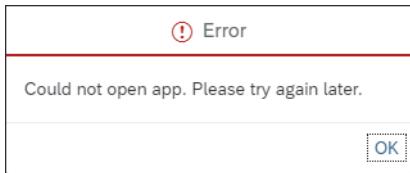
Now return to your code. Open the controller of the main view in `navigationStart/webapp/controller/Main.controller.js` and perform the following steps:

1. Change the method call of `toExternal` and set the action property under target to `dummy`, as shown in [Figure 3.29](#).

```
service.toExternal({
  target: {
    semanticObject: "NavigationTarget",
    action: "dummy" //Change to dummy, it will not success since the intent is not exist
  }
});
```

**Figure 3.29** Change to Intent That Cannot Be Resolved

2. Try the navigation again and find that an error message (see [Figure 3.30](#)) is displayed as a pop-up window.



**Figure 3.30** Error Message

3. Return to the main controller and add an `onInit` method. This method will be called when the view is initializing. Check the `NavigationTarget-dummy` intent; if it isn't available, call the `_disableButton` method, either when this intent doesn't exist or when a fatal error occurs. Don't forget to bind the callback method to the controller using `.bind(this)`. The relevant code is in [Listing 3.5](#).

```
onInit: function () {
  var service = sap.ushell.Container.getService("CrossApplication
  Navigation");
  var oTarget = {
    semanticObject: "navigationTarget",
    action: "dummy"
  };
  service.isNavigationSupported([oTarget]).done(function (o) {
    if (!o[0].supported) {
      //Call _disableButton when the intent is not supported
      this._disableButton();
    }
  })
}
```

```

        }.bind(this)).fail(function () {
            //fatal error
            this._disableButton();
        }.bind(this));
    },

```

**Listing 3.5** `navigateStart`: Checking Availability of Intent

4. Add the `_disableButton` method. In this method, set the `enable` property of the button to `false`, as shown in [Listing 3.6](#).

```

_disableButton:function(){
    var oBtn = this.byId("btnGo");
    oBtn.setEnabled(false);
}

```

**Listing 3.6** `navigationStart`: Change Enable State of Button

5. Run the `navigationStart` project in FLPSandbox again. You'll find that the **Go** button is disabled (see [Figure 3.31](#)).



**Figure 3.31** Go Button Disabled

6. Change the controller back to the correct intent in the `onGo` method, as well as in `onInit`.

#### 3.4.4 Navigation Back to Previous App

You can use the `backToPreviousApp` method to navigate back. If you've used a router for your development, you should know that most internal navigation will also trigger a change portion of the URL following the `#` sign in your address bar. Unlike the `back` function provided by a browser, the `backToPreviousApp` method will skip all records in the browsing history caused by the internal navigation of your current app and go back to the last state of the previous app.

Before using `backToPreviousApp`, you should use `isInitialNavigation` to determine if a “previous” app exists. If the method returns `true`, it indicates that the end user entered this app directly by a link, and you shouldn't provide the **Back** button or should redirect to the home page when the user clicks the button.

Now locate the source code of the navigationTarget project. Open the main controller in *navigationTarget/webapp/controller/main.controller.js* and perform the following steps:

1. Implement the `onBack` method to call the `backToPreviousApp` method of the cross-application navigation service using the code in [Listing 3.7](#).

```
onBack: function(oEvent) {
    //This code was generated by the layout editor.
    var service =
        sap.ushell.Container.getService("CrossApplicationNavigation");
        service.backToPreviousApp();
},
```

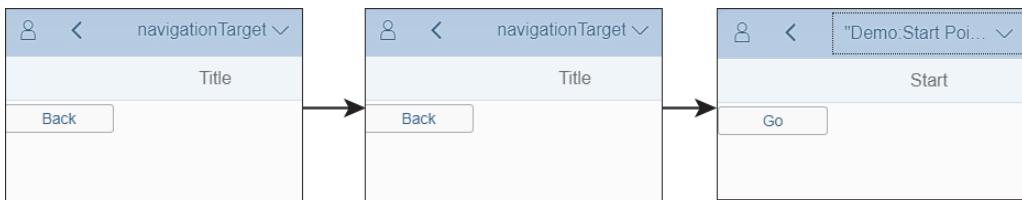
**Listing 3.7** `navigationEnd: Back to Previous App`

2. Implement the `onInit` method and use the code in [Listing 3.8](#) to make sure the button is disabled if a user entered this app directly via a link.

```
onInit: function () {
    var service = sap.ushell.Container.getService("CrossApplication
        Navigation");
    if (service.isInitialNavigation()) {
        this._disableButton();
    }
},
_disableButton: function () {
    var oBtn = this.byId("btnBack");
    oBtn.setEnabled(false);
},
```

**Listing 3.8** `navigationBack: Disable Button if No Previous App Exists`

3. Run FLPSandbox again, enter the Demo: Start Point of Navigation app, and click the **Go** button. Then click the **Back** button to navigate back, as shown in [Figure 3.32](#).



**Figure 3.32** Navigation Flow

4. Close the page and run FLPSandbox again and enter the Demo: Target of Navigation app. After the page has loaded, copy the URL address, open a new window in the browser, paste the URL into the address bar, and press **Enter**. The **Back** button should be disabled, as shown in [Figure 3.33](#), because you entered this page directly.

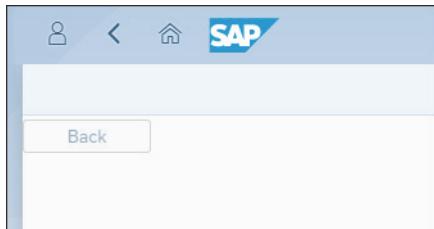


Figure 3.33 Disabled Due to Direct Link Access

### 3.4.5 Configuring Navigation Targets in App Descriptor

You may have noticed that a warning message is displayed when you call the `toExternal` method. The message warns you that it's risky to call the cross-application navigation service with a hard-coded intent. Technically, you can invoke the service anywhere in the application, but if you do so, it's hard to know and manage which applications you need to navigate to.

To make your code more readable and manageable, the best practice is to configure all possible cross-application navigation in the application descriptor (`manifest.json`). Then you can get the intent information from the metadata of your application.

In the application, under `sap.app`, you can add a `crossNavigation` property to describe all cross-application navigation information. The `outbound` property should look like that in [Listing 3.9](#).

```
"outbounds":{  
    "<target_name>":{  
        "semanticObject": "<semantic_object>",  
        "action": "<action>"  
    }  
},  
<other_target>  
....  
},
```

Listing 3.9 Configure Navigation Target in App Descriptor

In any controller, you can get this information using the following code:

```
var oTarget = this.getOwnerComponent().getManifestEntry("/sap.app/  
crossNavigation/outbounds/<target_name>")
```

### Tip

Currently, there's no tool or runtime support for the configuration of an outbound navigation target. However, it's mentioned in the application descriptor document. Use this method as a best practice for now; if a tool or runtime support is added later, then you won't need to change the code.

To implement this best practice for the `navigationStart` project, first double-click `manifest.json` under **navigationStart • webapp** to open the application descriptor. Then proceed with following steps:

1. Click the **Code Editor** button shown in [Figure 3.34](#) to open the code editor.



**Figure 3.34** Code Editor Button

2. Locate the "sap.app" segment and add an "outbounds" property under `crossNavigation`. Don't forget to add a comma after the "inbounds" property.
3. Under "outbounds", add a property with the name you want. Fill it with detailed information as in [Listing 3.10](#) and save the file.

```
{  
    "_version": "1.8.0",  
    "sap.app": {  
        ...  
  
        "crossNavigation": {  
            "inbounds": {  
                ...  
            },  
            "outbounds": {  
                ...  
            }  
        }  
    }  
}
```

```

        "myTarget":{
            "semanticObject":"navigationTarget",
            "action":"display"
        }
    }
}
...
}

```

**Listing 3.10** navigationStart: Configure Target in App Descriptor**Note**

If you haven't added inbound navigation before, you need to add the `crossNavigation` property manually.

4. Locate the main controller view by double-clicking **Main.controller.js** under **navigationStart • webapp • controller**.
5. Change the implementation of the `onGo` method as shown in [Listing 3.11](#).

```

onGo: function(oEvent) {
    //This code was generated by the layout editor.
    //Get instance of cross application navigation service
    var service =
sap.ushell.Container.getService("CrossApplicationNavigation");
    //Get target infromation from manifest.json
    var oTarget = this.getOwnerComponent().getManifestEntry("/sap.app/
crossNavigation/outbounds/myTarget");
    //Call external intent
    service.toExternal({
        target: oTarget
    });
},

```

**Listing 3.11** navigationStart: Get Intent Information from App Descriptor

6. Change the implementation of the `onInit` method as shown in [Listing 3.12](#).

```
onInit: function() {
    var service =
sap.ushell.Container.getService("CrossApplicationNavigation");
    var oTarget = this.getOwnerComponent().getManifestEntry("/sap.app/
crossNavigation/outbounds/mytarget");
    service.isNavigationSupported([oTarget]
).done(function(o) {
    if(!o[0].supported){
        //Call _disableButton when the intent is not supported
        this._disableButton();
    }
}).bind(this)).fail(function() {
    //fatal error
    this._disableButton();
}.bind(this));

},
```

**Listing 3.12** navigationStart: Checking Intent from App Descriptor

7. Try the app again; it should work just like before—but now your code is more maintainable!

---

### Warning

Now your code is pretty safe and robust. The one last thing you haven't implemented is checking if the service itself is available. We'll discuss this topic in [Chapter 4](#). There's a unified way to check supportability for all services provided by SAP Fiori launchpad.

---

## 3.5 Passing Parameters between Apps

When navigating from one app to another, you need to pass parameters among them. Although you can pass parameters in URLs, there's a more elegant way to pass parameters between components. The SAP Fiori launchpad API will handle all the necessary tasks.

To send parameters, you need to add a `params` property when calling the `hrefForExternal` or `toExternal` method. [Listing 3.13](#) shows the code for passing a parameter based on the `onGo` method from the previous section.

```
onGo: function(oEvent) {  
    //This code was generated by the layout editor.  
    //Get instance of cross application navigation service  
    var service =  
        sap.ushell.Container.getService("CrossApplicationNavigation");  
    //Get target information from manifest.json  
    var oTarget = this.getOwnerComponent().getManifestEntry("/sap.app/  
crossNavigation/outbounds/myTarget");  
    //Call external intent  
    service.toExternal({  
        target: oTarget,  
        params:{<parameterName>:<value>}  
    });  
,
```

**Listing 3.13** navigationStart: Passing Parameters to Navigation Target

To receive a parameter, you can use the `getComponentData` method of `UIComponent`.

[Listing 3.14](#) shows sample code for receiving a parameter based on the `onInit` method of the `navigationTarget` project from the previous section.

```
onInit:function(){  
    var service =  
        sap.ushell.Container.getService("CrossApplicationNavigation");  
    if(service.isInitialNavigation()){  
        this._disableButton();  
    }  
    //Get component  
    var oComponent = this.getOwnerComponent();  
    //Get component data  
    var oComponentData = oComponent.getComponentData();  
    //Get a map object for parameters  
    var oParams = oComponentData.startupParameters;  
    //Get specific parameter  
    var oValue = oParams.<paramName>;  
  
},
```

**Listing 3.14** navigationEnd: Receiving Parameters

### 3.6 Summary

In this chapter, you learned how an SAPUI5 application is loaded into SAP Fiori launchpad. We also explored the reasoning behind and use of intent-based navigation. You've now learned how to embed your app into SAP Fiori launchpad, and you've also learned how to navigate and passing information between apps.

In the next chapter, you'll learn more about the services provided by SAP Fiori launchpad as JavaScript APIs to perform daily operations related to SAP Fiori launchpad.

# Chapter 4

## Client-Side Services

*Client-side services help simplify your work when you're communicating with the runtime. In this chapter, we'll talk about how to use these services.*

Let's say you're trying to communicate with the runtime, such as when retrieving the user name of the current user. Or say you want to get information about the SAP Fiori launchpad itself—for example, the number of tile groups the user has. How can you perform these activities?

In general, your SAPUI5 application speaks to the backend system via an OData service. Do you need to develop an OData service to provide this service? If you've already done so, the next question to ask is how many versions of this OData service are needed if you want your SAPUI5 app to run on all versions of SAP Fiori launchpad, considering that all of them have different backend technologies.

Fortunately, SAP Fiori launchpad provides a set of JavaScript APIs. Those APIs can help you communicate with the runtime easily. These APIs are also backend-independent: most of them work in the same way whatever your backend system is. That makes transporting your SAPUI5 app from on-premise to the cloud and vice versa possible.

Technically, all communication with SAP Fiori launchpad is implemented by client-side services. In the previous chapter, you used a cross-app navigation service, which is used to extend SAP Fiori launchpad, as you will learn in the next chapter. However, in this chapter, we'll cover the most useful services to help you build a better SAPUI5 app, as follows:

- **User info services**

User info services can help you get common information about a current user, regardless which version of SAP Fiori launchpad you're running on. You can also use this service to get information about the current language, theme, and other settings. It also supports updating user preferences.

**■ Bookmark services**

Often, users want to save a current page as a shortcut. Of course, they can save such pages using the functions provided by their browsers, but this option doesn't always work. An SAPUI5 app may run in a web view of a mobile app, which doesn't have a bookmark function.

Bookmark services can simplify the work of saving the current state of an app as a tile and putting it on the launch page. The user can go back to this state by clicking the created tile.

**Note**

To use this service, you must make sure the state can be accessed by a deep-linking URL, which is implemented by router classes.

**■ Personalization services**

Personalization services help you save a user's state information, such as a shopping cart, multi selection, or table settings, into the user's local storage and as a backup in the backend system. That data will be valid even if the user logs off and on again. With this service, you save data that has one of the following properties:

- Is not your company's business data
- Is so complex that it can't be saved via URL parameters

Although client-side services are very powerful, you need to consider the following factors when using them:

- The SAPUI5 application must run in SAP Fiori launchpad as a component.
- Not all versions of SAP Fiori launchpad support all service methods, so you need to test them before using them in production environments.

## 4.1 User Info Service

In this section, you'll learn how to use the user info service. This is quite a simple service, and you'll learn not only how to use it but also the general steps to create and test a client-side service. We'll start with a demo of a simple service and then, building on the knowledge you've just gained, walk through the user info service API. We'll end the section by creating a more complex application using this service.

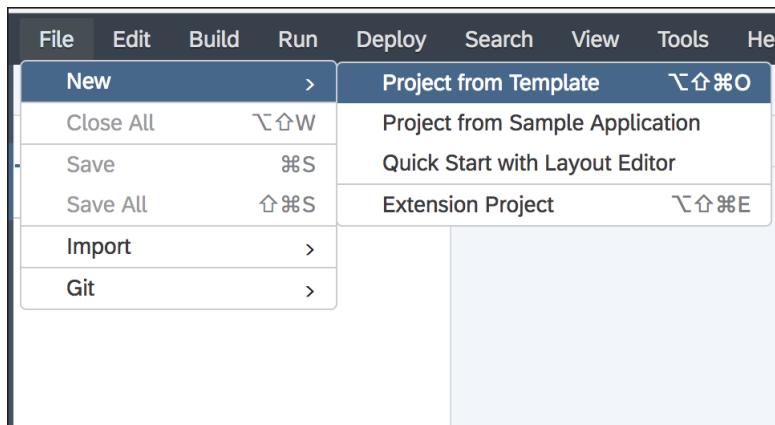
### 4.1.1 Creating and Testing a Simple SAPUI5 Application

Let's start with a very simple example. This example uses the service to get the ID of current user. We'll test the example both in the SAP Fiori launchpad sandbox and in SAP Cloud Platform Portal.

#### Creating an SAPUI5 Application

To show how to use this service, we'll walk through creating a simple SAPUI5 application. The application contains some fields and a button. When you click the button, you'll get information about the current user of the relevant API. To begin, follow these steps:

1. Open the SAP Web IDE full-stack version (as described in [Chapter 2, Section 2.2](#)).
2. Create a new project by selecting **File • New • Project From Template** in the menu, as shown in [Figure 4.1](#).



**Figure 4.1** Create Project from Template

3. Select **SAPUI5 Application** as the template and click **Next**, as shown in [Figure 4.2](#).
4. For **Project Name**, enter “UserInfoSimple”; for **Namespace**, enter “flpdev” (see [Figure 4.3](#)).

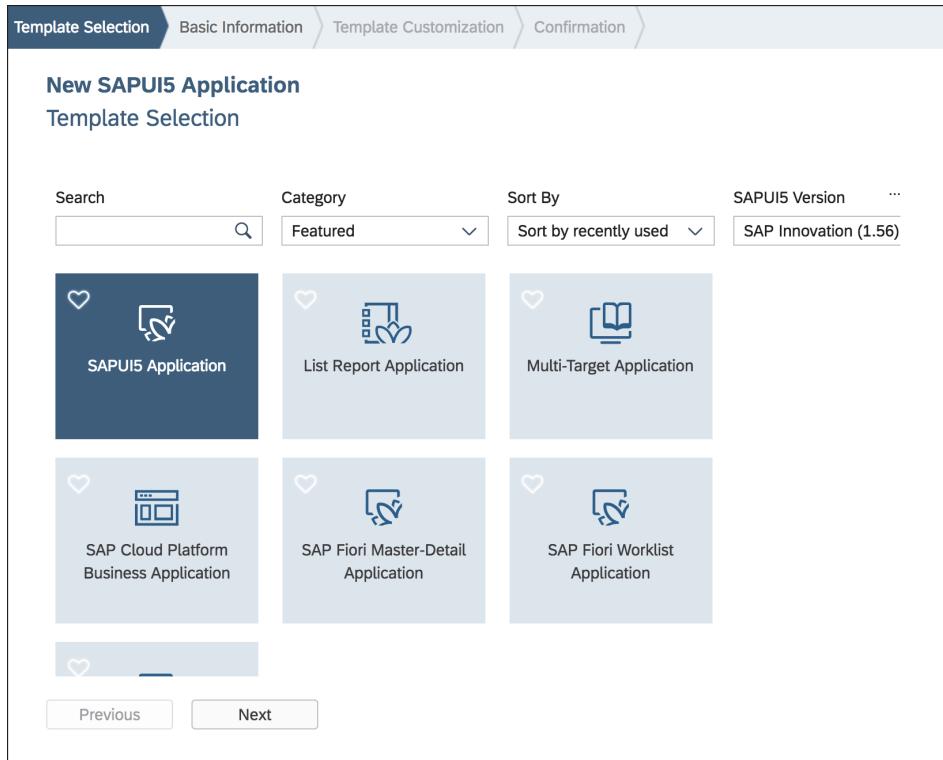


Figure 4.2 Template Selection

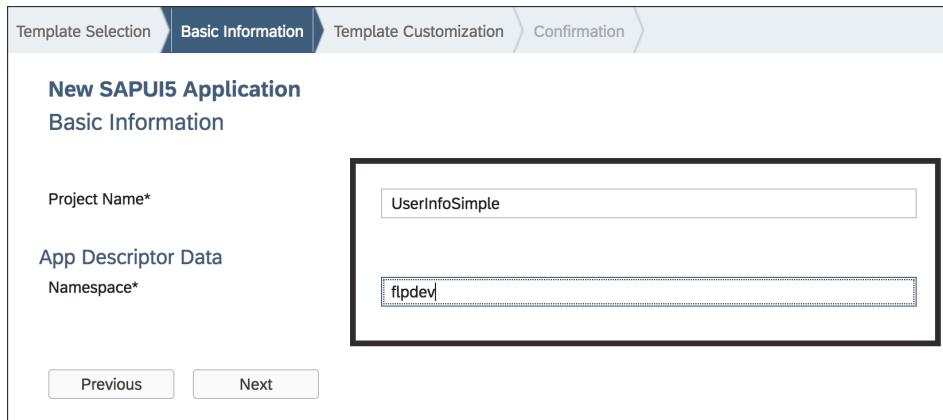


Figure 4.3 Basic Information

5. Change the **View Name** to “Main” and click **Finish**, as shown in Figure 4.4.

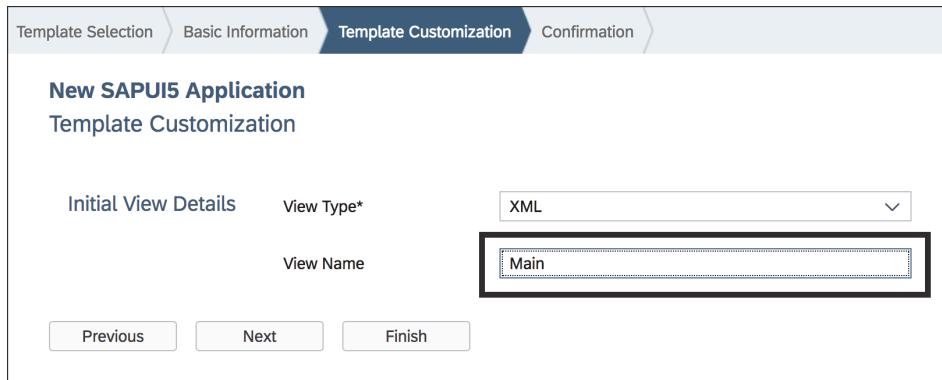


Figure 4.4 Template Customization

6. Expand the folder structure and follow the path to **UserInfoSimple • webapp • view • main.view.xml**; right-click and choose **Open Layout Editor**, as shown in Figure 4.5.

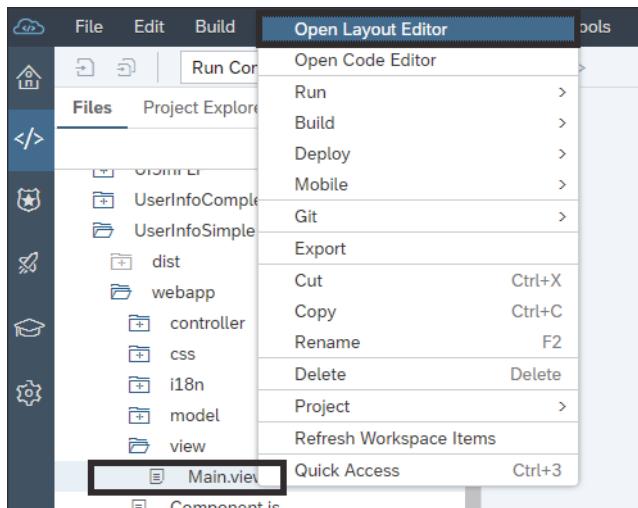


Figure 4.5 Open with Layout Editor

7. In the layout editor, enter “Label” in the search field and press **Enter**, then drag a **Label** control to the UI area (see Figure 4.6). Change the **Text** property to “User ID” in the **Properties** panel on the right side.

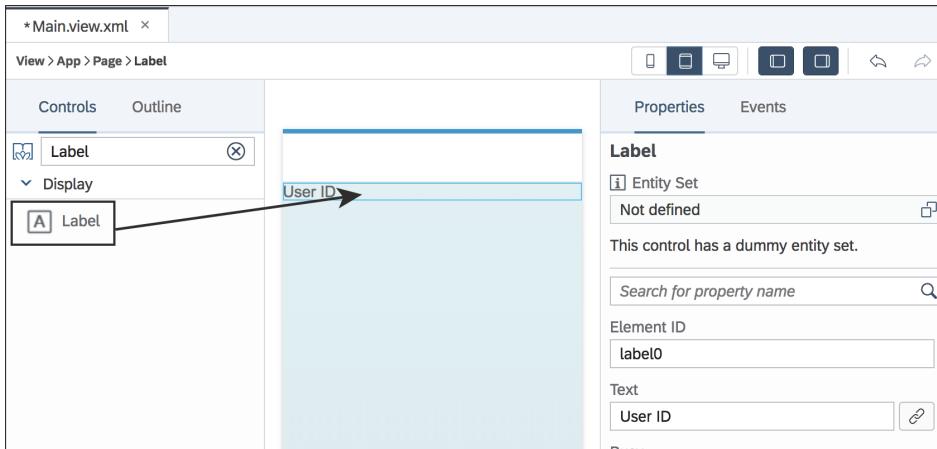


Figure 4.6 Drag and Drop Label

8. Drag an **Input** control under the label and set the **Element ID** property to “`inpUserID`”, as shown in [Figure 4.7](#).

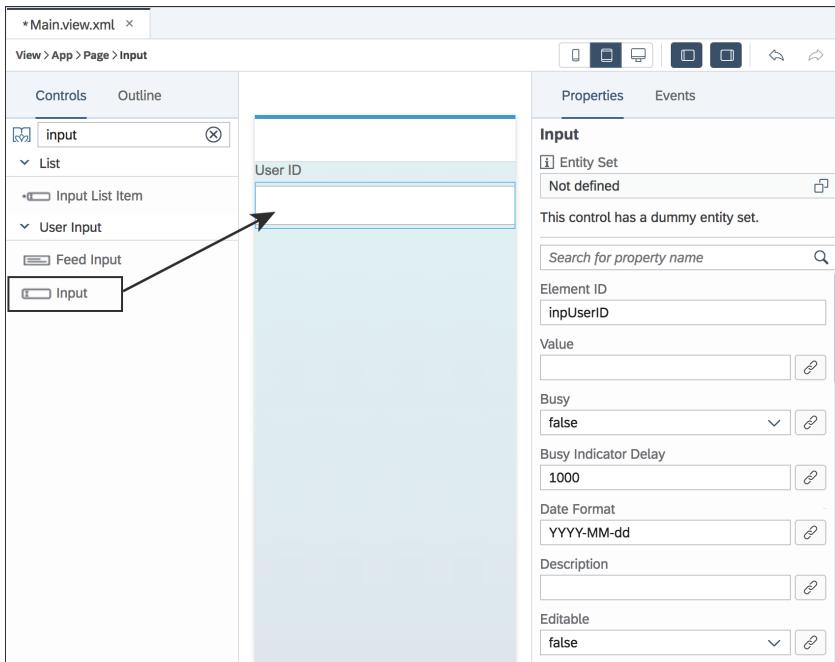


Figure 4.7 Drag and Drop Input Control

9. Drag and drop a **Button** control under the input control and change its **Text** property to “Get”, as shown in [Figure 4.8](#).

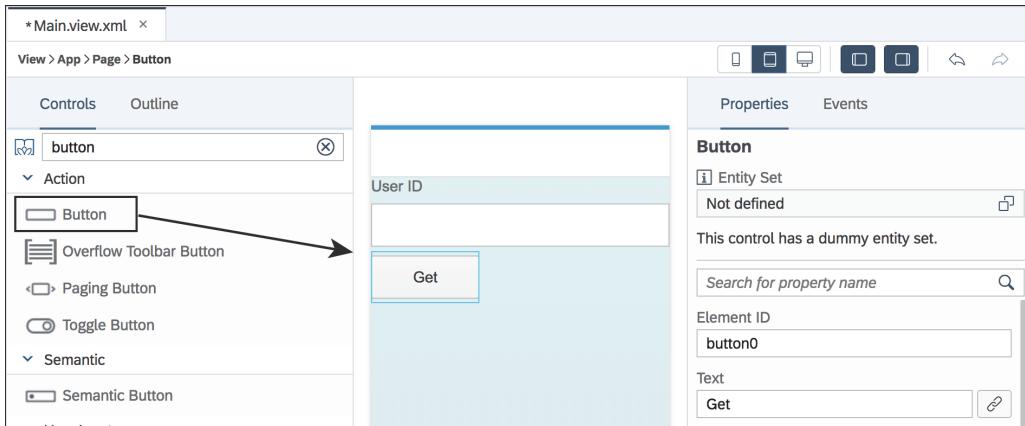


Figure 4.8 Add Button Control

10. Click the **Events** panel, then click next to the **Press** field and select **New Function**, as shown in [Figure 4.9](#).

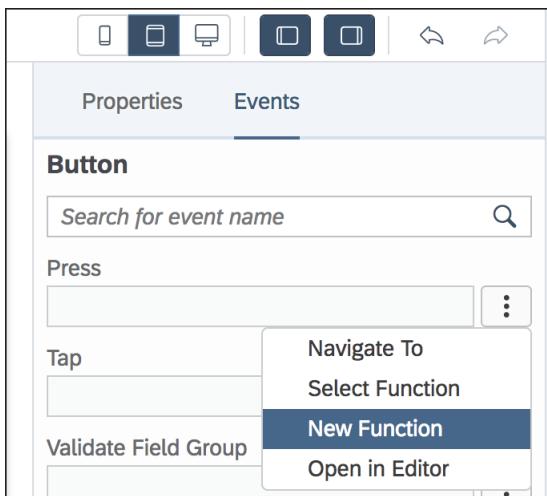
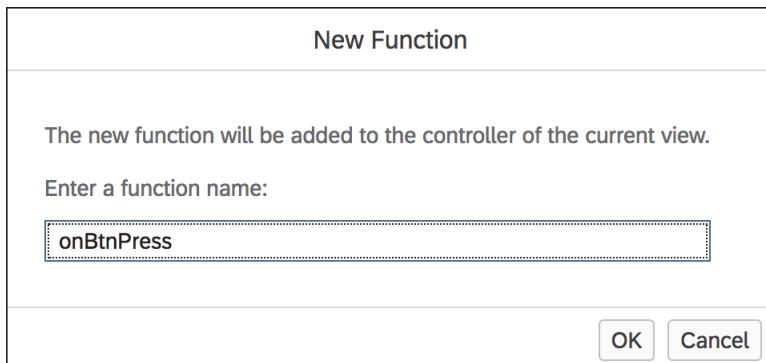


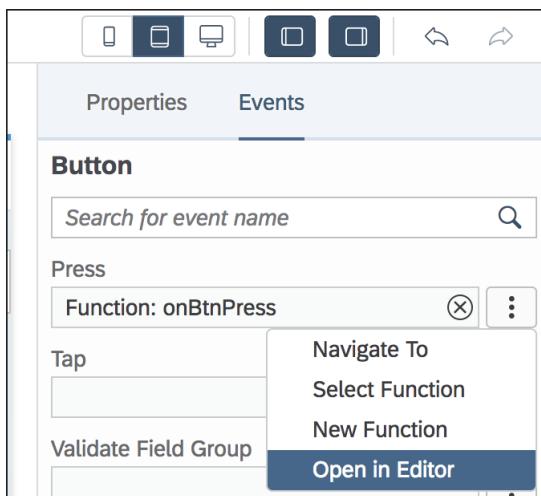
Figure 4.9 Create Event Handler

11. In the **Enter a Function Name** field, enter “onBtnPress” as your function name and click **OK**, as shown in [Figure 4.10](#).



**Figure 4.10** Enter Function Name

12. Next to the **Press** field, click **:** again and select **Open in Editor**, as shown in [Figure 4.11](#).



**Figure 4.11** Open in Editor

13. Add the code shown in [Listing 4.1](#) to the onBtnPress method.

```
//This code was generated by the layout editor.

        //Get a reference of the service
        //
Use sap.ushell.Container.getService to get a reference of the service
        //
To avoid runtime error, you need to test the existence of each level object
using &
        var oService = sap && sap.ushell && sap.ushell.Container &&
sap.ushell.Container.getService("UserInfo");
        //
If the environment doesn't support this service, you need to notify user or
write some fall-back code
        if (!oService) {
            //The best practice is to import it in the sap.ui.define part
            //I put it here just for simplify the procedure
            jQuery.sap.require("sap.m.MessageBox");
            sap.m.MessageBox.error("User Info service is not supported");
            return;
        }

        //

//Get user id from the service
var sUserID = oService.getId();

//Set the input field according to User ID
this.byId("inpUserID").setValue(sUserID);
```

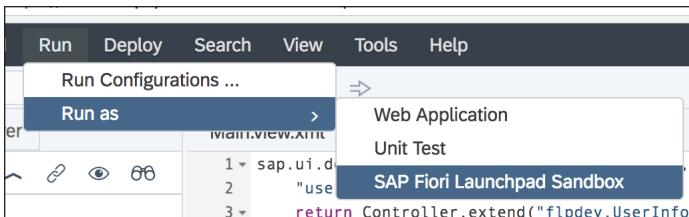
**Listing 4.1** UserSimple: Calling User Info Service in main.controller.js

14. Save all your changes by clicking  in the header toolbar of SAP Web IDE.

### Testing an SAPUI5 Application

Now that you've created the service, you can test it in both the SAP Fiori launchpad sandbox and in SAP Cloud Platform Portal by following these instructions:

1. Follow the menu path **Run • Run As • SAP Fiori Launchpad Sandbox**, as shown in [Figure 4.12](#).

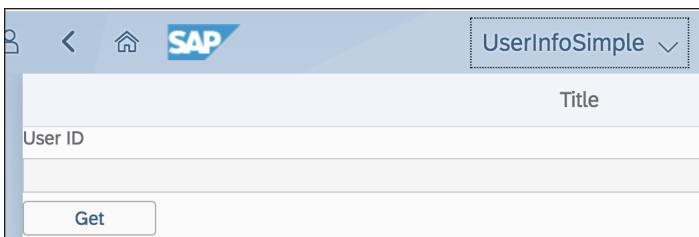


**Figure 4.12** Run in SAP Fiori Launchpad Sandbox

### Tip

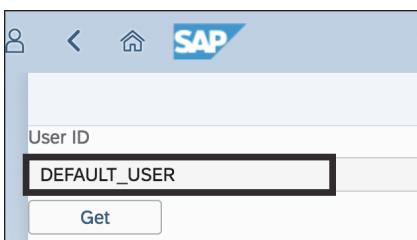
If it's your first time using SAP Web IDE, you'll need to change the settings in Chrome to allow pop-up windows. A pop-up dialog in SAP Web IDE will guide you through this process.

2. The result should look like [Figure 4.13](#). On this page, click **Get**.



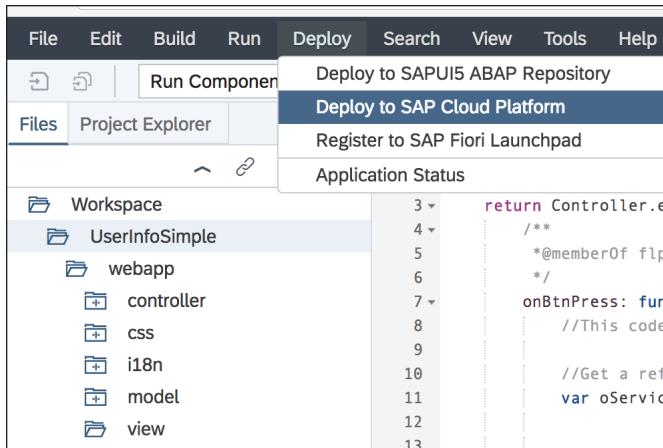
**Figure 4.13** UserInfoSimple in Sandbox

3. You'll see that **DEFAULT\_USER** appears in the **User ID** input box, as shown in [Figure 4.14](#). Because you're in a sandbox environment, the ID is always going to be **DEFAULT\_USER**.



**Figure 4.14** Test Results: SimpleUserInfo in Sandbox

4. Now it's time to test the service in SAP Cloud Platform Portal to see the real results. Switch back to SAP Web IDE and click **Deploy • Deploy to SAP Cloud Platform**, as shown in [Figure 4.15](#).



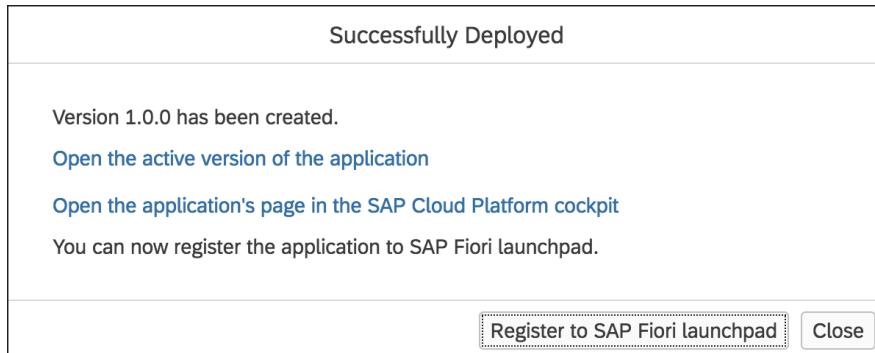
[Figure 4.15](#) Menu Path to Deploy Service

5. For the **Application Name** field, type in “userinfoSimple”, then click **Deploy** (see [Figure 4.16](#)).

The screenshot shows the 'Deploy Application to SAP Cloud Platform' dialog box. It has two tabs: 'Application Details' and 'Version Management'. The 'Application Details' tab is active, showing fields for Account (selected account), Project Name (UserInfoSimple), and Application Name (userinfoSimple). The 'Version Management' tab shows a Version field (1.0.0) with an 'Activate' checkbox checked. At the bottom are 'Deploy' and 'Cancel' buttons.

[Figure 4.16](#) Deploy Application to SAP Cloud Platform

6. After a while (generally a few minutes), a pop-up will appear to tell you that the deployment was successful. Click **Register to SAP Fiori Launchpad**, as shown in [Figure 4.17](#).



**Figure 4.17** Successfully Deployed Pop-Up

7. In the **General Information** step, click **Next** without any making any changes, as shown in [Figure 4.18](#).

The screenshot shows the 'Register to SAP Fiori Launchpad' interface. At the top, there's a navigation bar with tabs: 'General Information' (which is highlighted in dark blue), 'Tile Configuration', 'Assignment', and 'Confirmation'. Below the tabs, the section title 'Register to SAP Fiori Launchpad' is displayed in bold blue text, followed by 'General Information' in a smaller gray font. There are four input fields: 'Provider Account' (set to 'trial (flpportal)'), 'Application Name' (set to 'flpdev.UserInfoSimple.UserInfoSimple'), 'Description' (empty), and 'Intent' (a table with two rows). The 'Intent' table has columns: 'Semantic Object' (checkboxes for 'Semantic Object' and 'UserInfoSimple') and 'Action' (checkbox for 'Display'). At the bottom left is a link 'More information on intent properties.', and at the bottom right are 'Previous' and 'Next' buttons.

**Figure 4.18** Register to SAP Fiori Launchpad: General Information

8. In the **Tile Configuration** step, set **Title** to “User Info Test” and **Subtitle** to “simple”, then click **Next** (see [Figure 4.19](#)).

The screenshot shows the 'Tile Configuration' step of the 'Register to SAP Fiori Launchpad' wizard. The top navigation bar includes tabs for 'General Information', 'Tile Configuration' (which is highlighted in blue), 'Assignment', and 'Confirmation'. The main content area is titled 'Register to SAP Fiori Launchpad' and 'Tile Configuration'. It contains four input fields: 'Type' (Static), 'Title' (User Info Test), 'Subtitle' (simple), and 'Icon' (sap-icon://approvals). To the right of these fields is a preview window showing the tile configuration with the title 'User Info Test' and subtitle 'simple' checked. At the bottom are 'Previous' and 'Next' buttons.

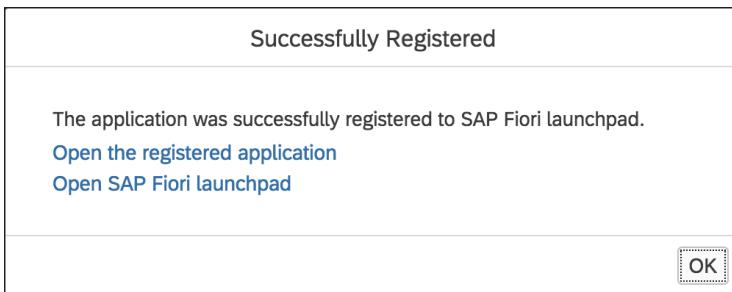
[Figure 4.19](#) Register to SAP Fiori Launchpad: Tile Configuration

9. In the **Assignment** step, click **Finish** without making any changes, as shown in [Figure 4.20](#).

The screenshot shows the 'Assignment' step of the 'Register to SAP Fiori Launchpad' wizard. The top navigation bar includes tabs for 'General Information', 'Tile Configuration', 'Assignment' (which is highlighted in blue), and 'Confirmation'. The main content area is titled 'Register to SAP Fiori Launchpad' and 'Assignment'. It contains three dropdown fields: 'Site' (DemoSite), 'Catalog' (Sample Catalog), and 'Group' (Sample Group). At the bottom are 'Previous', 'Next', and 'Finish' buttons.

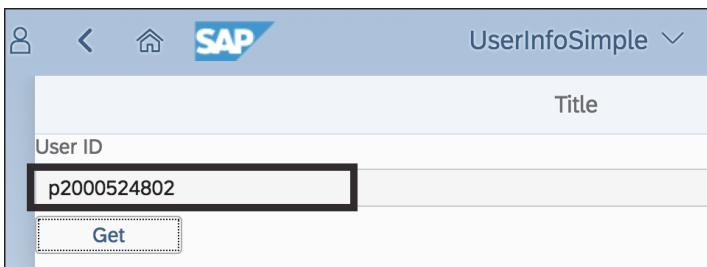
[Figure 4.20](#) Register to SAP Fiori Launchpad: Assignment

10. In the **Successfully Registered** pop-up, click the **Open the Registered Application** link, as shown in [Figure 4.21](#).



**Figure 4.21** Successfully Registered Pop-Up

11. Test your application again, and this time you will find that your SAP Cloud Platform user ID is displayed, as shown in [Figure 4.22](#).



**Figure 4.22** Test Results: Simple User Info in SAP Cloud Platform Portal

### 4.1.2 Exploring the User Info Service API

Now you know how to use the user info service and the `getId()` method. In this section, we'll guide you through analyzing the service in a variety of ways.

First, let's look at the SAPUI5 API documentation, as follows:

1. Open your browser and navigate to <https://ui5.sap.com>. [Figure 4.23](#) shows the landing page of this website.
2. Click **API Reference**. Enter “userinfo” in the search field and press **Enter**. Click **UserInfo** under **sap.ushell.services**, as shown in [Figure 4.24](#). Notice that there is only one method, `getId`, listed for this service.

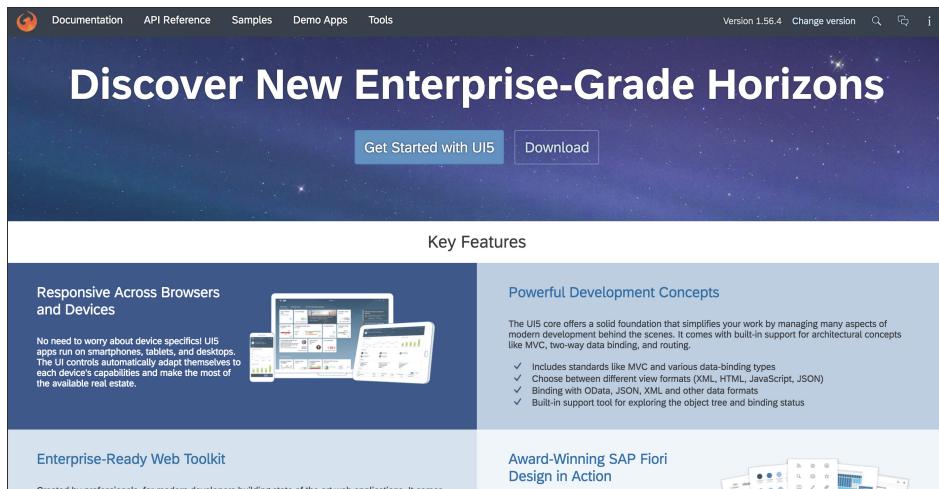


Figure 4.23 Index of SAPUI5 Documentation

**userinfo**

- ✓ sap.ushell.services
- UserInfo

**class sap.ushell.services.UserInfo**

**Control Sample:** N/A      **Visibility:** public

**Extends:** N/A      **Available since:** 1.16.3

**Module:** sap/ushell/services/UserInfo      **Application Component:** N/A

**Overview**      **Constructor**      **Methods**

The Unified Shell's user information service, which allows you to retrieve information about the currently logged-in user.

**Documentation links:**

- [sap.ushell.services.Container#getService](#)

**Constructor**

This method MUST be called by the Unified Shell's container only, others MUST call say instance of the user information service.

```
new sap.ushell.services.UserInfo()
```

**Methods**

**Summary**

| Method                | Description                 |
|-----------------------|-----------------------------|
| <a href="#">getid</a> | Returns the id of the user. |

Figure 4.24 Check Methods for User Info Service

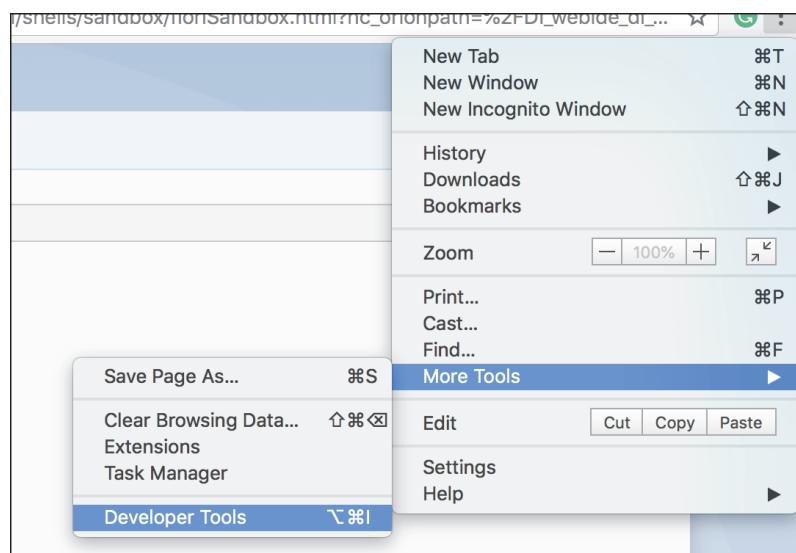
Although there is only one method listed for the user info service, many more methods can be used. As of the time of writing, client-side services aren't fully documented in the SAPUI5 SDK because it's an API provided by SAP Fiori launchpad, not a standard SAPUI5 API. As such, you need to learn how to investigate the service by exploring its objects at runtime.

To begin your exploration, proceed as follows in the Chrome console:

1. Return to SAP Web IDE, switching to the **Edit** page of *Main.controller.js*.
2. Add the following code in the `onBtnPress` method, after all the existing code:

```
//Output the service into console to invesgate it  
console.log(oService);
```

3. Save your code by clicking and then run it by clicking .
4. In Chrome, click and then select **More Tools • Developer Tools**, as shown in [Figure 4.25](#).



**Figure 4.25** Open Developer Tools

5. Click to clear the console, then click the **Get** button on the main screen, as shown in [Figure 4.26](#).
6. This will cause an object to be dumped to the console. Click the small triangle to expand it, as shown in [Figure 4.27](#).

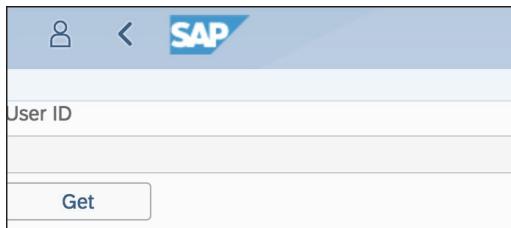


Figure 4.26 Testing User Info Service in Console



Figure 4.27 Dump Results of User Info Service

7. You can see the following five methods in this object (see [Figure 4.28](#)):

- getId
- getLanguageList
- getThemeList
- getUser
- updateUserPreferences

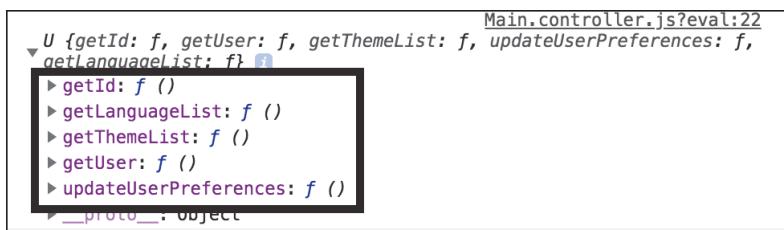


Figure 4.28 Expanded User Info Service

8. Now you can continue writing code in the `onBtnPress` method of `Main.controller.js`, as shown in [Listing 4.2](#), to test all methods—except `updateUserPreferences` because the name itself tells you that it won't return any information if you test it using `console.log()`.

```
//Try methods
    console.log("Testing getId");
    console.log(oService.getId());
    console.log("Testing getLanguageList");
    console.log(oService.getLanguageList());
    console.log("Testing getThemeList");
    console.log(oService.getThemeList());
    console.log("Testing getUser");
    console.log(oService.getUser());
```

#### Listing 4.2 Testing Methods of UserInfo Service

9. Save your changes and refresh the test page, open the console, and click **Get** again. You'll see that the `getId` method returns the ID of the current user, and the `getUser` method returns an object that has lots of methods to retrieve and change various properties of a user.

|   |                                   |
|---|-----------------------------------|
| Testing getId   | <u>Main.controller.js?eval:25</u> |
| DEFAULT_USER  | <u>Main.controller.js?eval:26</u> |
| Testing getLanguageList   | <u>Main.controller.js?eval:28</u> |
|   | <u>Main.controller.js?eval:29</u> |
| ► {state: f, always: f, then: f, promise: f, pipe: f, ...}                                |                                   |
| Testing getThemeList  | <u>Main.controller.js?eval:31</u> |
|   | <u>Main.controller.js?eval:32</u> |
| ► {state: f, always: f, then: f, promise: f, pipe: f, ...}                                |                                   |
| Testing getUser   | <u>Main.controller.js?eval:34</u> |
|   | <u>Main.controller.js?eval:35</u> |
| ► a {getEmail: f, getFirstName: f, getFullName: f, getId: f, getLanguageList: f, ...} [ ] |                                   |
| ► attachOnSetImage: f (f,d)   |                                   |
| ► getAccessibilityMode: f ()  |                                   |
| ► getChangedProperties: f ()  |                                   |
| ► getContentDensity: f ()   |                                   |
| ► getEmail: f ()  |                                   |
| ► getFirstName: f ()  |                                   |
| ► getFullName: f ()   |                                   |
| ► getId: f ()   |                                   |
| ► getImage: f ()  |                                   |
| ► getImageConsent: f ()   |                                   |
| ► getLanguage: f ()   |                                   |
| ► getLanguageBcp47: f ()  |                                   |
| ► getLastname: f ()   |                                   |
| ► getTheme: f (t)   |                                   |
| ► getThemeRoot: f (t)   |                                   |
| ► getTrackUsageAnalytics: f ()  |                                   |
| ► isJamActive: f ()   |                                   |
| ► isLanguagePersonalized: f ()  |                                   |

Figure 4.29 Dump Results of Methods of User Info Service

However, the `getLanguageList` and `getThemeList` methods return nothing but plain JavaScript promise objects. The result should resemble [Figure 4.29](#).

- Now continue by adding the code shown in [Listing 4.3](#) to the end of the `onBtnPress` method of `Main.controller.js`.

```
//Testing methods which returns promise object
```

```
oService.getLanguageList().then(function (oLanguageList) {
    console.log("Testing getLanguageList Again");
    console.log(oLanguageList)
});

oService.getThemeList().then(function (oThemeList) {
    console.log("Testing getThemeList Again");
    console.log(oThemeList);
});
```

**Listing 4.3** Testing Returned Promise Object for UserInfo Service

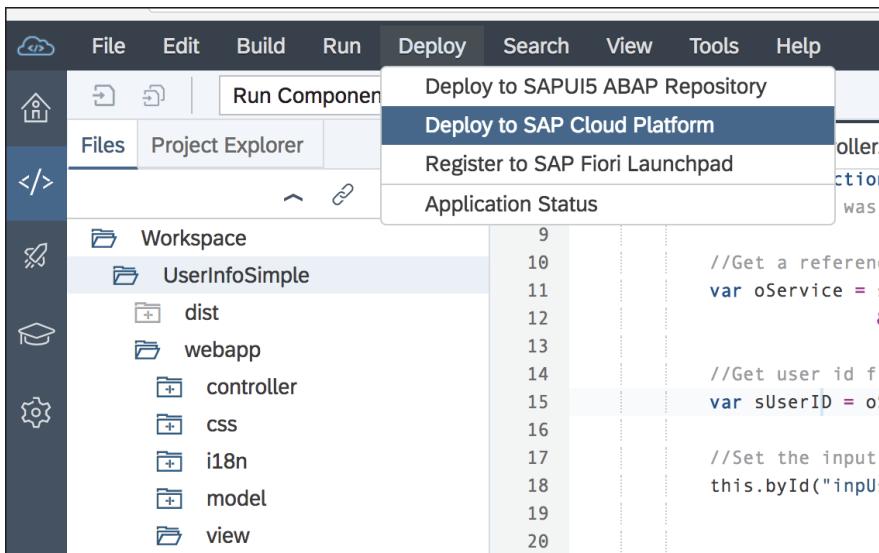
- Save, refresh, and test again; you'll see the new result in the console.



**Figure 4.30** Language List and Theme List Dumped in Console in Sandbox Environment

As shown in [Figure 4.30](#), the `getLanguageList` method returns an array with text and key fields, and the `getThemeList` method returns an object. This object has an `options` attribute, which is also an array with `id` and `name` fields. You can see that the sandbox supports various languages for testing but only one theme.

12. If you want to test it in SAP Cloud Platform Portal, you need to update the app in SAP Cloud Platform using the **Deploy • Deploy to SAP Cloud Platform** menu option, as shown in [Figure 4.31](#).



**Figure 4.31** Deploy to SAP Cloud Platform

13. This time, you don't need to create a new application. Just select the update option and click **Deploy**, as shown in [Figure 4.32](#).
14. Close the pop-up shown in [Figure 4.33](#), then switch back to the page you tested in SAP Cloud Platform Portal and refresh.
15. This time, you'll see that SAP Cloud Platform Portal only supports the browser's default language by default but supports various themes.

Deploy Application to SAP Cloud Platform

The selected application is already deployed to SAP Cloud Platform. Create a new version and deploy the updated application. A build will be triggered for the project. The build results will be deployed.

Application Details

Deploy a new application  Update an existing application

\*Account: p2000524802trial

Project Name: UserInfoSimple

\*Application Name: userinfosimple

Version Management

\*Version: 1.0.1  Activate

Application Status

State: STARTED

URL: [https://userinfosimple-p2000524802trial.dispatcher.hanatrial.ondemand...](https://userinfosimple-p2000524802trial.dispatcher.hanatrial.ondemand.com)

Versions

|                          | Version | Active |
|--------------------------|---------|--------|
| <input type="checkbox"/> | 1.0.0   | ✓      |

Figure 4.32 Update SAPUI5 Application in SAP Cloud Platform

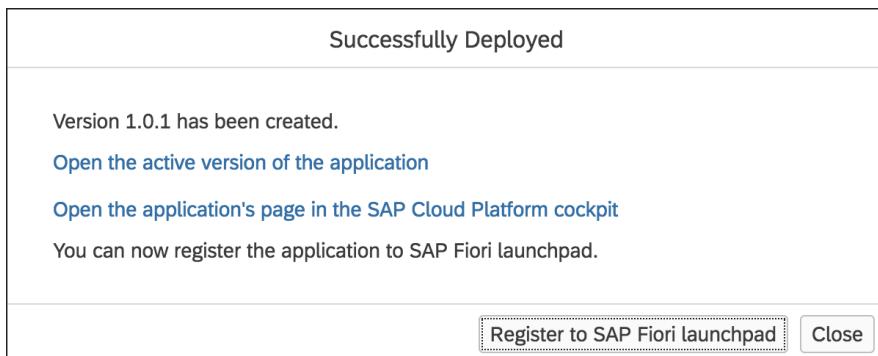


Figure 4.33 Successfully Updated SAPUI5 Application Pop-up

Now you know how to research a service if it isn't well documented. This applies to any SAPUI5 API, but keep in mind that the documentation is always the first thing you should check. Test APIs you find in this way carefully before use.

Table 4.1 summarizes key information about the user info service.

| Method                | Description                          | Returns   |
|-----------------------|--------------------------------------|---|
| getId                 | Get user's Id                        | A string  |
| getUser               | Get user object                      | An object containing methods for retrieving and changing user's preferences   |
| getLanguageList       | Get supported languages              | A promise that finally returns an array with text and key fields  |
| getThemeList          | Get supported themes                 | A promise that finally returns an object with an options attribute, which in turn is an array with id and name fields |
| updateUserPreferences | Apply a change against a user object | N/A   |

**Table 4.1** Methods of User Info Service

### 4.1.3 Creating and Testing a Complex SAPUI5 Application

In this section, you'll create a more complex SAPUI5 app, one that uses all the methods of the user info service. You'll then test the service to ensure that everything is working properly.

#### Creating an SAPUI5 Application

To begin, follow these steps:

1. Create a new SAPUI5 application from the template as before. This time, enter “UserInfoComplex” for the **Project Name** and “flpdev” for the **Namespace**, as shown in Figure 4.34. Perform all other steps as detailed in Section 4.1.1.

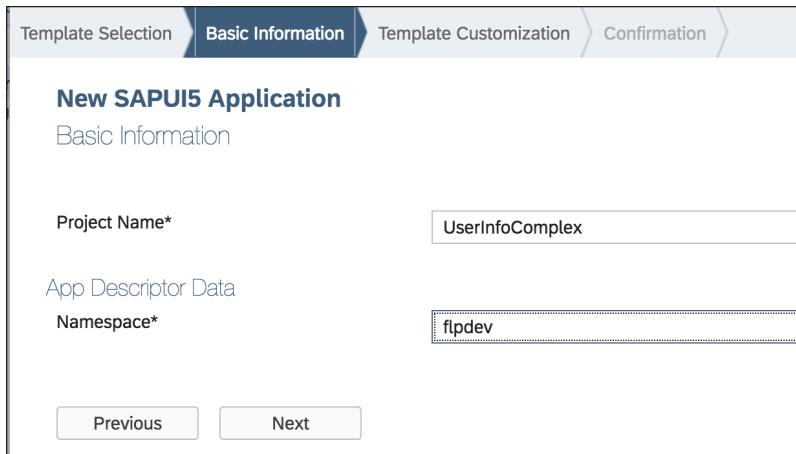


Figure 4.34 Create Project for Complex User Info Example

2. Double-click **manifest.json** under **UserInfoComplex • webapp**, then switch to the **Models** tab and click the plus sign +, as shown in [Figure 4.35](#).

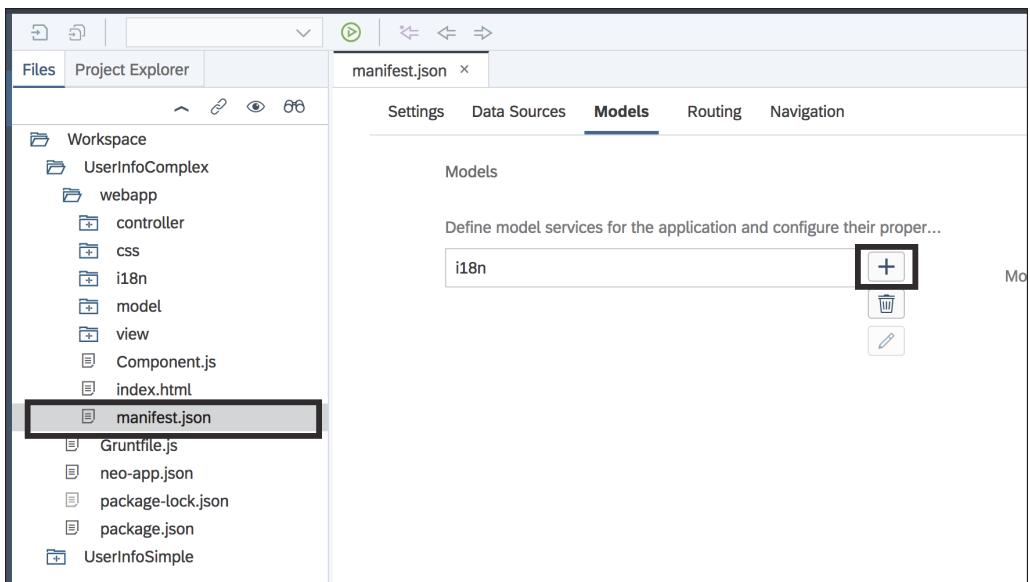
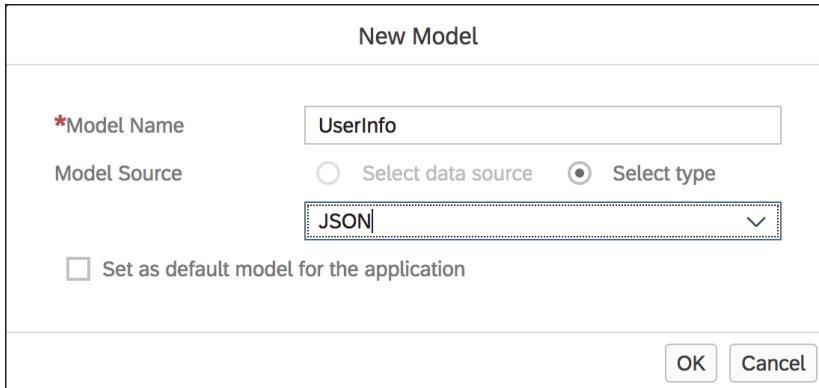


Figure 4.35 Model Tab in manifest.json

**Note**

If your screen displays lots of code instead of the descriptor editor, click **Descriptor Editor** at the bottom of the editor area.

3. In the **New Model** pop-up window, set **Model Name** to “UserInfo”, select the **Select Type** radio button, and set the type to **JSON** in the select control and click **OK**, as shown in [Figure 4.36](#).



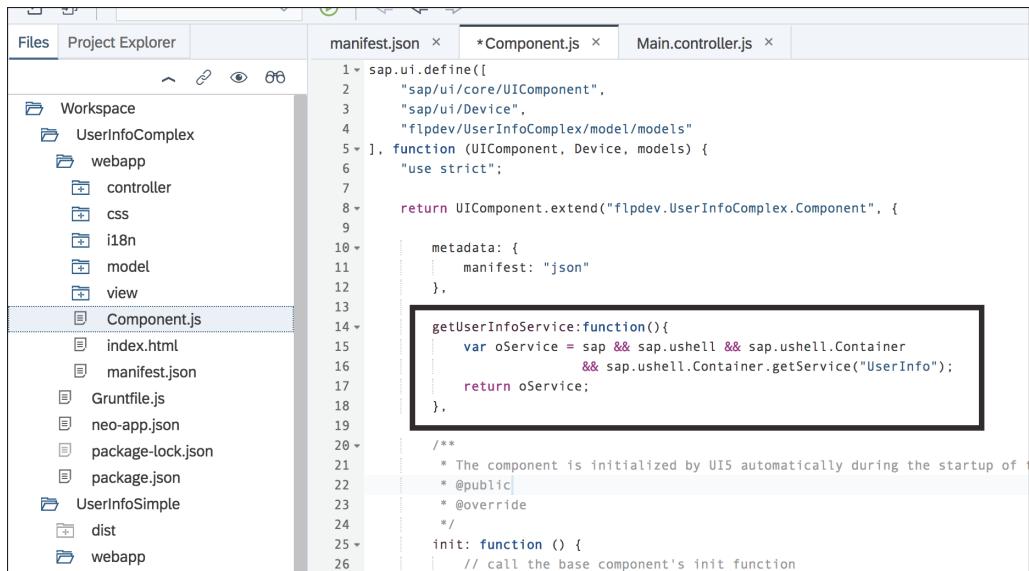
**Figure 4.36** New JSON Model

4. Open **Component.js** under **UserInfoComplex • webapp** and add a method called **getUserInfoService** before the **init** method, using the code shown in [Listing 4.4](#). You should add this code in the spot shown in [Figure 4.37](#).

```
//It is better to centralize the access to services in component
getUserInfoService:function(){
    var oService = sap && sap.ushell && sap.ushell.Container
        && sap.ushell.Container.getService("UserInfo");
    return oService;
},
```

**Listing 4.4** UserInfoComplex: Get Service in Component

5. Edit **Main.view.xml** with the layout editor by right-clicking it and selecting **Open With • Layout Editor** from the context menu, as shown in [Figure 4.38](#).



The screenshot shows the SAP Studio interface. The Project Explorer on the left lists several projects and files under 'Workspace'. The 'Component.js' file is selected in the list. The code editor on the right displays the content of Component.js. A specific section of the code is highlighted with a black rectangle:

```

1  sap.ui.define([
2      "sap/ui/core/UIComponent",
3      "sap/ui/Device",
4      "flpdev/UserInfoComplex/model/models"
5  ], function (UIComponent, Device, models) {
6      "use strict";
7
8      return UIComponent.extend("flpdev.UserInfoComplex.Component", {
9
10         metadata: {
11             manifest: "json"
12         },
13
14         getUserInfoService:function(){
15             var oService = sap && sap.ushell && sap.ushell.Container
16                 && sap.ushell.Container.getService("UserInfo");
17             return oService;
18         },
19
20         /**
21          * The component is initialized by UI5 automatically during the startup of
22          * @public
23          * @override
24         */
25         init: function () {
26             // call the base component's init function

```

Figure 4.37 Where to Write getUserInfoService Method

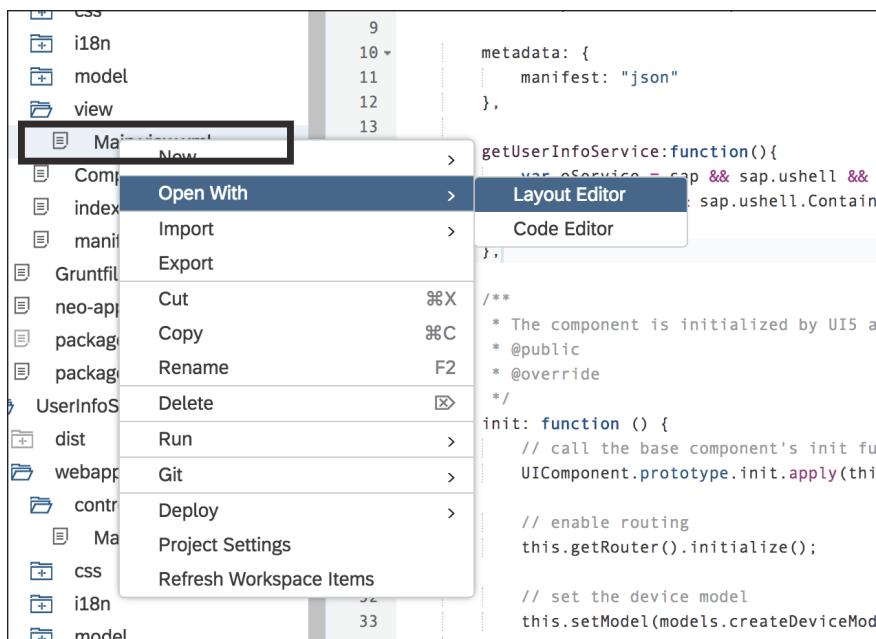
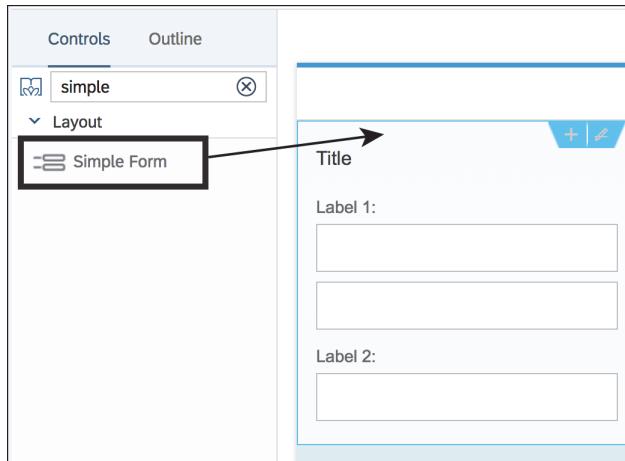


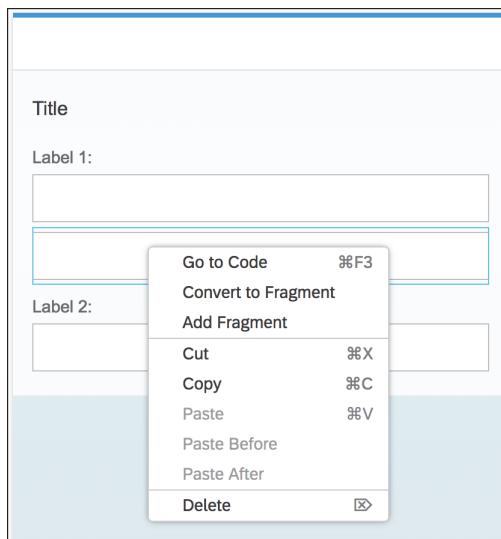
Figure 4.38 Open Layout Editor for Main View

6. Search for “simple” and drag a **Simple Form** control to the view, as shown in [Figure 4.39](#).



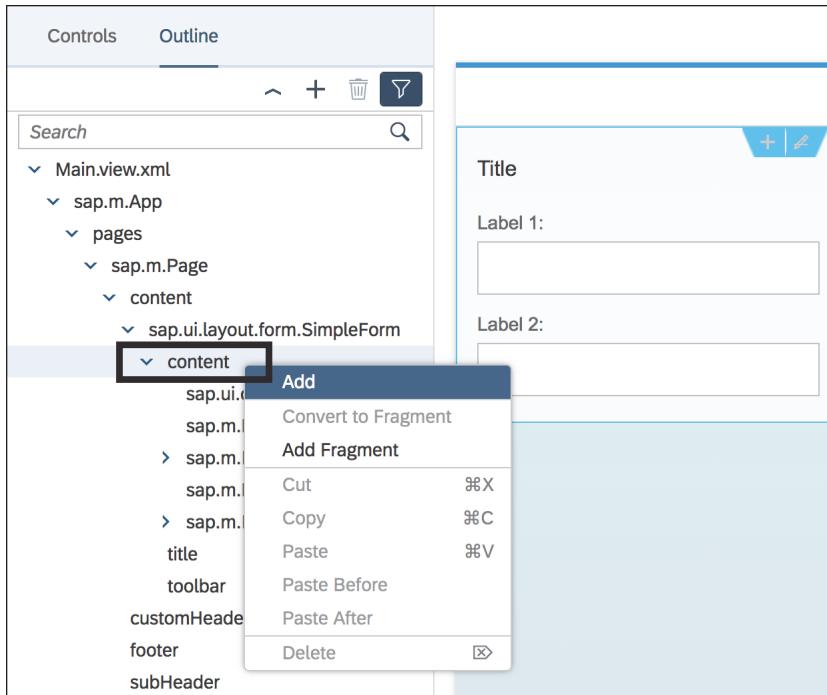
**Figure 4.39** Drag Simple Form

7. Delete the second input control by right-clicking it and choosing **Delete**, as shown in [Figure 4.40](#).



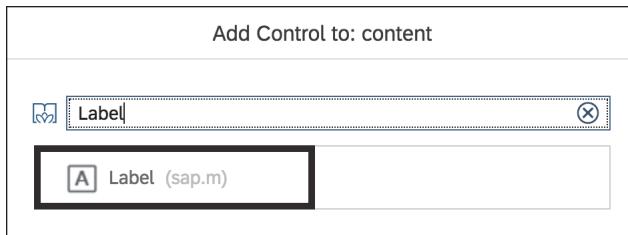
**Figure 4.40** Delete Second Input Control

8. Because the simple form control is truly complex, switch to the **Outline** tab to continue adjusting it.
9. Expand the outline structure until you see the **content** aggregation under the **SimpleForm** control. Right-click the content aggregation and click **Add**, as shown in [Figure 4.41](#).



**Figure 4.41** Add Elements to Content Aggregation of Simple Form

10. Search for “Label” and click the **Label** control in the result list, shown in [Figure 4.42](#).



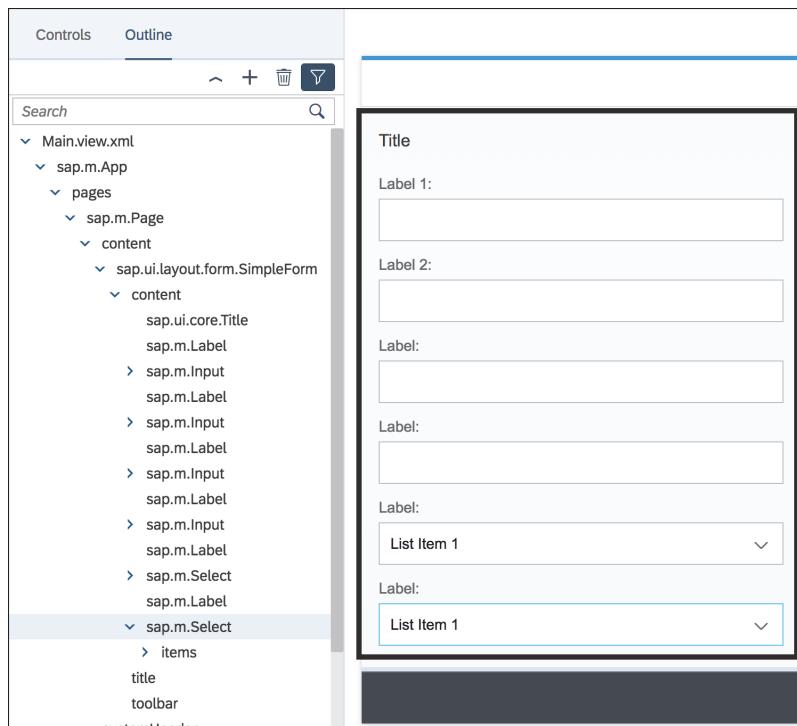
**Figure 4.42** Add Label to Content of Simple Form

11. Continue to add controls to the content aggregation according to Table 4.2.

| Sequence | Control Type |
|----------|--------------|
| 1        | Input        |
| 2        | Label        |
| 3        | Input        |
| 4        | Label        |
| 5        | Select       |
| 6        | Label        |
| 7        | Select       |

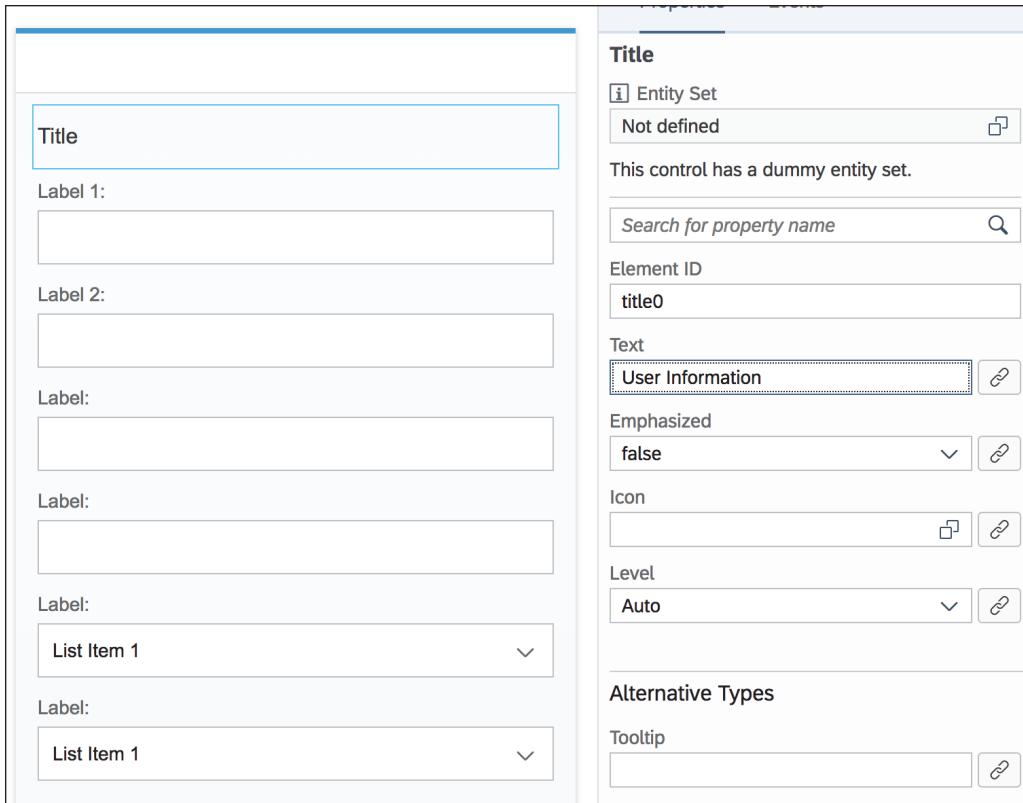
**Table 4.2** Controls for Main View of UserInfoComplex

12. The result should look like Figure 4.43.



**Figure 4.43** Results of Adding Controls to Simple Form

13. Change the **Title** property of the **Title** control to “User Information”, as shown in [Figure 4.44](#).



**Figure 4.44** Change Property for Title

14. Change the **Text** properties for all label controls for the six fields: make them read **User ID**, **First Name**, **Last Name**, **E-Mail**, **Language**, and **Theme**. The result should look like [Figure 4.45](#).
15. Switch back to the **Controls** tab, then search for and drag a **Toolbar** control to the footer part of the page, as shown in [Figure 4.46](#).

User Information

User ID:

First Name:

Last Name:

E-Mail:

Language:

Theme:

Figure 4.45 Text for Labels

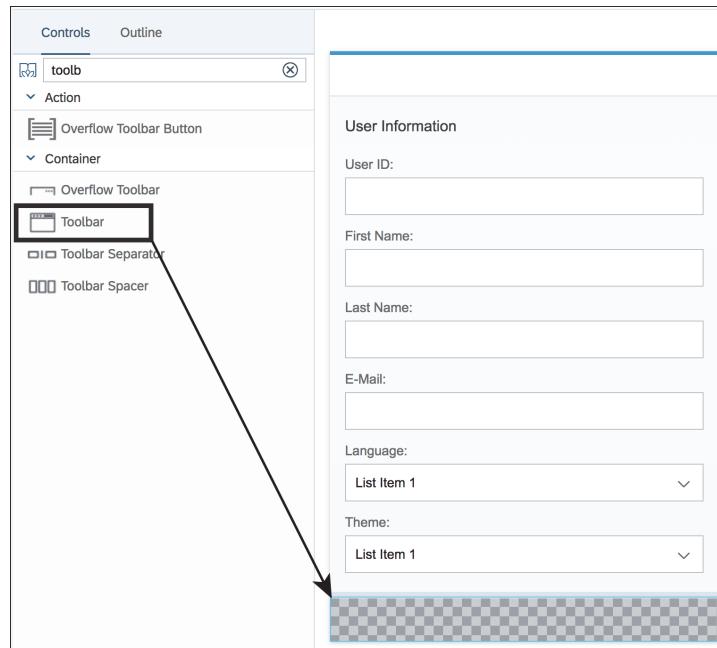
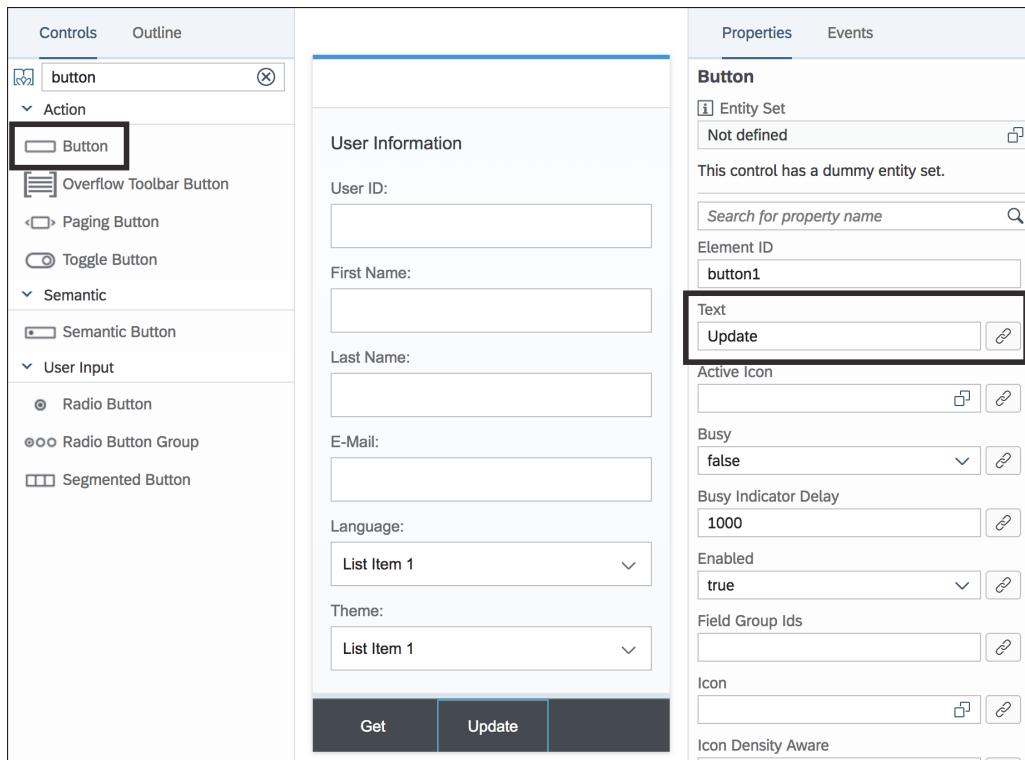


Figure 4.46 Add Toolbar

16. Add two buttons—with **Text** set to “Get” and “Update”—to the toolbar, as shown in [Figure 4.47](#).



**Figure 4.47** Add Buttons to Toolbar

17. Set the **Value** property of the input field for **User ID** to “{UserInfo>/Id}”, as shown in [Figure 4.48](#). This will bind the value to the JSON model we created earlier in this section. You will add data to this model later in this example.
18. Set the binding paths for all other input controls according to the values shown in [Table 4.3](#).

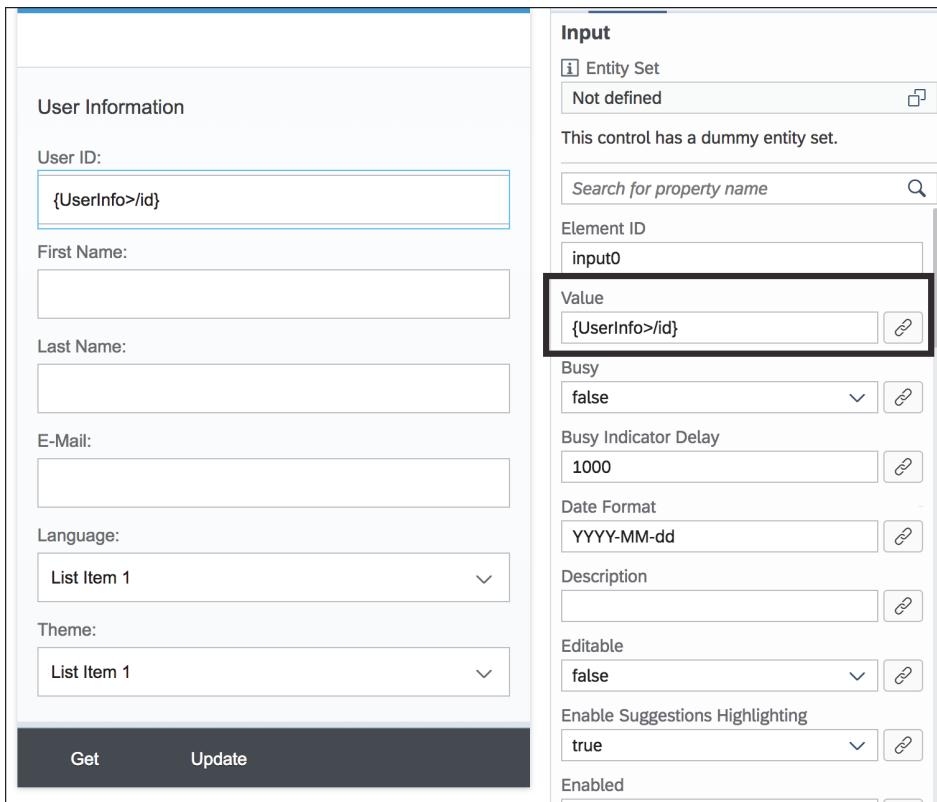


Figure 4.48 Set Binding Path for User ID

| Control Type | For Field  | Property Name | Binding Path          |
|--------------|------------|---------------|-----------------------|
| Input        | First Name | value         | {UserInfo>/firstName} |
| Input        | Last Name  | value         | {UserInfo>/lastName}  |
| Input        | E-Mail     | value         | {UserInfo>/email}     |
| Select       | Language   | selectedKey   | {UserInfo>/language}  |
| Select       | Theme      | selectedKey   | {UserInfo>/theme}     |

Table 4.3 UserInfoComplex: Data Binding Info

19. The result in the layout editor should look like Figure 4.49.

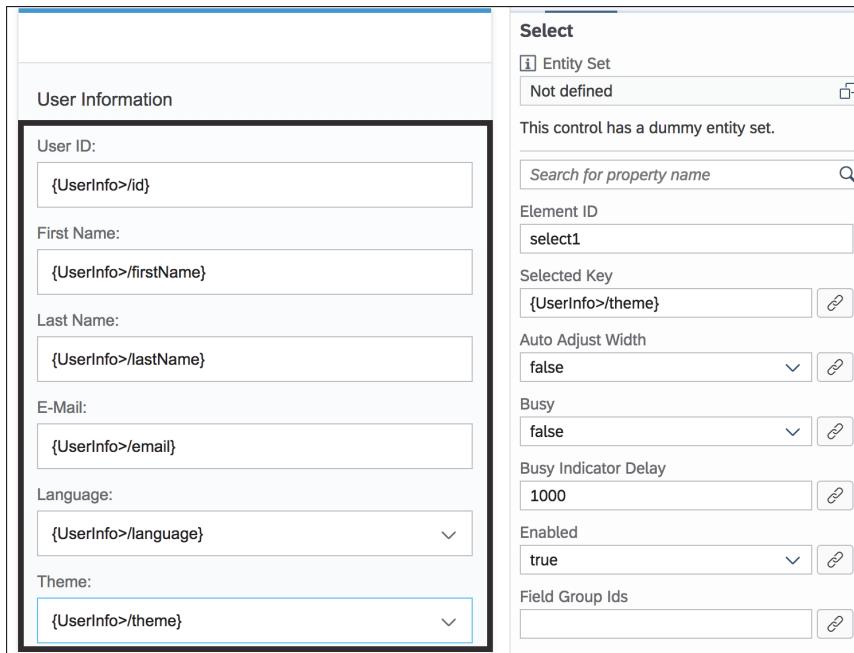


Figure 4.49 Setting Binding Paths for All Input Fields

20. Switch to the **Outline** tab, expand the **Items** aggregation for the select control, and delete all items in it using the context menu shown in [Figure 4.50](#). Do this for both select controls.

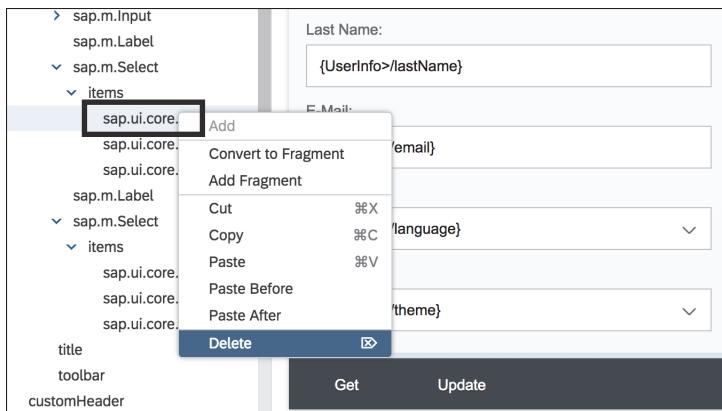
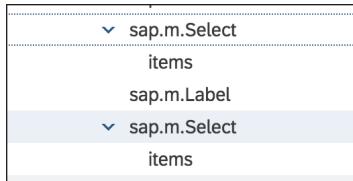


Figure 4.50 Delete All Items for Select Controls

21. All elements in the items will gone, as shown in [Figure 4.51](#).



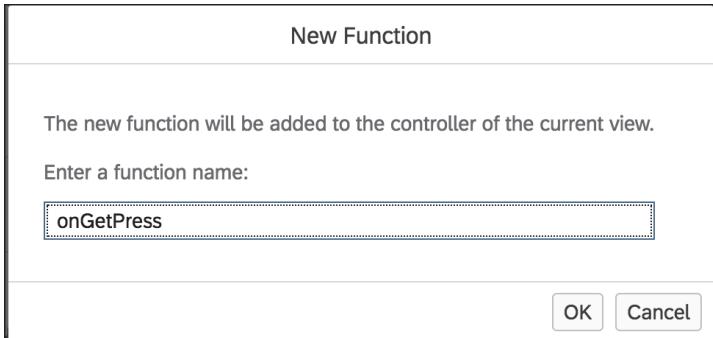
**Figure 4.51** Outline after Deleting All Items for Select Controls

22. Change the **Element Id** properties of the select controls to “selLanguage” and “selTheme”.
23. Select the **Get** button, switch to the **Events** tab, and create an event handler for **Press** by clicking **:** and selecting **New Function**, as shown in [Figure 4.52](#).

|  |  | Properties  | Events                            |
|--|--|---|-----------------------------------|
|  |  | <b>Button</b>   |                                   |
|  |  | <input type="text" value="Search for event name"/> <span>🔍</span> |                                   |
|  |  | Press   | <span>⋮</span>                    |
|  |  | Tap   | <span>⋮</span><br>Navigate To     |
|  |  | Validate Field Group  | <span>⋮</span><br>Select Function |
|  |  |   | <span>⋮</span><br>New Function    |
|  |  |   | <span>⋮</span><br>Open in Editor  |
|  |  | Model Context Change  |                                   |
|  |  | <span>⋮</span>  |                                   |
|  |  | Format Error  |                                   |
|  |  | <span>⋮</span>  |                                   |
|  |  | Parse Error   |                                   |
|  |  | <span>⋮</span>  |                                   |
|  |  | Validation Error  |                                   |
|  |  | <span>⋮</span>  |                                   |
|  |  | Validation Success  |                                   |
|  |  | <span>⋮</span>  |                                   |

**Figure 4.52** Create Event Handler for Get Button

24. Enter “onGetPress” and click **OK**, as shown in [Figure 4.53](#).



**Figure 4.53** Input Function Name

25. Edit the function by clicking **:** and selecting **Open in Editor**, as shown in [Figure 4.54](#).



**Figure 4.54** Jump to Editor for onGetPress Method

26. Write the code in [Listing 4.5](#) in the onGetPress method.

```
//Get service reference from the component
var oService = this.getOwnerComponent().getUserInfoService();
```

```
//
```

Check the availability of the service, if it is null, do nothing.

```
if (!oService) {
    //You can add your own fallback code here
    return;
}
```

```
//Get the reference of the JSON model you've created before
var oModel = this.getOwnerComponent().getModel("UserInfo");
//Get the user object
var oUser = oService.getUser();
//Create a json object with values get from the user object
var oData = {
    id: oUser.getId(),
    firstName: oUser.getFirstName(),
    lastName: oUser.getLastName(),
    email: oUser.getEmail(),
    language: oUser.getLanguage(),
    theme: oUser.getTheme()
};
//
Fill the model with the data, the screen fields will filled with those data
since they have already binded
oModel.setData(oData);

//Get the supported language list using promise
oService.getLanguageList().then(function (languageList) {
    //Get the reference of Select control for language
    var oSelLanguage = this.byId("selLanguage");
    //Loop the language list
    $.each(languageList, function (i, language) {
        //Create selection item for each language supported
        oSelLanguage.addItem(new sap.ui.core.Item({
            key: language.key,
            text: language.text
        }));
    });
    //
    Don't forget to bind the closure to "this", it is needed to access controls
    in the view in the closure
    }.bind(this));

//
Similiar to language, mention that the array of themes is property 'option'
of the resolved object
oService.getThemeList().then(function (themeList) {
```

```

var oSelTheme = this.byId("selTheme");
$.each(themeList.options, function (i, theme) {
    oSelTheme.addItem(new sap.ui.core.Item({
        key: theme.id,
        text: theme.name
    }));
});
}.bind(this));

```

**Listing 4.5** UserInfoComplex: Code for Pressed Button

27. Add a handler for the press event of the **Update** button, called `onUpdatePress`. Then edit the source code as shown in [Listing 4.6](#), as you did previously for the **Get** button.

```

var oService = this.getOwnerComponent().getUserInfoService();
if (!oService) {
    return;
}
var oModel = this.getOwnerComponent().getModel("UserInfo");
var oUser = oService.getUser();

//Change theme according to user selection
oUser.setTheme(oModel.getProperty("/theme"));
//Change language according to user selection
oUser.setLanguage(oModel.getProperty("/language"));

//Apply changes
oService.updateUserPreferences();

```

**Listing 4.6** UserInfoComplex: Code for Updated Button Pressed

28. Save all fields by clicking .

### Testing an SAPUI5 Application

To test your application, follow these steps:

1. View your application in the SAP Fiori launchpad sandbox by clicking **Run • Run as • SAP Fiori Launchpad Sandbox**.
2. On the test page, click **Get**, change the **Language** to any other one you want, then click **Update**, as shown in [Figure 4.55](#).

Title

User Information

User ID:

First Name:

Last Name:

E-Mail:

Language:

Theme:

The screenshot shows a user information form with fields for User ID, First Name, Last Name, E-Mail, Language, and Theme. The Language field has a dropdown menu open, listing various language codes. The 'en-US' option is selected. Other visible options include 'Browser Language', 'en-UK', 'en', 'de-DE', 'he-IL', 'ru-RU', and 'ru'. At the bottom, there are 'Get' and 'Update' buttons.

Figure 4.55 Test Result, Complex User Info Demo in Sandbox

3. Click the person icon in the top-left corner, then click **Settings**, as shown in [Figure 4.56](#).

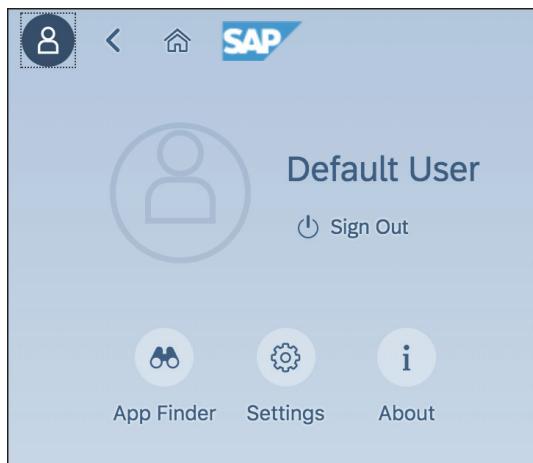
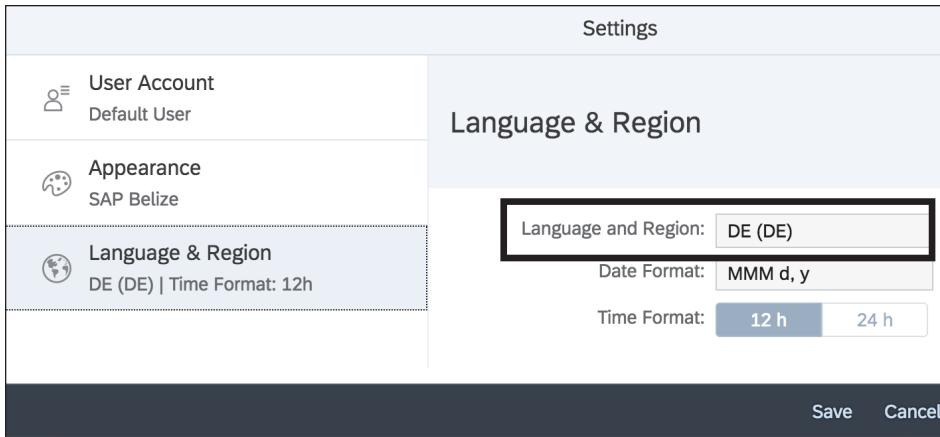


Figure 4.56 Check User Settings

4. On the next screen, you'll see that the language has been changed according to your selection, as shown in [Figure 4.57](#).



**Figure 4.57** Check Changed Language Setting

5. Deploy your new application to SAP Cloud Platform Portal as you've done before. The only difference now is that the **Title** should be “UserInfo” and the **Subtitle** should be “complex”, as shown in [Figure 4.58](#).

The screenshot shows the "Tile Configuration" step of the deployment process. The title is "Register to SAP Fiori Launchpad" and the sub-section is "Tile Configuration". There are four configuration fields: "Type" (Static), "Title" (UserInfo), "Subtitle" (complex), and "Icon" (sap-icon://approvals). To the right, there's a preview window showing a tile with the title "UserInfo" and subtitle "complex" (with a checked checkbox). At the bottom are "Previous" and "Next" buttons.

**Figure 4.58** Deploy Complex Example to SAP Cloud Platform Portal

6. Test the application again. This time, change the **Theme** to **SAP Belize Plus**, as shown in Figure 4.59.

The screenshot shows the SAP Cloud Platform Portal interface. At the top, there's a blue header bar with the SAP logo and the application title "UserInfoComplex". Below the header is a light gray content area. On the left, there's a vertical sidebar with icons for user profile, back, forward, and home. The main content area has a title "User Information". It contains several input fields: "User ID" (p2000524802), "First Name" (steve), "Last Name" (guo), "E-Mail" (flpdev@outlook.com), and dropdown menus for "Language" (Browser Language) and "Theme". The "Theme" dropdown is open, showing options: "SAP Belize" (which is selected), "SAP High Contrast Black", "SAP Belize", and "SAP Belize Plus". At the bottom, there are two buttons: "Get" and "Update".

**Figure 4.59** Test Result in SAP Cloud Platform Portal

7. After the theme is changed, it will appear as shown in Figure 4.60.

This screenshot is identical to Figure 4.59, showing the SAP Cloud Platform Portal with the "UserInfoComplex" application. The theme is set to "SAP Belize Plus", which is reflected in the dark blue header and sidebar colors. The content area, input fields, and dropdown menu options are the same as in Figure 4.59.

**Figure 4.60** Theme Changed to SAP Belize Plus

## 4.2 Bookmark Service

The bookmark service helps you create, update, delete, and count tiles. This service is used when a user wants to save the state of an application. The state must be directly accessible via a URL.

### Tip

SAPUI5 applications that apply best practices use a concept called *intent-based navigation*, as discussed in [Chapter 3](#). That means every major operation—for example, navigation among views or selecting a record—will change some part of the URL. To achieve this, a router needs to be used; the router class will help you change the URL when the navigation occurs. It will also help you to load specific pages with corresponding parameters according to the URL.

For the bookmark service, we'll also start with a simple example, then we'll discuss the other functions and options of this service. Because the previous section covered the user info service in detail, some of the steps that are similar here will be described much more concisely as you already know how to do them.

### 4.2.1 Creating an SAPUI5 Application

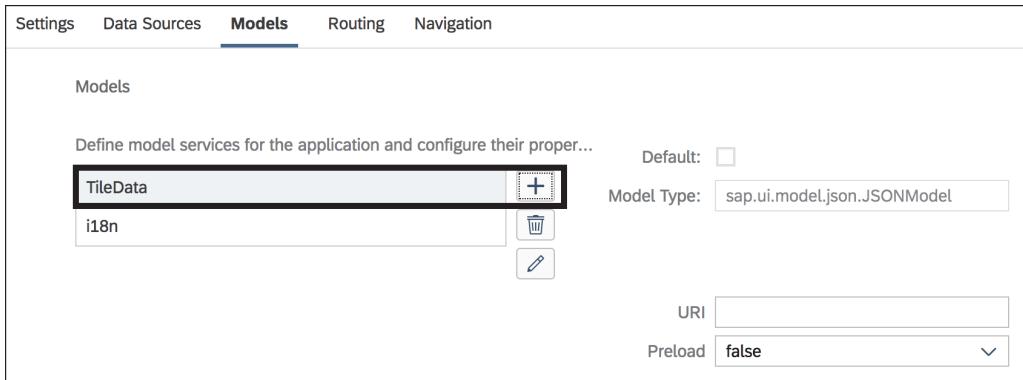
Now let's create a simple SAPUI5 application. The application contains a form and lets you input properties of a tile; it also has a button. When you click the button, a new tile will be created on the home page according to your input. To begin, follow these steps:

1. Open your SAP Web IDE full-stack version and create an SAPUI5 project, using the information in [Table 4.4](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | BookmarkSimple     |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

**Table 4.4** BookmarkSimple: Project Properties

2. Edit *manifest.json* in the descriptor editor, and add a JSON model called “TileData” in the **Models** tab, as shown in Figure 4.61.



**Figure 4.61** Result of Models Tab in manifest.json

3. Edit *view/Main.view.xml* in the layout editor. Add a simple form, and fill it in with controls according to Table 4.5. Change the title to “Bookmark Testing”. The result should look like Figure 4.62.

| Sequence | Control Type | Note              |
|----------|--------------|-------------------|
| 1        | Label        | Text:Title        |
| 2        | Input        |                   |
| 3        | Label        | Text:URL          |
| 4        | Input        |                   |
| 5        | Label        | Text:Subtitle     |
| 6        | Input        |                   |
| 7        | Label        | Title:Service URL |
| 8        | Input        |                   |
| 9        | Label        | Title:Icon        |
| 10       | Input        |                   |
| 11       | Icon         |                   |

**Table 4.5** BookmarkSimple: Controls and Properties in UI

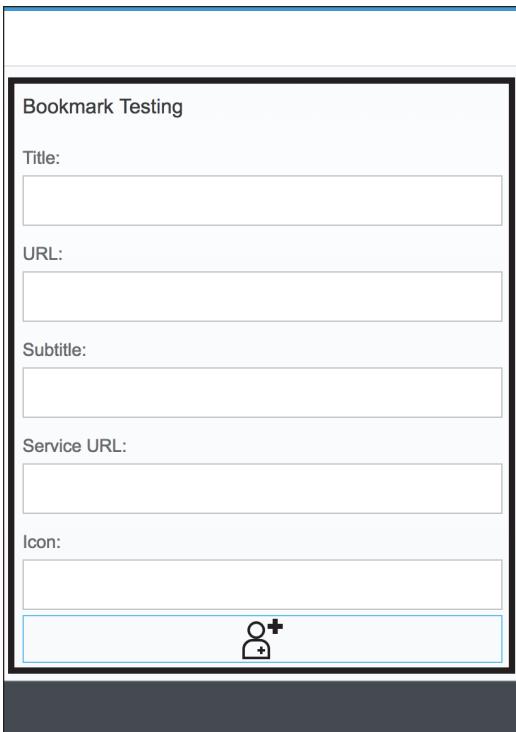


Figure 4.62 Result of Form for Bookmark Testing App

4. Set data binding according to [Table 4.6](#). The last input control will bind the same fields with the icon control. This is for previewing the icon after entering its URL. The result should look like [Figure 4.63](#).

| Control               | Property | Binding                |
|-----------------------|----------|------------------------|
| Input for Title       | value    | {TileData>/title}      |
| Input for URL         | value    | {TileData>/url}        |
| Input for Subtitle    | value    | {TileData>/subtitle}   |
| Input for Service URL | value    | {TileData>/serviceUrl} |
| Input for Icon        | value    | {TileData>/icon}       |
| Icon control          | src      | {TileData>/icon}       |

Table 4.6 Bookmark Simple: Data Binding Info

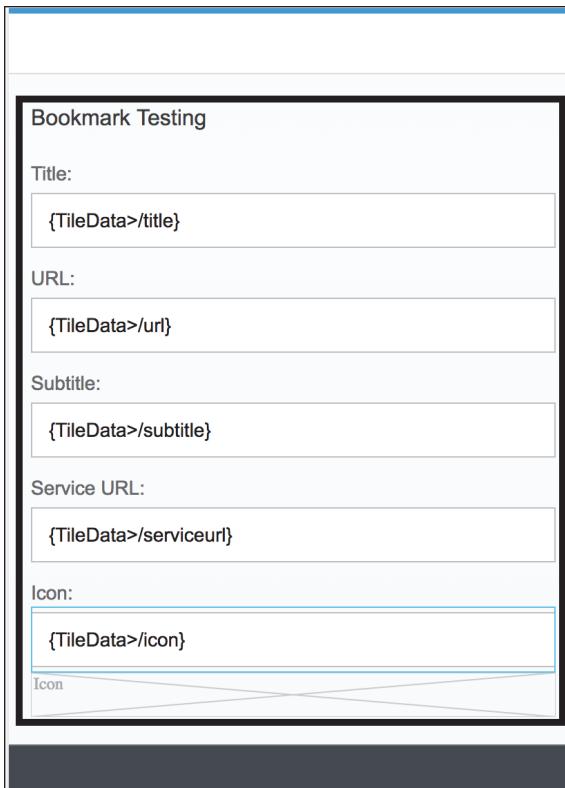
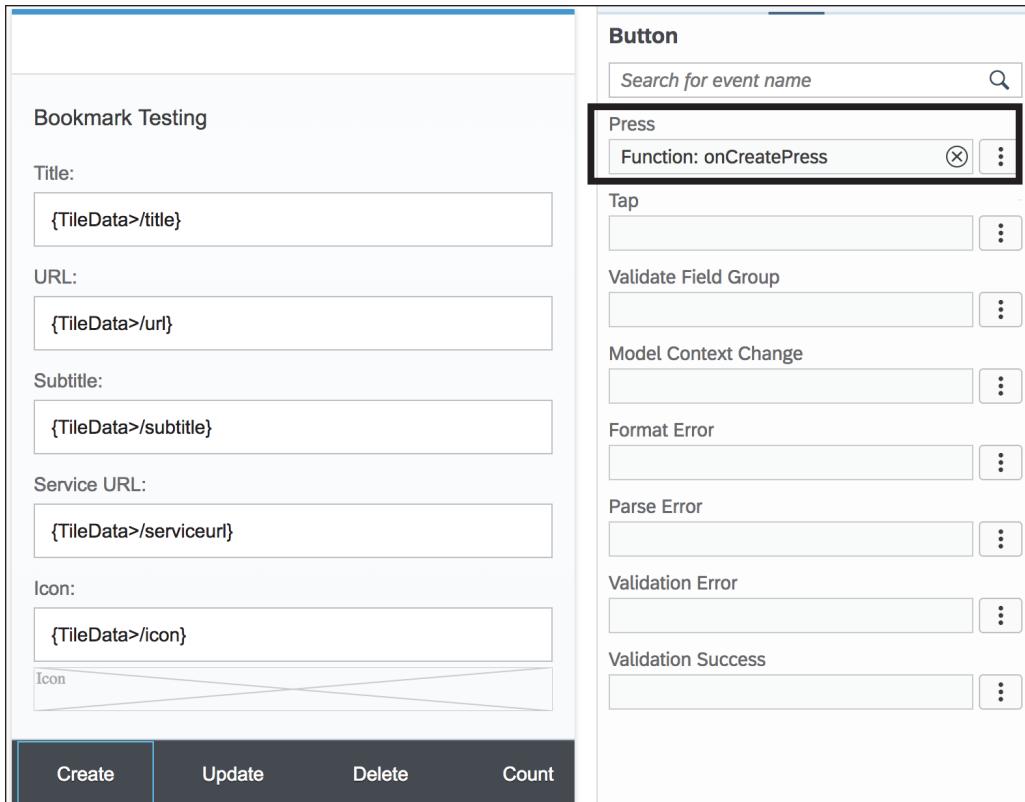


Figure 4.63 Results of Data Binding of Form for Bookmark Testing App

5. Add a toolbar to the footer, then add four buttons to the toolbar (see [Figure 4.64](#)). The buttons are **Create**, **Update**, **Delete** and **Count**; change the text for each accordingly. In the **Event** panel, create an event handler for the press event of each button: `onCreatePress`, `onUpdatePress`, `onDeletePress`, and `onCountPress`.
6. Edit `Component.js` to create a new method called `getBookmarkService`, just as you did for the user info app you created in [Section 4.1.3](#), using the code in [Listing 4.7](#).

```
getBookmarkService:function(){
    var oService = sap && sap.ushell && sap.ushell.Container
        && sap.ushell.Container.getService("Bookmark");
    return oService;
},
```

**Listing 4.7** BookMarkSimple: Get Service in Component.js



**Figure 4.64** Add Toolbar, Button, and Event Handler to Bookmark Testing App

7. Edit the `onCreatePress` method in `controller/Main.controller.js` using the code in Listing 4.8. Get user input from the JSON model, then call the `addBookmark` method of the service to add a tile. You also need to process the promise type return object.

```
onCreatePress: function (oEvent) {
    //This code was generated by the layout editor.
    //Get a reference of the service
    var oService = this.getOwnerComponent().getBookmarkService();
    if (!oService) {
        return;
    }
    //Get user input data from JSON model
    var oData =
        this.getOwnerComponent().getModel("TileData").getData();
```

```
//Call the service, add tile
oService.addBookmark({
    title: oData.title,//Title
    url: oData.url,//The link will be navigated to. Use http(s)://xxxx for external link. Use #
<SemanticObject>-<Action>/... for others aps in Fiori Launchpad
    icon: oData.icon,//The icon
    info: null,//Long text for the tile, will display on the bottom-right corner of the tile
    subtitle: oData.subtitle,//subtitle, display under title
    serviceUrl: oData.serviceUrl,//A url point to an odata service call which return specic format to display
    a number
    serviceRefreshInterval: 0,//Auto refresh interval, 0 for manually refresh
    numberUnit: null
}).then(function () {
    //Success
    jQuery.sap.require("sap.m.MessageBox");
    sap.m.MessageBox.success("The tile has been created success
fully!");
}).fail(function (message) {
    //Failed
    jQuery.sap.require("sap.m.MessageBox");
    sap.m.MessageBox.error(message);
});
},
```

**Listing 4.8** BookmarkSimple: Create Tile

8. Edit the `onUpdatePress` method in `controller/Main.controller.js` using the code in [Listing 4.9](#). Get user input from the JSON model, then call the `updateBookmarks` method of the service to change a tile. You also need to process the promise type return object.

---

#### Note

The bookmark service identifies a bookmark tile by its URL. Therefore, when updating a bookmark, all tiles with the same URL will be updated. It isn't possible to change the URL for a specific tile; a new URL needs a new tile.

```

onUpdatePress: function (oEvent) {
    //This code was generated by the layout editor.
    //Get service reference
    var oService = this.getOwnerComponent().getBookmarkService();
    if (!oService) {
        return;
    }
    //Get data
    var oData =
        this.getOwnerComponent().getModel("TileData").getData();
        //The update method require 2 parameters
        //
        The first parameter is the URL, use this to identify existing tiles
        //
        The second parameter is a JSON object just same as createBookmarks.
        //
        Since we have same peroperty names for JSON model and the paramter, this ti
        me we use the JSON directly
        oService.updateBookmarks(oData.url,oData).then(function (number
    ) {
        //
        If it is successed, the number of affected tiles will be passed through the
        callback
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.success(number + " tiles have been updated
        successfully");
        }).fail(function (message) {
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.error(message);
    });

},

```

**Listing 4.9** BookmarkSimple: Update Tile

9. Edit the `onDeletePress` method in `controller/Main.controller.js` using the code in [Listing 4.10](#). Get user input from the JSON model, then call the `deleteBookmarks` method of the service to delete tiles. You also need to process the promise type return object.

```
onDeletePress: function (oEvent) {
    //This code was generated by the layout editor.
    //Get service info
    var oService = this.getOwnerComponent().getBookmarkService();
    if (!oService) {
        return;
    }
    //Get data
    var oData = this.getOwnerComponent().getModel("TileData").getData();
    //The delete method only requires url
    oService.deleteBookmarks(oData.url).then(function (number) {
        //If it is succeeded, the number of affected tiles will be passed
        through the callback
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.success(number + " tiles have been deleted success
        fully");
    }).fail(function (message) {
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.error(message);
    });
},
},
```

**Listing 4.10** BookmarkSimple: Delete Tile

10. Edit the `onCountPress` method in `controller/Main.controller.js` using the code in [Listing 4.11](#). Get user input from the JSON model, then call the `countBookmarks` method of the service to delete tiles. You also need to process the promise type return object.

```
onCountPress: function (oEvent) {
    //This code was generated by the layout editor.
    var oService = this.getOwnerComponent().getBookmarkService();
    if (!oService) {
        return;
    }

    var oData = this.getOwnerComponent().getModel("TileData").getData();
    //The count method only requires url
    oService.countBookmarks(oData.url).then(function (number) {
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.success(number + " tiles exist for current url");
```

```

        }).fail(function (message) {
            jQuery.sap.require("sap.m.MessageBox");
            sap.m.MessageBox.error(message);
        });

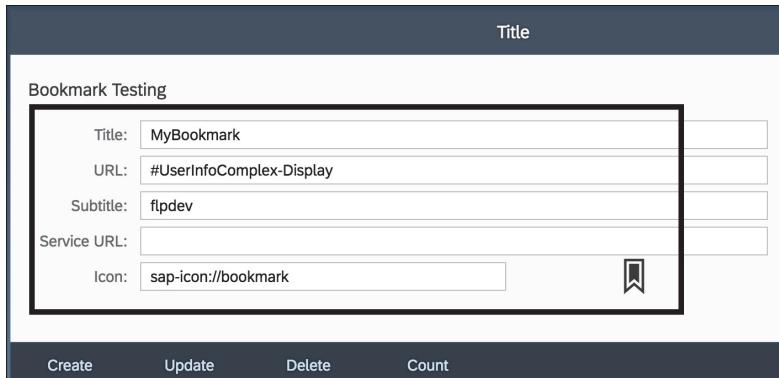
    }

```

**Listing 4.11** BookmarkSimple: Perform Tile Count

11. Save your code changes by clicking .
12. Deploy the app to SAP Cloud Platform by following the menu path **Deploy** • **Deploy to SAP Cloud Platform**.
13. After deployment, register the app to SAP Fiori launchpad. Use “Bookmark” as the tile **Title** and “simple” as the **Subtitle**.
14. Preview your app in SAP Fiori launchpad. Fill in the form with the sample data shown in Table 4.7 and click **Create**, as shown in Figure 4.65.

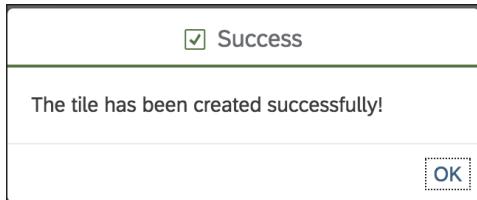
| Field    | Value to Enter             |
|----------|----------------------------|
| Title    | “My Bookmark”              |
| URL      | “#UserInfoComplex-Display” |
| Subtitle | “flpdev”                   |
| Icon     | “sap-icon://bookmark”      |

**Table 4.7** BookmarkSimple: Test Data for Tile


| Title  |  |
|--|--|
| <b>Bookmark Testing</b>  |  |
| Title:   | <input type="text" value="MyBookmark"/>  |
| URL:   | <input type="text" value="#UserInfoComplex-Display"/>  |
| Subtitle:  | <input type="text" value="flpdev"/>  |
| Service URL:   | <input type="text"/>   |
| Icon:  | <input type="text" value="sap-icon://bookmark"/>  |
| <a href="#">Create</a> <a href="#">Update</a> <a href="#">Delete</a> <a href="#">Count</a> |  |

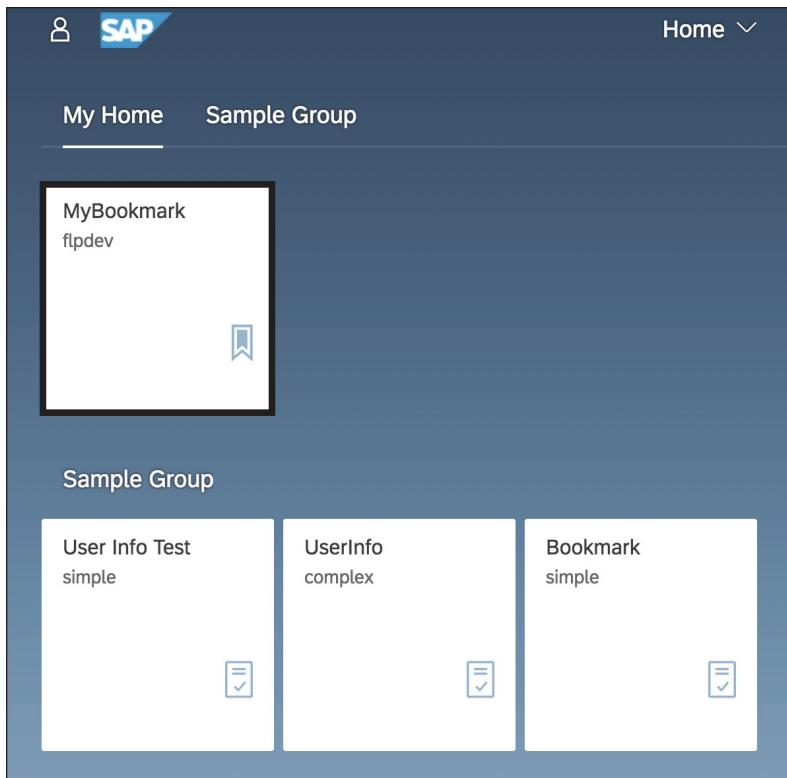
**Figure 4.65** Testing Bookmark App

15. The **Success** dialog shown in Figure 4.66 will appear.



**Figure 4.66** Success Message for Creating Tile

16. Click to go to the launch page, where you'll see a new tile created in the **My Home** group, as shown in Figure 4.67.



**Figure 4.67** Bookmark Tile Added

17. You can also test updating, deleting, and counting in the same way.

## 4.2.2 Using Additional Functions

By using the AddBookmarkButton control, you can simplify the process of adding a bookmark. You just need to use the control and set a property for it.

This control is a specific button just for adding bookmarks. You can't use it to update, delete, or count bookmarks—but normally users can do those things by editing the homepage. Therefore, using this control is the best practice when you only want to add bookmarks.

To use this control, you must edit the XML view manually, following these steps:

1. Import namespace sap.ushell.ui.footerbar into the view, as shown in [Figure 4.68](#).

```
<mvc:View xmlns:html="http://www.w3.org/1999/xhtml" xmlns:mvc="sap.ui.core.mvc" xmlns="sap.m"
    xmlns:footerbar="sap.ushell.ui.footerbar"
    controllerName="flpdev.BookmarkSimple.controller.Main" displayBlock="true">
```

**Figure 4.68** Import Namespace for AddBookmarkButton

2. Use it in the XML view. The only required parameter is customURL; you can fill it via data binding or by setting via a callback function using the setBeforePressHandler method, as shown in [Figure 4.69](#).

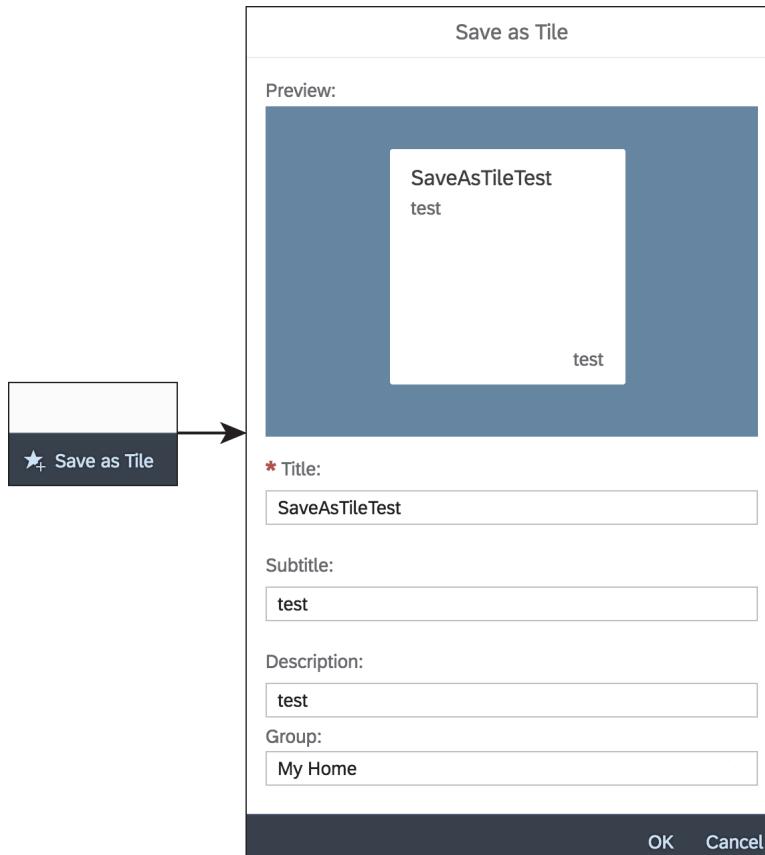
### Warning

If you haven't provided a URL, the current URL will be used by default.

```
<Toolbar class="sapContrast sapContrastPlus" width="100%" id="toolbar1">
    <content>
        <!--<Button text="Create" width="100px" id="button0" press="onCreatePress"/><!--&gt;
        &lt;footerbar:AddBookmarkButton customUrl="{TileData>/url}" />
        <Button text="Update" width="100px" id="button1" press="onUpdatePress"/>
        <Button text="Delete" width="100px" id="button2" press="onDeletePress"/>
        <Button text="Count" width="100px" id="button3" press="onCountPress"/>
    </content>
</Toolbar>
```

**Figure 4.69** Usage of AddBookmarkButton in XML View

3. The button has a default look and feel. When a user clicks it, a pop-up window will let that user provide information, as shown in [Figure 4.70](#).



**Figure 4.70** Runtime Behavior of AddBookmarkButton Control

At this point, we've explained the most useful functions of the bookmark service, though the bookmark service can even do more powerful things. Because you won't often use these powerful functions in a normal SAPUI5 app (if a user has requirements of that sort, he can just edit the homepage himself!), we'll just list them briefly here:

- **Bookmark existing tiles**

With the `addCatalogToGroup` method, you can add a tile that already exists in a user's catalog to a specific group. You can use the `LaunchPage` client service to get a list of available catalogs and groups.

- **Adding a bookmark to any group**

The `addBookmark` method has a second parameter that's used to specify a tile group.

Using this parameter, you can add a bookmark to groups other than the home group. You can use the LaunchPage client service to get a list of available groups.

- **Adding dynamic tiles**

When you provide a `serviceUrl` attribute for a tile, it will be a dynamic tile, which is dynamically numbered. For this, you need to provide an OData function import with returning data of the type described in document.

For more information, access the documentation for SAP Fiori Launchpad on ABAP 7.52 via the path **Administration Guide • Setting up Launchpad Content • Setting Up Content With Launchpad Designer • Creating and Configuring Tiles • Configuring Tiles • Dynamic App Launcher Tiles • OData Structure for Dynamic App Launchers**.

## 4.3 Personalization Service

A personalization service can help you save the application state for a set period. The data that is persisted will not be lost when a user logs off and then logs on again. We'll again start with a simple example before moving on to discuss some specifics of data storage locations and how to handle complex data with a personalization service.

### 4.3.1 Creating an SAPUI5 Application

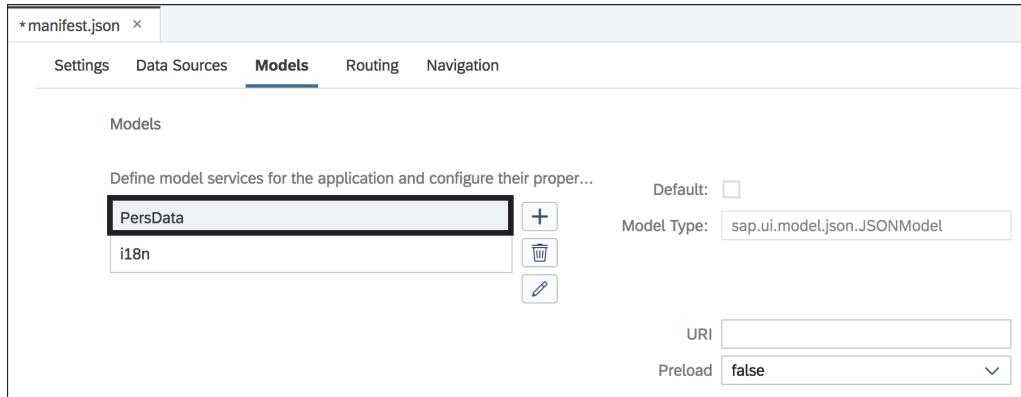
In this section, you'll create a simple SAPUI5 application that persists user input, as follows:

1. Open your SAP Web IDE full-stack version and create an SAPUI5 application according to [Table 4.8](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | PersDemo           |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

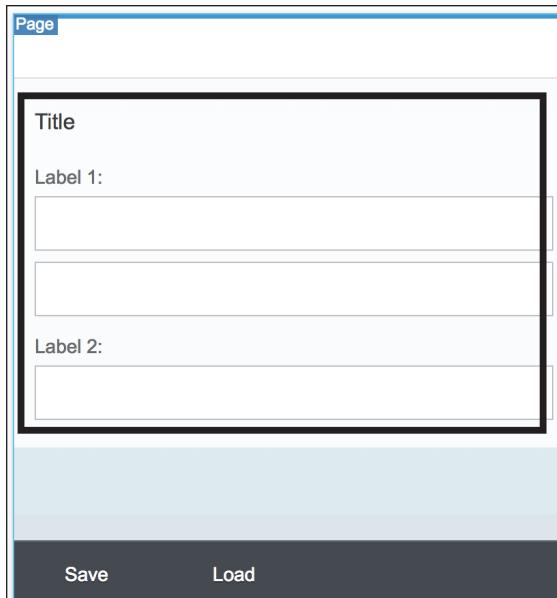
**Table 4.8** PersDemo: Project Properties

2. Open *manifest.json* and create a JSON model called *PersData* (see [Figure 4.71](#)).



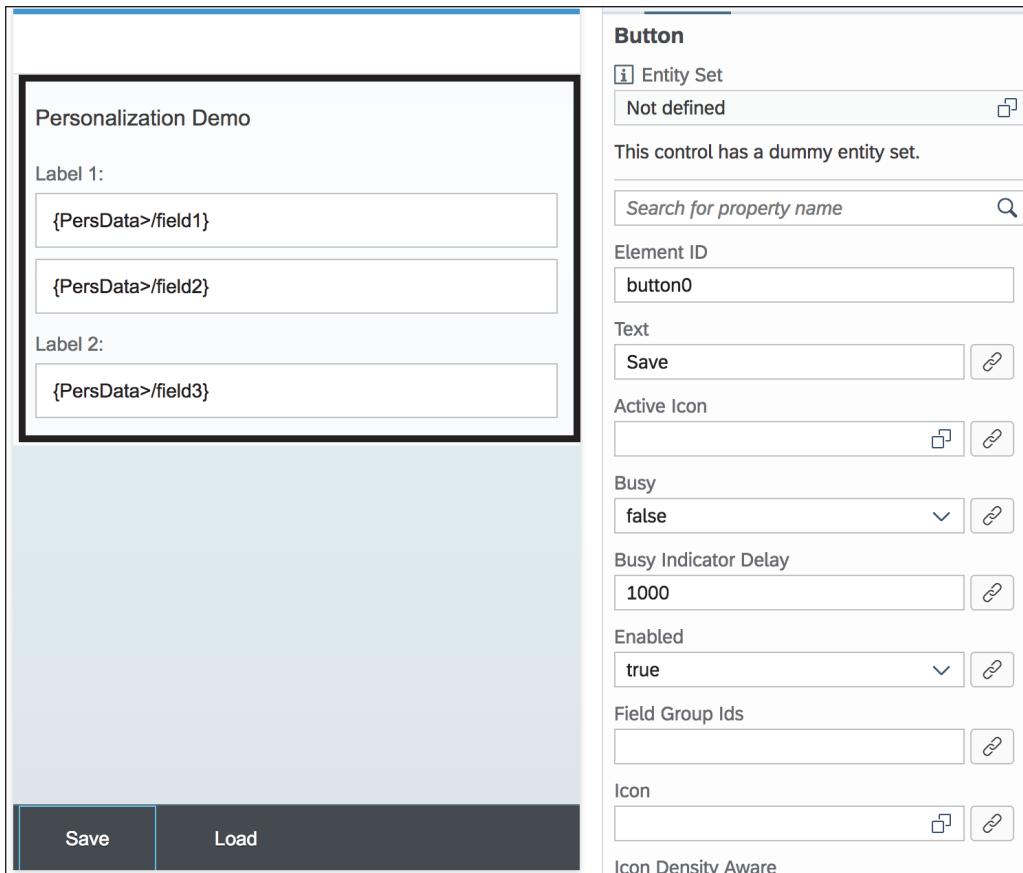
**Figure 4.71** JSON Model for PersDemo App

3. Open **view • Main.view.xml** using the layout editor. Drag a **SimpleForm** to the screen and keep it in its default state. Then drag a **Toolbar** and two **Buttons** into the footer area. Change the **Text** for those buttons to “Save” and “Load” (see [Figure 4.72](#)).



**Figure 4.72** Layout of Main View for PersDemo App

4. Bind the three input controls to `{PersData>/field1}`, `{PersData>/field2}`, and `{PersData>/field3}`. Add `onSave` and `onLoad` event handlers for the corresponding buttons, as shown in [Figure 4.73](#).



**Figure 4.73** Data Binding and Event Handlers for PersDemo App

5. Open `Component.js` and add the `initPersService` method using the code in [Listing 4.12](#).

```
initPersService: function () {
    //1.Get reference of the service
    var oService = sap && sap.ushell && sap.ushell.Container
    &&
    sap.ushell.Container.getService("Personalization");
```

```
        if (oService) {
//2.Define options for the service
            var oScope = {
                keyCategory: oService.constants.keyCategory.FIXED_KEY,
                writeFrequency: oService.constants.writeFrequency.LOW,
                clientStorageAllowed: true
            };
//3.Define an identifier for the data you want to save
//This make sure you will not retrive wrong data
            var oPersId = {
                container: "flpdev.PersDemo",//
Usually use namespace of your app
                item: "persData">//Give a meaningful name
            };
            //
Create a personalizer. Use this object to save and load data
            //The 3rd parameter should be the component itself
            this.oPersonalizer =
oService.getPersonalizer(oPersId, oScope, this);
        }
    },

```

**Listing 4.12** PersDemo: Initialize Personalization Service

6. Call the method in [Listing 4.13](#) in the `init()` method.

```
init: function () {
    ....
    //Call this init method
    this.initPersService();

}
```

**Listing 4.13** PersDemo: Initialize Component

7. Edit the `onSave` method of `Main.controller.js`. In this method, first get data from the JSON model, then save it with the personalization service, as shown in [Listing 4.14](#).

```
onSave: function (oEvent) {
    //This code was generated by the layout editor.
    //Get the JSON Model
    var oModel = this.getOwnerComponent().getModel("PersData");
```

```
var oData = oModel.getData();
//Save Data

var oPersonalizer = this.getOwnerComponent().oPersonalizer;
//Save data. return a promise
oPersonalizer.setPersData(oData)
.done(function(){
    //Deal with success
})
.fail(function(){
    //Log errors
});

},
```

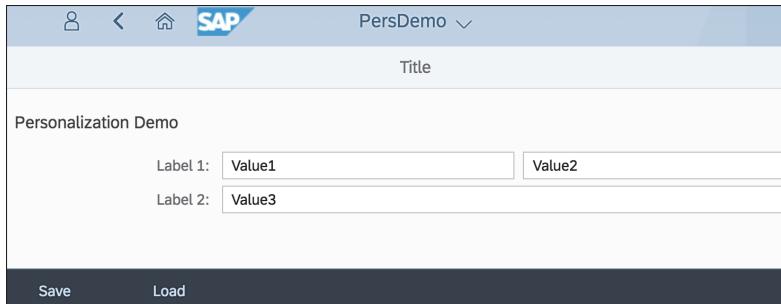
**Listing 4.14** PersDemo: Save Data Using Personalization Service

8. Edit the `onLoad` method to get data from the personalization service, as shown in [Listing 4.15](#). Note that you must deal with these values when the promise has been resolved.

```
onLoad: function (oEvent) {
    //This code was generated by the layout editor.
    var oPersonalizer = this.getOwnerComponent().oPersonalizer;
    //The getPersData method returns a promise
    //In the resolve method, it will pass the data saved before
    oPersonalizer.getPersData()
    .done(function (oPersData) {
        //Success. Process the data
        var oModel = this.getOwnerComponent().getModel("PersData");
        oModel.setData(oPersData);
    })
    .bind(this))
    .fail(function(){
        //Error. Log error
    })
}
```

**Listing 4.15** PersDemo: Fetch Data from Personalization Service

9. Save all the code and test it in the SAP Fiori launchpad sandbox by following menu path **Run • Run as SAP Fiori Launchpad Sandbox**.
10. The result should look like Figure 4.74.



**Figure 4.74** Resulting PersDemo App

11. Fill the fields with some data, then click **Save**.
12. Refresh your browser; all the data will disappear.
13. Click **Load**; your data will reappear.

### 4.3.2 Data Storage Locations

Now that you've tested this service, let's discuss some of its details. Data can be stored in many places, as discussed in Table 4.9.

| Data Location                                    | Pros   | Cons  |
|--|--|---|
| User's local disk                                | Fast; no roundtrip; valid even after closing the browser | Not secure; may be cleaned by user                                    |
| Browser's memory; handled by SAP Fiori launchpad | Fast; no roundtrip; secure                               | Will be lost if user closes browser                                   |
| Memory of SAP NetWeaver AS ABAP frontend server  | Fast; never loose  | Needs roundtrip; memory consumption; update needed if data is updated |
| SAP NetWeaver AS ABAP physical table             | Never loose  | Needs roundtrip; database access                                      |

**Table 4.9** Data Storage Options for Personalization Service

These data storage options are usually arranged so as to best suit the requirements for stability, performance, and security. Remember that when you create the personalizer, you need to provide an `oScope` parameter—and in `oScope`, you can control how to store data.

By default, the user's local disk isn't used for security reasons, but if you're saving noncritical data, then you can set `clientStorageAllow` to `true` in the scope object. Thereafter, the local disk will be used as a cache to improve performance.

You can also set the frequency of data changes via the `writeFrequency` property. The default value is `HIGH`, which means that the cache in ABAP service memory isn't used because then changes will lead to a refresh of the cache. If your data doesn't change very frequently, you can set `writeFrequency` to `LOW` to improve performance.

A `validity` property in `oScope` is used to set the duration for which the data is stored. You can set it to 0 or another number to represent how many minutes data should be stored. You can also set it to `infinity` to keep the data in the ABAP frontend server database forever.

If you set the validity to 0, no storage methods on the ABAP server will be used; data will be lost when the browser is closed.

#### SAP Fiori Launchpad Sandbox

Because you're in testing mode when using the sandbox, the data is stored on the local disk.

The data is stored in tables `/UI2/CONTAINER` and `/UI2/ITEM` in the ABAP frontend server; you can check these tables using Transaction SE16. And you can clean all of the tables using Transaction `/UI2/PERS_EXPIRED_DELETE`.

### 4.3.3 Handling Complex Data

When you want to handle complex data, you can use a container instead of the personalizer. The personalizer saves and loads all data at once; in contrast, a container can save lots of items, and you can save or load each item individually. It enables much more flexibility and is much powerful.

It can be used to store a list of variants or the complex settings of a table, but for now, it's not the best way to do so. Therefore, we encourage you not to use this service do anything too complex.

Now, let's change the PersDemo app to use container mode and see how it works. To begin, follow these instructions:

1. Change the `initPersService` method in `Component.js` to `initPersContainer`, and replace the implementation using the code in [Listing 4.16](#).

```
initPersContainer: function () {
    var oService = sap && sap.ushell && sap.ushell.Container &&
sap.ushell.Container.getService("Personalization");
    if (oService) {
        var oScope = {
            keyCategory: oService.constants.keyCategory.FIXED_KEY,
            writeFrequency: oService.constants.writeFrequency.LOW,
            clientStorageAllowed: true
        };
        // When you get a container, you cache all data to localside
        // This needs time, so a promise is required
        oService.getContainer("flpdev.PersDemo2", oScope, this)
            .fail(function () {
                jQuery.sap.log.error("Loading personalization data
failed.");
            })
            .done(function (oContainer) {
                this.persContainer = oContainer;

                }.bind(this));
    }
},
```

**Listing 4.16** PersDemo: Initialize Service Using Container Mode

2. Change the code in the `init` method to invoke this new method.
3. Change the `onSave` method in `Main.controller.js` using the code in [Listing 4.17](#).

```
onSave: function (oEvent) {
    //This code was generated by the layout editor.
    var oModel = this.getOwnerComponent().getModel("PersData");
    var oContainer = this.getOwnerComponent().persContainer;
    //You can set data 1 by 1
    oContainer.setItemValue("F1", oModel.getProperty("/field1"));
    oContainer.setItemValue("F2", oModel.getProperty("/field2"));
```

```

        oContainer.setItemValue("F3", oModel.getProperty("/field3"));
        //After those, invoke method save to save them in one batch
        oContainer.save()
            .fail(function () {
                //Log something
            })
            .done(function () {
                // Tell the user that the data is saved
            });
    },
}

```

**Listing 4.17** PersDemo: Save Data Using Container Mode

4. Change the load method in *Main.controller.js* using the code in [Listing 4.18](#).

```

onLoad: function (oEvent) {
    //This code was generated by the layout editor.
    var oModel = this.getOwnerComponent().getModel("PersData");
    var oContainer = this.getOwnerComponent().persContainer;
    //As the data is loaded when the container was initialized
    //You don't need a promise here
    oModel.setProperty("/field1",oContainer.getItemValue("F1"));
    oModel.setProperty("/field2",oContainer.getItemValue("F2"));
    oModel.setProperty("/field3",oContainer.getItemValue("F3"));

}

```

**Listing 4.18** PersDemo: Fetch Data Using Container Mode

### Restrict Personalization Service Usage

After learning about this option, you must have a lot of ideas about how to use this service. But remember: SAP has various solutions to save and load data. This service is a generic way to save temporary and noncritical data to improve user experience and simplify your development, but where a more specific method exists, you need to use it.

Here are some examples in which a personalization service could be used but isn't the best solution:

- To save an app state, such as which document or which screen is displayed, it's better to save it in a URL and then save it as a tile using a bookmark service.
- To save table settings, variants, or filter conditions, it's better to use SAPUI5 flexibility services (like smart table/smart filter bar/smart variant controls).
- To let a user customize her layout and save the layout info, use SAPUI5 flexibility services.
- You can use this service to save an uncompleted form, but currently SAP is developing a new draft service based on core data services (CDS), Business Object Processing Framework (BOPF), and SAP Fiori elements. Check whether that solution suits your needs first.

## 4.4 Summary

In this chapter, you learned how to write platform-independent code to interact with SAP Fiori launchpad via the client-side services it provides. All those services facilitate common usage and can improve user experience in a simple way. Visit <https://ui5.sap.com/#/api/sap.ushell.services> to investigate the full API for client services.

In next chapter, you'll learn about another set of APIs that can be used to extend the standard SAP Fiori launchpad UI.

# Chapter 5

## Extensibility

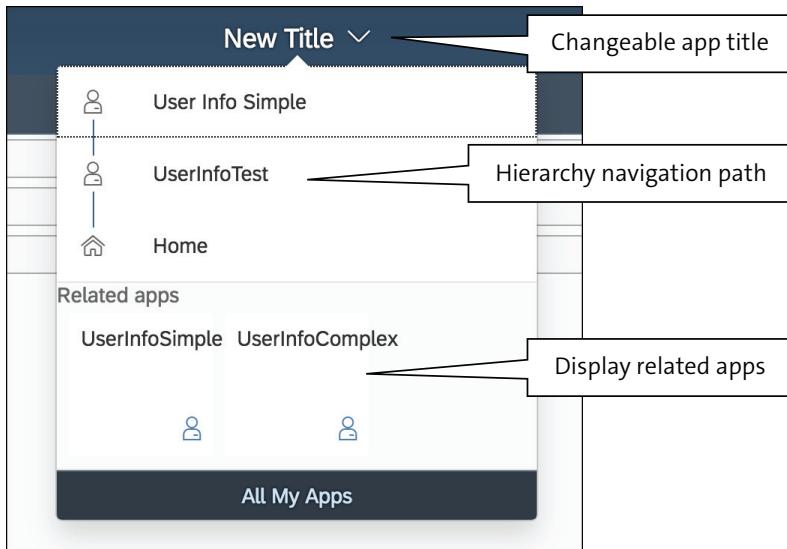
*With extensions, you can add new functions to SAP Fiori launchpad without changing its code. In this chapter, you'll see how powerful these extensions can be.*

SAP Fiori launchpad provides various ways to add additional functions to it. In this chapter, we'll go through all major extensibility options using a set of meaningful examples. Those examples will show you how to extend various areas of SAP Fiori launchpad. We'll start with the title and the pop-up that appears when you click the title, and then extend the header bar of the home page. Next, you'll learn how to add a subheader, footer, and left-hand toolbar to your home page. Finally, we'll focus on the Me Area. Before we get into the examples, however, let's start with an overview of where and how SAP Fiori launchpad can be extended.

### 5.1 Extension Options

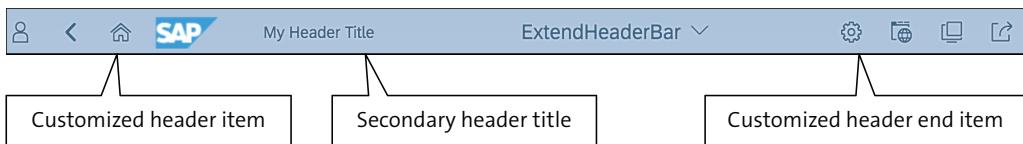
For every SAP Fiori application, the title of the app, which is stored in a resource bundle (*i18n\_##.properties*) in the `appTitle` key, will be displayed. That's because a configuration in `manifest.json` called `title` under the `sap.app` domain points to the key of the resource bundle.

The first challenge is to modify the title as needed (see [Figure 5.1](#)). Then you can add more content in the context menu of the title. You can add a list of related apps and can add a set of links in a hierarchy that can guide users to any level of an app.



**Figure 5.1** Extend Title and Its Context Menu

Next, look at the shell header (see [Figure 5.2](#)). If you don't want to change the app title, you can optionally set a secondary title to the left of the app title. You can also add buttons, which we call *header items* when they're at the beginning of the header and *header end items* when they're at the end.



**Figure 5.2** Extend Shell Header

You can even add more custom elements to the launch page (see [Figure 5.3](#)). On the launch page, you can add a subheader bar under the shell header, as well as a custom footer bar. You can also add toolset items to the left side, which will be hidden automatically when none of the toolset items can be used.

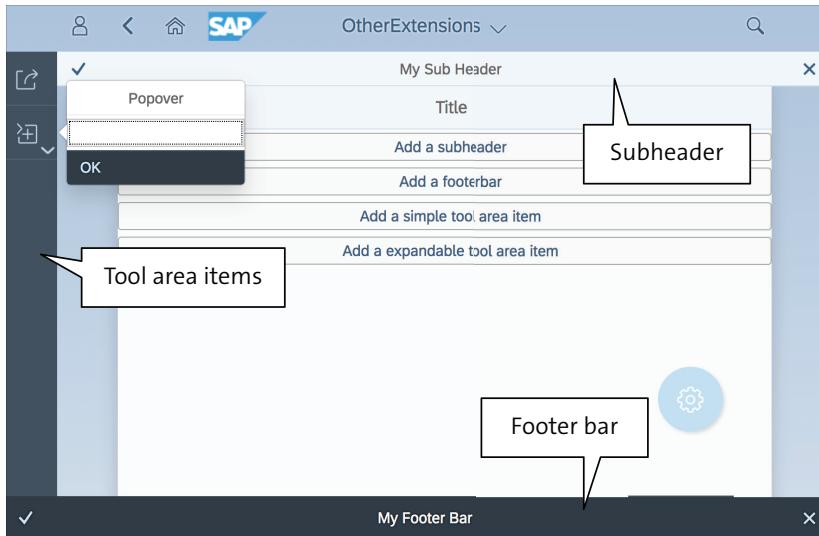


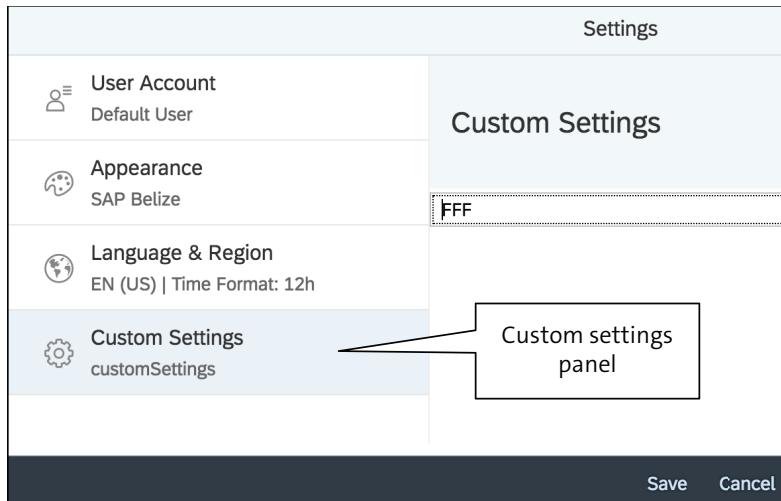
Figure 5.3 Other Elements Added to Launch Page

When you click the icon that looks like a person in the top-left corner, you'll switch to the Me Area, which contains important functions for the launchpad, as shown in [Figure 5.4](#).



Figure 5.4 Add Custom Action to Me Area

You can add both custom buttons and any setting options you need by clicking on the **Custom Settings** button (shown in [Figure 5.5](#)).



**Figure 5.5** Add Setting Options to Settings Pop-Up

## 5.2 App Title Information Extensions

Now, let's discuss how to customize an app's title. For all operations on the title and its context menu, you need help from `ShellUIService` under the `sap.ushell.ui5service` namespace. Invoking this service is different from client services you've used before. To use it, you need to declare a service alias in the app descriptor and call that alias when you need to. The service provides the methods listed in [Table 5.1](#) to help you handle app titles and their context menus.

| Method                      | Description                   |
|-----------------------------|-------------------------------|
| <code>setTitle</code>       | Change the app title          |
| <code>setRelatedApps</code> | Set the list of related apps  |
| <code>setHierarchy</code>   | Set a list of hierarchy nodes |
| <code>getTitle</code>       | Get the app's title           |

**Table 5.1** Important Methods of `ShellUIService`

This section will show you how to change title settings using `ShellUIService`. You'll first create a blank SAPUI5 project and then add three buttons as triggers for three

parts of your code. Then you'll write code to change the title of the app. Finally, you'll change the hierarchy and related apps content of the pop-up shown when clicking the title.

### 5.2.1 Preparing a Project

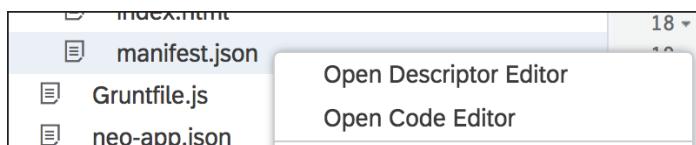
Create a simple project for this section via the following steps:

1. Enter SAP Web IDE full-stack and create an SAPUI5 application project, using the information in [Table 5.2](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | ChangeTitle        |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

**Table 5.2** ChangeTitle: Project Settings

2. Edit `webapp/manifest.json` using the code editor by choosing **Open Code Editor** in the context menu of the file, as shown in [Figure 5.6](#).



**Figure 5.6** Open manifest.json with Descriptor Editor

3. To manipulate the app title, you need to declare an alias for the service in the app descriptor, under the “sap.ui5” domain, as shown in [Figure 5.7](#).

#### Note

This method of managing services is much elegant than you've used previously. But unfortunately, for now, ShellUIService is the only service you can invoke in this way.

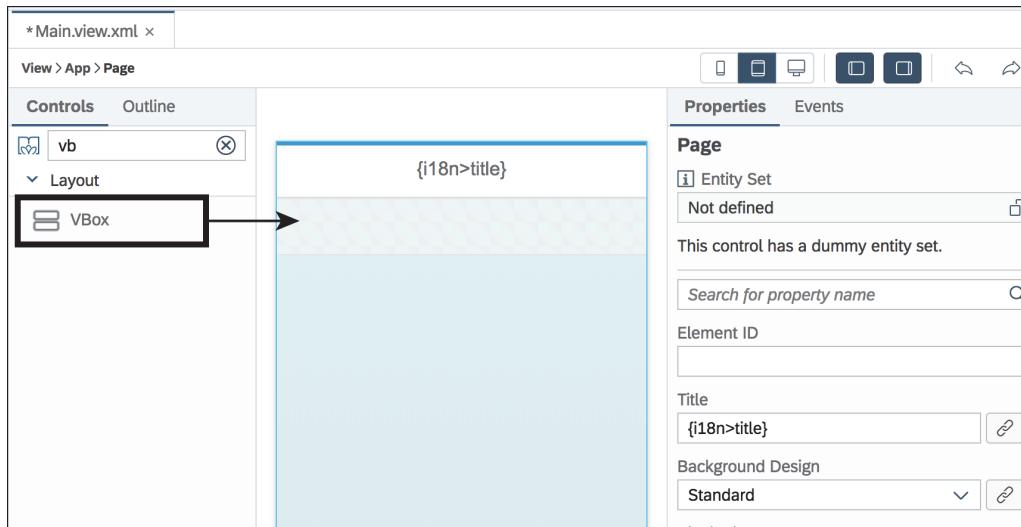
```

"sap.ui5": {
    "services": {
        "ShellUIService": {
            "lazy": false,
            "factoryName": "sap.ushell.ui5service.ShellUIService"
        }
    },
    " rootView": {
        "viewName": "flpdev.ChangeTitle.view.Main",
        "type": "XML"
    }
}

```

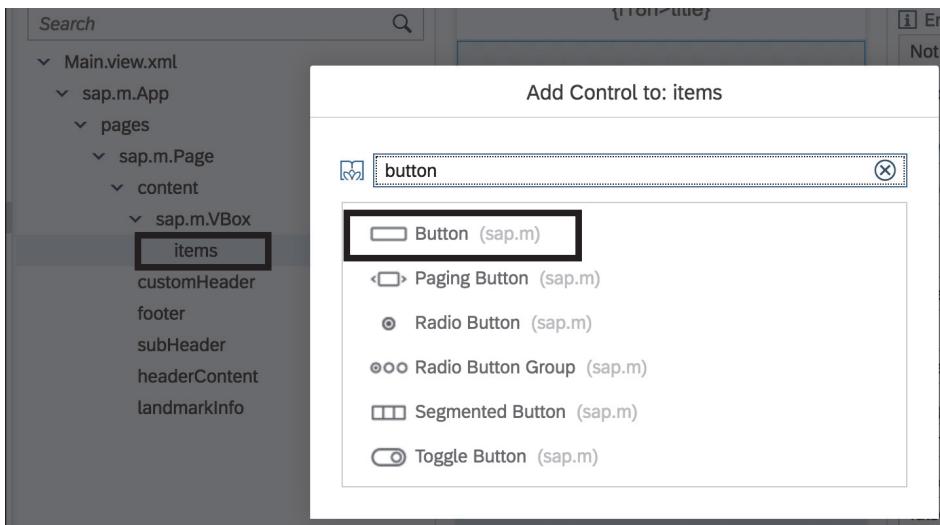
**Figure 5.7** Configure ShellUIService in manifest.json

4. Open *webapp/view/Main.view.xml* using the layout editor.
5. In the layout editor, search for “VBox” and drag it in to the content of the page control, as shown in [Figure 5.8](#).



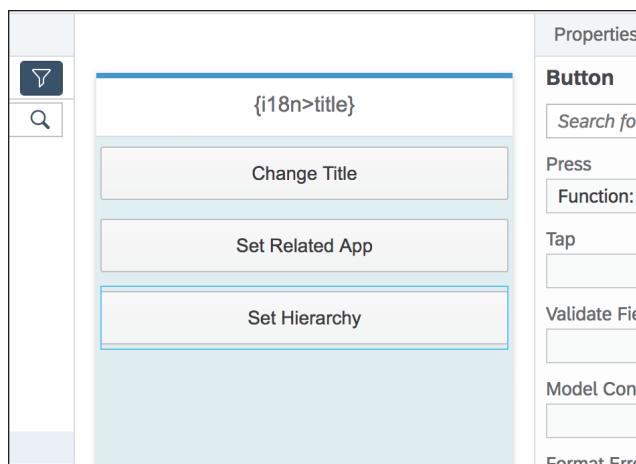
**Figure 5.8** Drag VBox onto Screen

6. Switch to the **Outline** tab, expand the tree, and locate the **Items** aggregation under the **VBox** control. Add a button control by right-clicking on **Items** and then clicking **Add**. Then search for “Button” and select it in the search results, as shown in [Figure 5.9](#).



**Figure 5.9** Add Button Control for VBox

7. Repeat this step several times until you have three buttons. Change the width of those buttons to 100% and change their texts to the following (see [Figure 5.10](#)):
  - “Change Title”
  - “Set Related App”
  - “Set Hierarchy”



**Figure 5.10** Result Screen for ChangeTitle Demo

8. Create event handler functions for the press event of each button, as you did in [Chapter 4](#), according to [Table 5.3](#).

| Button          | Event Handler Method |
|-----------------|----------------------|
| Change Title    | changeTitle          |
| Set Related App | setRelatedApp        |
| Set Hierarchy   | setHierarchy         |

**Table 5.3** Change Title: Event Handlers for Buttons

9. Save all your changes by clicking on .

### 5.2.2 Changing the App Title

To change the title, you need to get the reference of the service and use the `setTitle` method. To begin, follow these steps:

1. Since you've declared an alias for the service in the app descriptor, the service itself can be fetched using the `getService` method of the UI component, as follows:

```
oComponent.getService(<ServiceAliasName>)
```

2. The code returns a promise object, with which you can get the service once it's resolved. You can get reference of the service using the code in [Listing 5.1](#).

```
oComponent.getService(<ServiceAliasName>)
.done(function(oService){
  //use the service
});
```

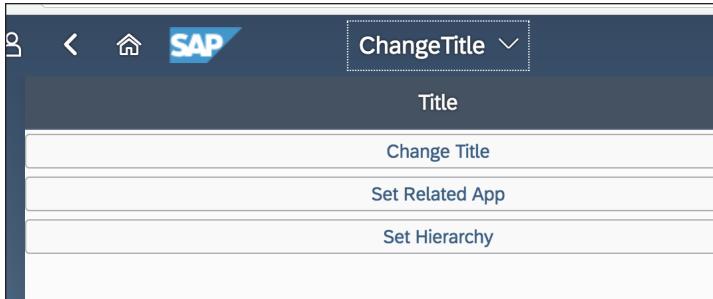
**Listing 5.1** Code for Calling Service Defined in Application Descriptor

3. Change the title accordingly. Now open `controller/Main.controller.js` and fill the `changeTitle` method with the code in [Listing 5.2](#).

```
var oComponent = this.getOwnerComponent();
oComponent.getService("ShellUIService")
  .then(function (oService) {
    oService.setTitle("New Title");//Change the title
 });
```

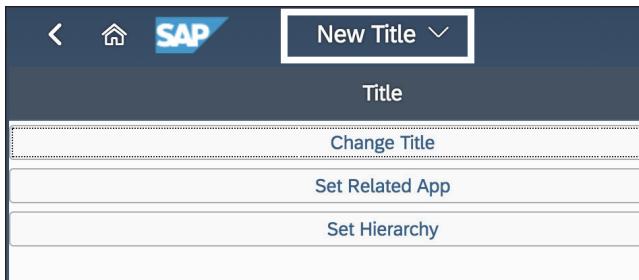
**Listing 5.2** Change Title

4. Run the app in the SAP Fiori launchpad sandbox, as shown in [Figure 5.11](#).



**Figure 5.11** Original Title Displayed

5. Click the **Change Title** button and the title of your app will be changed, as shown in [Figure 5.12](#).



**Figure 5.12** Changed App Title

Additionally, there are other ways of changing the title. First, add some setting parameters in the app descriptor; after that, the title will change to the title property you've configured in the router configuration for each target. The configuration in the app descriptor should like [Listing 5.3](#).

```
{  
  "sap.ui5": {  
    "services" : {  
      "ShellUIService": {  
        "factoryName": "sap.ushell.ui5service.ShellUIService",  
        "settings": {  
          "setTitle": "auto" // Auto change title  
        }  
      }  
    }  
  }  
}
```

```
    }
}
```

**Listing 5.3** Change Title: Service Configuration in Application Descriptor

If you want it to be even more flexible, generate a dynamic string when the router change is triggering. You can create an event listener in UIComponent for the titleChanged event of your router to change the app title via JavaScript.

**Note**

The router is a core concept of SAPUI5. To learn more about it, check out any book on basic SAPUI5 programming.

### 5.2.3 Changing the Title Context Menu

To control the context menu of the title, use setRelatedApp and setHierarchy. Continue the ChangeTitle project as follows:

1. Edit *webapp/controller/Main.controller.js*.
2. Fill the `setRelatedApp` method with the code in [Listing 5.4](#).

```
setRelatedApp: function (oEvent) {
    //This code was generated by the layout editor.
    var oComponent = this.getOwnerComponent();
    //Get the service and return a promise
    oComponent.getService("ShellUIService")
        .then(function (oService) {
            //For this method, you need to provide an array of tiles
            //The intent points to exercises in previous chapters
            //If you deploy it on SAP Cloud Portal, the link will work
            oService.setRelatedApps([
                {
                    title: "UserInfoSimple",
                    icon: "sap-icon://employee",
                    subtitle: "simple",
                    intent: "#UserInfoSimple-display"
                },
                {
                    title: "UserInfoComplex",
                    icon: "sap-icon://employee",

```

```

        subtitle: "complex",
        intent: "#UserInfoComplex-display"
    }]);
});
},

```

**Listing 5.4 ChangeTitle: Setting Related Apps**

- Fill the setHierarchy method with the code from [Listing 5.5](#).

```

setHierarchy: function (oEvent) {
//This code was generated by the layout editor.
    var oComponent = this.getOwnerComponent();
    oComponent.getService("ShellUIService")
        .then(function (oService) {
//For this method, you need to provide an array of links
//The intent points to exercises in previous chapters
//If you deploy it on SAP Cloud Portal, the link will work
            oService.setHierarchy([
                {
                    title: "User Info Simple",
                    icon: "sap-icon://employee",
                    intent: "#UserInfoSimple-display"
                },
                {
                    title: "UserInfoTest",
                    icon: "sap-icon://employee",
                    intent: "#UserInfoComplex-display"
                }
            ]);
        });
}

```

**Listing 5.5 Change Title: Set Hierarchy****Note**

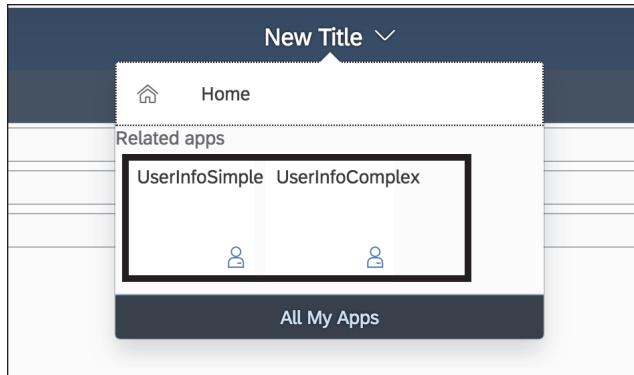
In general, the hierarchy navigation should provide a set of links, which should be accessed in the sequence. In our case, we used user info apps just for demoing the functional and visual effects.

- Deploy the application to SAP Cloud Platform and register it to SAP Fiori launchpad.
- Open the app in SAP Cloud Platform Portal.

**Tip**

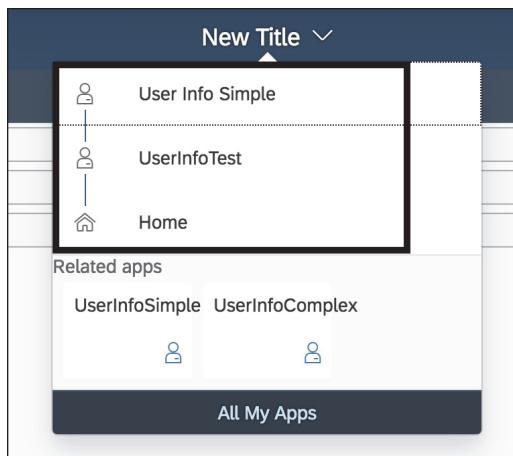
This demo also can run in the SAP Fiori launchpad sandbox, but you'll see 404 errors when clicking the links.

6. Click the **Set Related App** button, then click the arrow next to the app title. You will see tiles in the popover area, as shown in [Figure 5.13](#).



**Figure 5.13** Related Apps

7. Click the **Set Hierarchy** button and open the popover again; now you'll see hierarchical navigation options, as shown in [Figure 5.14](#).



**Figure 5.14** Hierarchical Navigation

## 5.3 Shell Header Extensions

In this section, you'll create a demo to extend the shell header. To do so, you'll need a renderer object. To get the object, use the following code:

```
var oRenderer = sap.ushell.Container.getRenderer("fiori2")
```

The methods of the renderer listed in [Table 5.4](#) can be used to extend the shell header.

| Method              | Description  |
|---------------------|--|
| setHeaderVisibility | Sets the header visibility                             |
| setHeaderTitle      | Sets the title in the SAP Fiori launchpad shell header |
| addHeaderItem       | Adds an icon at beginning of the header                |
| addHeaderEndItem    | Adds an icon at the end of the header                  |
| showHeaderItem      | Shows a header item                                    |
| showHeaderEndItem   | Shows a header end item                                |
| hideHeaderItem      | Hides a header item                                    |
| hideHeaderEndItem   | Hides a header end item                                |

**Table 5.4** Methods for Shell Header of Renderer Object

### Note

For detailed information, you can go to <https://ui5.sap.com/#/api>, search for “fiori2”, and click **Render** in the search results.

In the following sections, you'll prepare an SAPUI5 application with several buttons as the triggers for your code, that work in much the same way as previous buttons you've created. Then you'll write code for adding and managing elements in the shell header. Finally, you'll learn about the concept of state, useful to determine when your extension will be put into effect.

### 5.3.1 Preparing a Project

Create an SAPUI5 application with six buttons by following these steps:

1. Enter the SAP Web IDE full-stack version and create a new project using the information in [Table 5.5](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | ExtendHeaderBar    |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

**Table 5.5** ExtendHeaderBar: Project Settings

2. Edit the view using the layout editor, and add a VBox and a set of buttons as you did with the example in the previous section. The buttons and corresponding event handler methods should use the information in [Table 5.6](#).

| Button Text             | Event Handler Method |
|-------------------------|----------------------|
| Set header title        | setHeaderTitle       |
| Add header item         | addHeaderItem        |
| Add header end item     | addHeaderEndItem     |
| Item with link          | addItemWithLink      |
| Item with pop-up window | addItemWithPopup     |
| Item with popover       | addItemWithPopover   |

**Table 5.6** ExtendHeaderBar: Event Settings for Buttons

3. Add a label control with text “Result”.
4. Add an input control with the ID “inpResult”.
5. The resulting screen should look like [Figure 5.15](#).
6. Edit `webapp/controller/Main.controller.js`. This time you need add code to get the reference of the service and save it as a member of the controller when the view is initialized; the renderer object will be used by event handlers for every button for which you need to get the reference when the view is initializing. Create an `onInit` method and fill it with following code:

```
onInit: function () {
    this._oRenderer = sap.ushell.Container.getRenderer("fiori2");
},
```

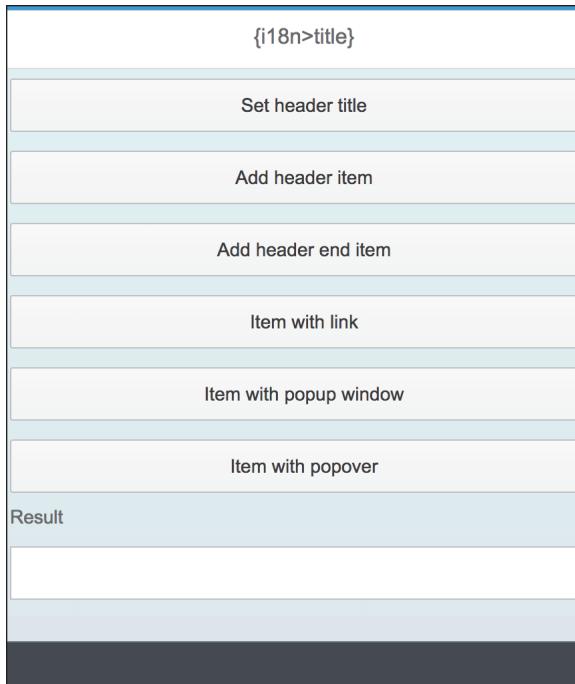


Figure 5.15 Screen for ExtendHeaderBar Demo

7. Save all your code.

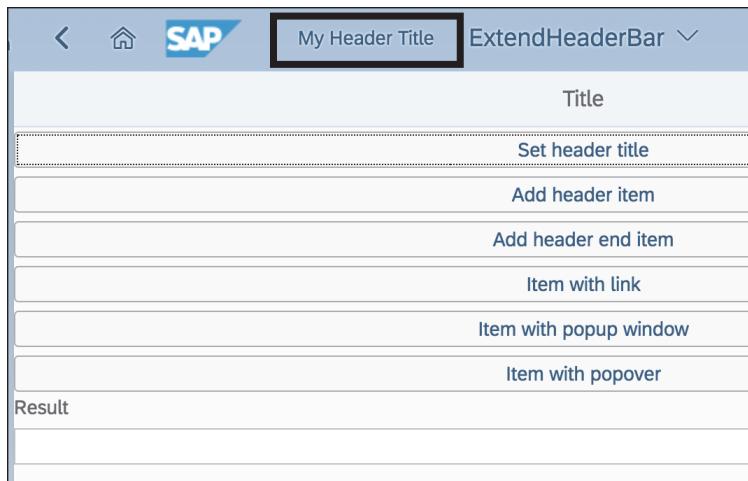
### 5.3.2 Setting a Secondary Header Title

To set a secondary header title, you need to use the `setHeaderTitle` method of the renderer, which receives a string as parameter. Fill the `setHeaderTitle` method of the controller with the code in [Listing 5.6](#).

```
setHeaderTitle: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.setHeaderTitle("My Header Title");
},
```

Listing 5.6 ExtendHeaderBar: setHeaderTitle

This is quite simple, and you can test it in SAP Fiori launchpad. The result should look like [Figure 5.16](#).



**Figure 5.16** Result of Setting Secondary Header Title

### 5.3.3 Managing Header Items

By default, in an app, you can see three elements at the beginning of the shell header: the **Back** button, **Home** button, and SAP logo (or your company's logo). On the home page, you'll see only the **Me Area** button and the logo in the header.

You're limited to three header items, and the standard ones are always at the front. That means items you've added will be hidden in an app's page, but there is room for you to add one custom header item when staying on the home page.

As a result, normally you use a header end item, which will be put on the right side. There's space for four items, but if you enabled features such as the notification center, it will be reduced to three.

To add a header item, you need to use the `addHeaderItem` method as follows:

```
this._oRenderer.addHeaderItem({  
    icon: "sap-icon://<some icon>"  
},true);
```

The first parameter is a JSON object, has the same properties as a button control. The second one indicates that if the item will display immediately. If you don't need it to

display right now, you can save the return value and call `showHeaderItem` later. The code should look like [Listing 5.7](#).

```
//Hide the control after creation
var oItem = this._oRenderer.addHeaderItem({
    icon: "sap-icon://<some icon>"
},false);
//Show the item by filling the array of IDs
this._oRenderer.showHeaderItem([oItem.getId()]);
```

**Listing 5.7** Show Header Item after Its Creation

When you want to show a header end item, the code is similar, except that you need to add a parameter to represent the type of control for the item in the first parameter. The control type must be `sap.ushell.ui.shell.ShellHeadItem`.

#### Note

Currently, the API of `addHeaderEndItem` seems not to be identical to that of `addHeaderItem`, and there is no API reference for the `ShellHeadItem` control. Those issues may be solved in future versions. You can query the API reference of the `Renderer` class in the SAPUI5 SDK to check.

The code should look like [Listing 5.8](#).

```
this._oRenderer.addHeaderEndItem(
    "sap.ushell.ui.shell.ShellHeadItem",
    {
        icon: "sap-icon://action-settings"
    },
    true);
```

**Listing 5.8** Add Header End Item

Now let's work through an example, in which you'll not only add those items, but also try out some of the common uses of header items, as follows:

1. Return to SAP Web IDE and continue editing `webapp/controller/Main.controller.js`.
2. Finish the `addHeaderItem` method with the code shown in [Listing 5.9](#).

```
addHeaderItem: function (oEvent) {  
    //This code was generated by the layout editor.  
  
    this._oRenderer.addHeaderItem({  
        icon: "sap-icon://add"  
    },true);  
},
```

**Listing 5.9** ExtendHeaderBar: Add Header Item

3. Finish the addHeaderEndItem method with the code shown in [Listing 5.10](#).

```
addHeaderEndItem: function (oEvent) {  
    //This code was generated by the layout editor.  
    this._oRenderer.addHeaderEndItem(  
        "sap.ushell.ui.shell.ShellHeadItem",  
        {  
            icon: "sap-icon://action-settings"  
        },  
        true);  
},
```

**Listing 5.10** ExtendHeaderBar: Add Header End Item

4. Add a new header end item, which will redirect to an external URL when a user clicks it. Finish the addItemWithLink method with the code shown in [Listing 5.11](#).

```
addItemWithLink: function (oEvent) {  
    //This code was generated by the layout editor.  
    this._oRenderer.addHeaderEndItem(  
        "sap.ushell.ui.shell.ShellHeadItem",{  
            icon: "sap-icon://internet-browser",  
            press: function () {  
                sap.m.URLHelper.redirect("http://www.sap-press.com");  
            }  
        },true);  
},
```

**Listing 5.11** ExtendHeaderBar: Redirect to URL after Clicking Header End Item

5. Add a new header end item, which generates a pop-up window when a user clicks it. Finish the addItemWithPopup method with the code shown in [Listing 5.12](#).

```
addItemWithPopup:function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.addHeaderEndItem(
        "sap.ushell.ui.shell.ShellHeadItem",{
            icon: "sap-icon://popup-window",
            press: function () {
                jQuery.sap.require("sap.m.MessageBox");
                sap.m.MessageBox.information(
                    "The header end item is pressed", {
                        icon: sap.m.MessageBox.Icon
                    });
            }
        },true);
},
```

**Listing 5.12** ExtendHeaderBar: Provide Pop-Up after Clicking Header End Item

6. Finally, add a new header end item, which displays a popover just under it. It needs to include an input field. After the user fills in the field with a value and clicks the **Save** button, the value should display in the **inpResult** input field on your screen. Finish the `addItemWithPopover` method with the code shown in [Listing 5.13](#).

```
addItemWithPopover: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.addHeaderEndItem(
        "sap.ushell.ui.shell.ShellHeadItem",{
            icon: "sap-icon://action",
            press: function (oEvent) {
                if (!this._oPopover) {
                    //We use JS to create a popover here
                    //You can also use an XML fragment
                    var oPopover = new sap.m.Popover({
                        title:"Popover",
                        placement:"Bottom"
                    });
                    var oInput = new sap.m.Input();
                    var oToolbar = new sap.m.Toolbar();
                    var oButton = new sap.m.Button({
                        text:"OK",
                        press: function () {
```

```
        this._oPopover.close();
        var oResult = this.byId("inpResult");
        oResult.setValue(oInput.getValue());

    }.bind(this)
    });
    oToolbar.addContent(oButton);
    oPopover.addContent(oInput);
    oPopover.setFooter(oToolbar);
    this._oPopover = oPopover;
}
this._oPopover.openBy(oEvent.getSource());
}.bind(this)
},true);
}
```

**Listing 5.13** ExtendHeaderBar: Create Popover and Catch User Input after Clicking Header End Item

7. Deploy your SAPUI5 application to SAP Cloud Platform Portal and test it.
8. Click the **Add Header Item** button, then switch to the home page to view it, as shown in Figure 5.17.



**Figure 5.17** Add Header Item

9. Enter the tile again and click **Add Header End Item**. The result should look like Figure 5.18.



**Figure 5.18** Add Header End Item

10. Click the **Item with Link** button, then click the new item; you'll jump to a website like the one shown in Figure 5.19.

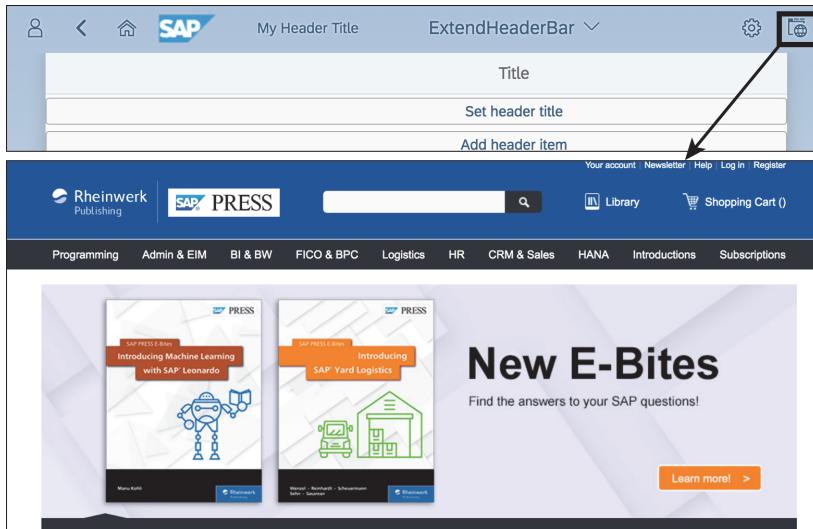


Figure 5.19 Item with Link

11. Click the **Item with Pop-Up Window** button, then click the new item, and you'll see a message box like the one shown in [Figure 5.20](#).

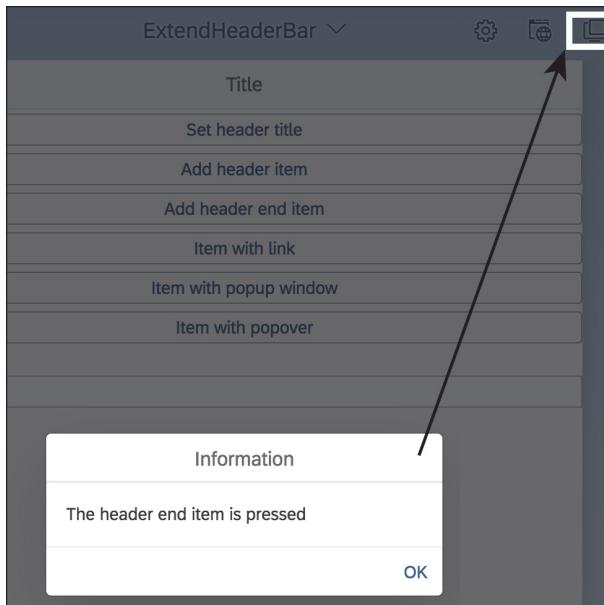
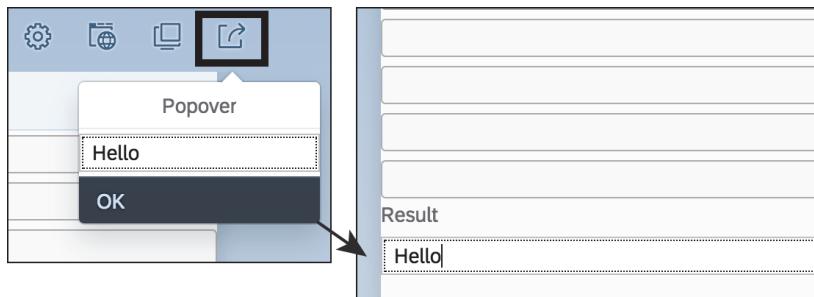


Figure 5.20 Item with Pop-Up Window

12. Click **OK** to close the pop-up window.
13. Click **Item with Popover**, then click the new item. A popover will display. Enter something in the input field and click **OK**. The data you've provided will display in the input control on the screen. It should look like [Figure 5.21](#).



**Figure 5.21** Item with Popover

#### 5.3.4 Managing Extension Element States

When you add an extension element, a factor that needs to be considered is when the extension should become available. Does the extension apply for all apps in the system or only for the current app? The SAP Fiori launchpad provides a `bCurrentState` Boolean parameter and an `aStates` array parameter to control this.

[Table 5.7](#) describes the values of these parameters and their effects.

| <code>bCurrentState</code> | <code>aStates</code>         | Effect                     |
|----------------------------|------------------------------|----------------------------|
| <code>true</code>          | N/A                          | Valid only for current app |
| <code>false</code>         | <code>["home"]</code>        | Valid only for homepage    |
| <code>false</code>         | <code>["app"]</code>         | Valid for all apps pages   |
| <code>false</code>         | <code>["home", "app"]</code> | Always valid               |
| (default)                  | (default)                    | Always valid               |

**Table 5.7** States Parameter and States Array

To use `bCurrentState` and `aStates`, you need the third and fourth parameters for the methods of the `renderer` object, as shown in the code in [Listing 5.14](#).

```
addHeaderItem: function (oEvent) {  
    //This code was generated by the layout editor.  
  
    this._oRenderer.addHeaderItem({  
        icon: "sap-icon://add"  
    },true,  
    false, //bCurrentState  
    ["app"] //aStates  
};
```

**Listing 5.14** ExtendHeaderBar: Using State Parameters

### Tip

Because some parts of the SAP Fiori launchpad API are not consistent, for some methods they don't accept parameters but instead accept JSON objects. For those methods, you can't find bcurrentState and aStates in the API document. But you can still pass those parameters as properties of the JSON object. You should write code like that shown in [Listing 5.15](#).

```
this._oRenderer.<someMethod> ({  
    icon: "sap-icon://add",  
    bVisible:true,  
    bcurrentState:false,  
    aStates:["app"]  
});
```

**Listing 5.15** ExtendHeaderBar: Passing State Parameters as Properties

## 5.4 Launch Page Extensions

In this section, you'll create subheader, footer bar, and toolset items. Although they're not as commonly used as the elements you added in the previous section, they're useful for some cases.

### 5.4.1 Preparing a Project

Create an SAPUI5 application with some buttons for testing the services you'll call in this demo:

1. Enter SAP Web IDE full-stack version and create a new project using the information in [Table 5.8](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | OtherExtensions    |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

**Table 5.8** OtherExtensions: Project Settings

2. Edit the view using the layout editor and add a VBox and a set of buttons in the same way you did in [Section 5.2](#). The buttons and corresponding event handler methods should be as shown in [Table 5.9](#).

| Button Text                      | Event Handler Method      |
|----------------------------------|---------------------------|
| Add a subheader                  | addSubHeader              |
| Add a footerbar                  | addFooterBar              |
| Add a simple tool area item      | addSimpleToolareaItem     |
| Add an expandable tool area item | addExpandableToolareaItem |

**Table 5.9** OtherExtensions: Event Listeners

3. The result should look like [Figure 5.22](#).
4. Edit *webapp/controller/Main.controller.js*. This time you need to add code to get the reference of the service and save it as a member of the controller when the view is initialized, since the renderer object will be used by event handlers for every button you need to get the reference when the view is initializing. Create an **onInit** method and fill it with the following code:

```
onInit: function () {
    this._oRenderer = sap.ushell.Container.getRenderer("fiori2");
},
```

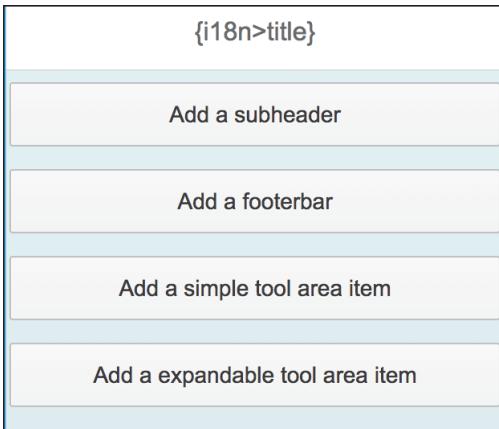


Figure 5.22 Screen for OtherExtensions Demo

5. Save all your code.

#### 5.4.2 Adding a Subheader

A subheader can act as an additional toolbar or title bar. A sap.m.Bar, which has three aggregations for left-, middle-, and right-aligned controls, is used to create the header toolbar.

You can implement the addSubHeader method using the code shown in [Listing 5.16](#).

```
addSubHeader: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.addShellSubHeader({
        //sap.m.Bar is the only accepted value
        controlType: "sap.m.Bar",
        //A JSON object contains all properties/aggregations/events for the bar
        oControlProperties: {
            contentLeft: [
                new sap.m.Button({
                    icon: "sap-icon://accept"
                })
            ],
            contentMiddle: [
                new sap.m.Label({
                    text: "My Sub Header"
                })
            ]
    });
}
```

```
        })
    ],
    contentRight: [
        new sap.m.Button({
            icon: "sap-icon://decline"
        })
    ]
},
//visible and state information provided in the attribute
bIsVisible: true,
bCurrentState: false
});
},
```

**Listing 5.16** OtherExtensions: Add Subheader

The test result of this button should look like [Figure 5.23](#).



**Figure 5.23** Adding Subheader

### 5.4.3 Adding a Footer Bar

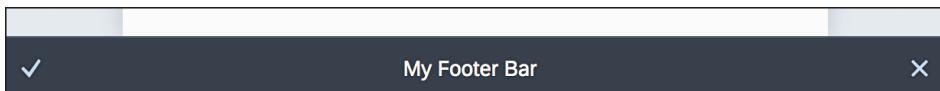
As the counterpart of subheader, you can also add a footer bar. To do so, change the implementation of the `addFooterBar` method with the code shown in [Listing 5.17](#).

```
addFooterBar: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.setShellFooter({
        //sap.m.bar is the only accepted value
        controlType: "sap.m.Bar",
        oControlProperties: {
            contentLeft: [
                new sap.m.Button({
                    icon: "sap-icon://accept"
                })
            ],
        }
    });
},
```

```
        contentMiddle: [
            new sap.m.Label({
                text: "My Footer Bar"
            })
        ],
        contentRight: [
            new sap.m.Button({
                icon: "sap-icon://decline"
            })
        ]
    }
});
```

### **Listing 5.17** OtherExtensions: Add Footer Bar

The testing result should like Figure 5.24.



**Figure 5.24** Add Footer Bar

#### 5.4.4 Adding Tool Area Items

Tool area items are clickable and expandable items displayed in the left tool area. The tool area is not displayed by default because there is no item in it by default. When you add an item to it, it will display automatically.

There are two kinds of items: normal and expandable. For normal items, the user can just click them to access them. For expandable ones, there will be a small triangle in the bottom-right corner of the item, the `onExpand` callback function will be called, and usually a popover on the right should be displayed.

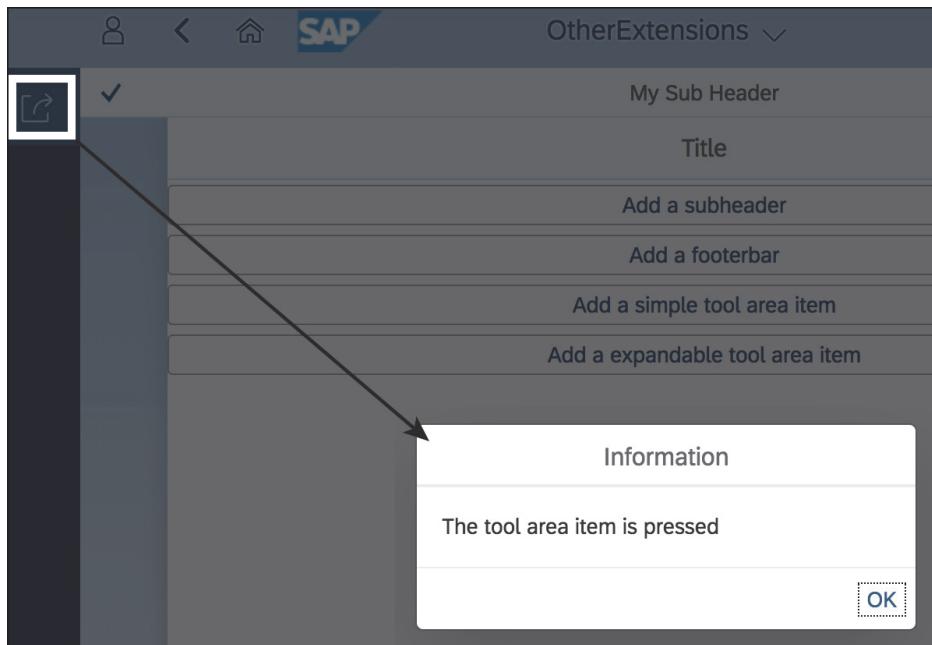
To implement a normal tool area item, fill the `addSimpleToolAreaItem` method with the code shown in Listing 5.18.

```
addSimpleToolareaItem: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.addToolAreaItem({
        icon: "sap-icon://action",
```

```
press: function (oEvent) {
    jQuery.sap.require("sap.m.MessageBox");
    sap.m.MessageBox.information(
        "The tool area item is pressed", {
            icon: sap.m.MessageBox.Icon
        });
    },
},
//isVisible and state
true,
false);
},
```

**Listing 5.18** OtherExtensions: Add Simple Tool Area Item

Run it in the SAP Fiori launchpad sandbox or the SAP Cloud Platform Portal and click the corresponding button. A tool area will be displayed at the left. Click the first item and a pop-up window will be generated, as shown in [Figure 5.25](#).



**Figure 5.25** Add Simple Tool Area Item

Now, add an expandable tool area item. Change the code of *webapp/controller/Main.controller.js* by altering the `addExpandableToolareaItem` method according to the code shown in [Listing 5.19](#).

```
addExpandableToolareaItem: function (oEvent) {
    //This code was generated by the layout editor.
    this._oRenderer.addToolAreaItem({
        icon: "sap-icon://expand",
        //Set expandable to true
        expandable: true,
        expand: function (oEvent) {
            if (!this._oPopover) {
                var oPopover = new sap.m.Popover({
                    title: "Popover",
                    placement: "Right"
                });
                var oInput = new sap.m.Input();
                var oToolbar = new sap.m.Toolbar();
                var oButton = new sap.m.Button({
                    text: "OK",
                    press: function () {
                        this._oPopover.close();
                    }.bind(this)
                });
                oToolbar.addContent(oButton);
                oPopover.addContent(oInput);
                oPopover.setFooter(oToolbar);
                this._oPopover = oPopover;
            }
            this._oPopover.openBy(oEvent.getSource());
        }.bind(this)
    }, true, false);
}
```

**Listing 5.19** OtherExtensions: Add Expandable Tool Area Item

When you test the code, you'll see a new item with a small triangle. Click the triangle and the popover will display, as shown in [Figure 5.26](#).

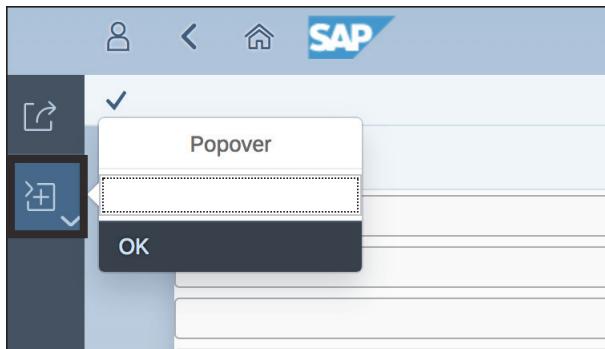


Figure 5.26 Tool Area Item with Popover

## 5.5 Me Area Extensions

You've already added lots of elements to the home page. In this section, we'll change the focus to the Me Area. You'll create a project to add content to the Me Area and setting options to the settings dialog.

### 5.5.1 Preparing a Project

Create a SAPUI5 application with some buttons for testing the services you'll call in this demo:

1. Enter SAP Web IDE full-stack version and create a new project using the information in Table 5.10.

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | ExtendMeArea       |
| Namespace    | fipdev             |
| View Type    | XML                |
| View Name    | Main               |

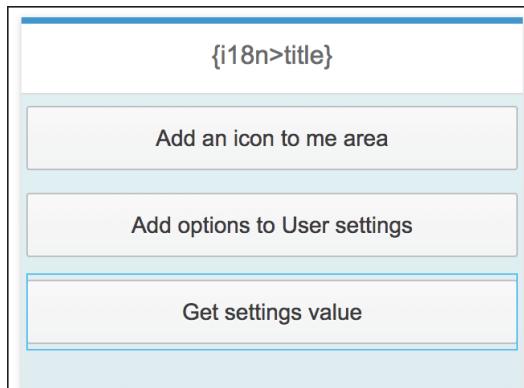
Table 5.10 ExtendMeArea: Project Settings

2. Edit the view using the layout editor and add a VBox and a set of buttons using the same methods detailed in [Section 5.2](#). The buttons and corresponding event handler methods should be as shown in [Table 5.11](#).

| Button Text                  | Event Handler Method |
|------------------------------|----------------------|
| Add an icon to me area       | addIcon              |
| Add options to User Settings | addSetting           |
| Get settings value           | getSettingsValue     |

**Table 5.11** ExtendMeArea: Event Handlers

3. The result should look like [Figure 5.27](#).



**Figure 5.27** Screen for Extending Me Area

4. Edit `webapp/controller/Main.controller.js`. This time you need add code to get the reference of the service and save it as a member of the controller when the view is initialized. Since the renderer object will be used by event handlers for every button, you need to get the reference when the view is initializing. Create an `onInit` method and fill it with following code:

```
onInit: function () {
  this._oRenderer = sap.ushell.Container.getRenderer("fiori2");
},
```

5. Save your application.

### 5.5.2 Adding a Button to the Me Area

The Me Area will display when a user clicks the person icon at the top-left corner of SAP Fiori launchpad. It contains the following buttons for the user:

- **App Finder**

Here a user will see all the tile catalogs she can access.

- **Settings**

This lets the user change the theme, language, and other aspects of SAP Fiori launchpad.

- **Give Feedback**

This is an optional function configured by the administrator to let a user share her ideas in a simple way. The feedback may be sent to SAP directly if the customer allows this and has configured it as an option.

- **About**

This button can be used to check general information for the SAP Fiori launchpad.

As a developer, you can add additional buttons to the Me Area. Note that the **App Finder** button should always be first. The button you add can either be the second button or be the last one.

Another constraint is that there is a maximum of five positions for buttons in the Me Area; since there are three or four buttons by default, you only have one or two spaces for custom buttons. If the number is exceeded, the last button will change to "..."; you can access additional buttons by clicking on it.

For now, add a button called **New Action** to the Me Area. To do so, change the `addIcon` method in your controller according to the code shown in [Listing 5.20](#).

```
addIcon: function (oEvent) {  
    this._oRenderer.addUserAction({  
        //Only sap.m.button is acceptable  
        controlType: "sap.m.Button",  
        oControlProperties: {  
            text: "NewAction",  
            icon: "sap-icon://accept",  
            press: function () {}  
        },  
        bIsVisible: true,  
        bCurrentState: false,  
    });  
},
```

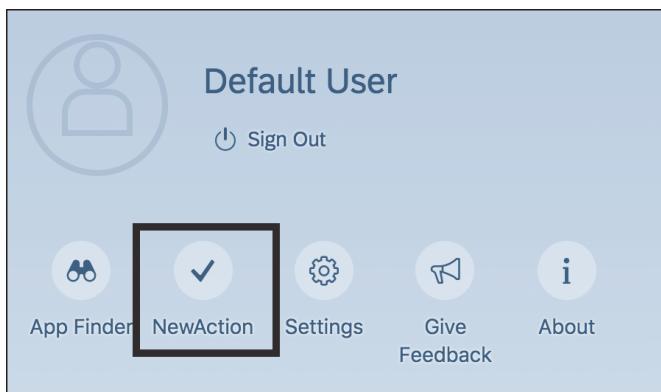
```
bIsFirst: true
});
},
},
```

**Listing 5.20** ExtendMeArea: Add Me Area Button

The `addUserAction` method accepts a JSON object, which has the following properties:

- The `ControlType` property only accepts `sap.m.Button` as a value.
- `oControlProperties` is a JSON object that describes properties, aggregations, and events of `sap.m.Button`.
- `bisVisible` indicates whether the button is shown.
- `bCurrentState` is generally set to false, because when a user opens the Me Area, the state changes, and if the value is true, you'll never see the new button.
- If `bIsFirst` is set to true, the button will occupy the second position (remember, the **App Finder** button will always be first). Otherwise it will be the last button.

To test the button, use the SAP Fiori launchpad sandbox or SAP Cloud Platform Portal. The test result should look like [Figure 5.28](#).



**Figure 5.28** Test Result for Adding Button to Me Area

### 5.5.3 Adding Setting Options

You can also add customized options in the user settings dialog. To add settings, you need to use `addUserPreferencesEntry` to add an entry.

The following properties are important for the entry:

- title  
The title displayed to the user.
- value  
A unique ID to identify your entry.
- content  
An event handler for when the content of settings needs to be loaded. It returns a deferred jQuery object, which returns the contents of the settings panel when it's resolved.
- onSave  
With this event handler, when a user saves the settings, a deferred object is needed as a returning value, which contains objects you want to access later.
- onCancel  
An event handler for when a user clicks the **Cancel** button.

Note that the loading and operation of settings entries are controlled by SAP Fiori launchpad itself. Most of those operations are asynchronous, and deferred objects are commonly used in such cases. If you aren't familiar with this, just follow the code in [Listing 5.21](#) and remember to put everything you need to return into the resolve method as an object.

Now change the `addSetting` method in the control with the code shown in [Listing 5.21](#).

```
addSetting: function (oEvent) {
    //You want to display an input control
    var oInput = new sap.m.Input();
    var oEntry = {
        title: "Custom Settings",
        value: function () {
            return jQuery.Deferred().resolve("customSettings");
        },
        content: function () {
            //Declare the deferred object
            var deferred = jQuery.Deferred();
            //Provide the input control when this page is loading
            deferred.resolve(oInput);
            //Return the deferred object
            return deferred;
        },
        onSave: function () {
```

```

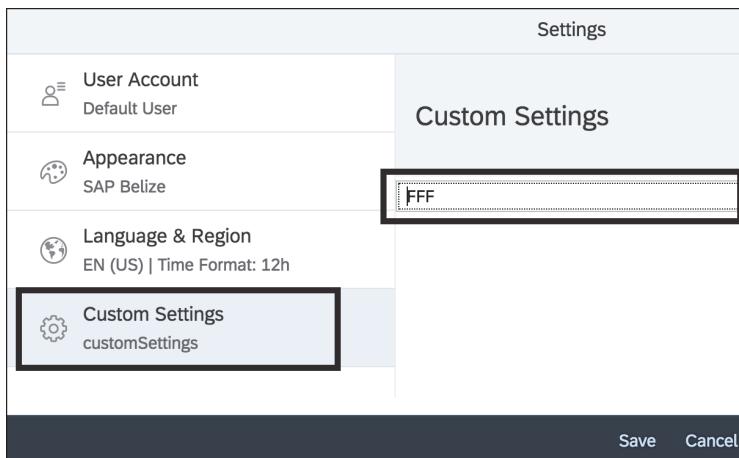
        //For now, we just have a message without saving anything
        sap.m.MessageToast.show("Settings Save");
    }
};

//Add the entry
this._oRenderer.addUserPreferencesEntry(oEntry);
},

```

**Listing 5.21** ExtendMeArea: Add Setting Options

To test, use the SAP Fiori launchpad sandbox or SAP Cloud Platform Portal. The test result should like [Figure 5.29](#).



**Figure 5.29** Test Result for Adding User Settings Entries

#### 5.5.4 Fetching Data from Custom Setting Options

Because the newly added setting is available, you now need to access the setting values. As you've learned, the `onSave` method of the entry should return a deferred object that contains an object representing user input. But how do you access the deferred object and get a value from it? In this section, you'll change the previous implementation and we'll show you how to get information from the entry.

Continue to change the code of the controller. In the `onSave` event for the entry, you need to create a deferred object and save it as a property of the controller itself, so that it can be accessed in any method of the controller. This makes it so that all your

event handlers in the controller can get data from user settings. When the deferred object is resolved, you need to return the value of the input control.

To access the controller in the `onSave` method, you need to set its listener to the controller by adding `bind(this)`.

Now change the `onSave` method to reflect the code shown in [Listing 5.22](#).

```
onSave: function () {
    this._inputValueDeferred = new jQuery.Deferred();
    this._inputValueDeferred.resolve(oInput.getValue());
    return this._inputValueDeferred;
}.bind(this)
```

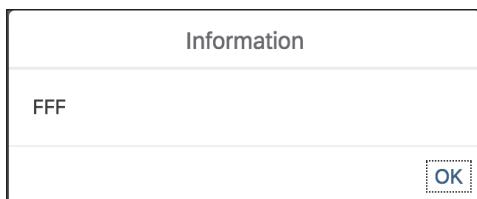
**Listing 5.22** ExtendMeArea: Callback Function when User Saves Settings

In the `getSettingsValue` method, you can access the deferred object and get its value when it's resolved. Change the code for the `getSettingsValue` method as shown in [Listing 5.23](#).

```
getSettingsValue: function (oEvent) {
    //This code was generated by the layout editor.
    this._inputValueDeferred.then(function (sValue) {
        jQuery.sap.require("sap.m.MessageBox");
        sap.m.MessageBox.information(
            sValue, {
                icon: sap.m.MessageBox.Icon
            });
    });
}
```

**Listing 5.23** ExtendMeArea: Fetch Value Set by User

Test it, and the results should look like [Figure 5.30](#).



**Figure 5.30** Test Result for Getting Value from User Settings Entry

## 5.6 Summary

After reading this chapter, you should be able to extend the SAP Fiori launchpad by adding your own custom elements. This is quite useful when you want to make use of the space of SAP-reserved UI areas. You can either add special functions to your apps or add those functions globally in the system.

In next chapter, we'll discuss how to create new types of tiles beyond the standard tile types provided by SAP to further customize the experience of your SAP Fiori launchpad.



# Chapter 6

## Custom Tile Types

*Tile design is important for making a good first impression on an end user. In this chapter, you'll learn how to improve that first impression by creating your own tiles to best suit your requirements.*

By default, SAP provides standard tiles, dynamic tiles, and news tiles on SAP Fiori launchpad. If you've enabled the SAP Smart Business service in SAP Cloud Platform or corresponding services in an on-premise system, you'll also have a couple of new tiles with charts to display KPI info quickly.

In this chapter, you'll learn not only how to design a tile type of your own but also how to deploy it on both SAP Cloud Platform Portal and on-premise SAP NetWeaver AS ABAP. You'll also learn how to integrate customized tiles with tile management abilities provided by those platforms.

### 6.1 Creating a Custom Tile

The `GenericTile` control under package `sap.m` is the foundation of any type of tile. All custom tiles should be based on the generic tile to ensure they display well in SAP Fiori launchpad.

In this section, you'll learn basics of the generic tile control and will see which types of content can be embedded into it. Then you'll create a custom tile, guided by an example.

#### 6.1.1 Basics of a Generic Tile

In [Figure 6.1](#), you can see that the generic tile provides basic information such as a header and a subtitle. A `TitleContent` control that's about 60% of the height of the tile occupies the bottom part of the tile. It also provides properties like a footer and a unit to help you quickly display information in the footer.

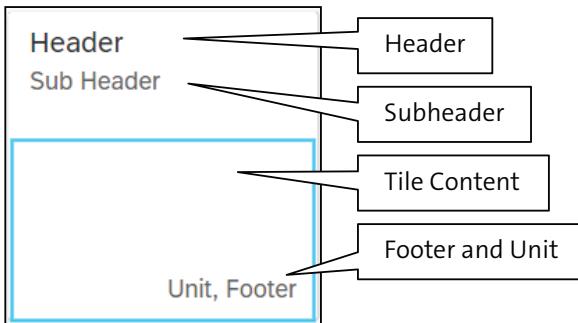


Figure 6.1 Basic Structure of a Generic Tile

*Frame types* describe the shape and how much space a tile will occupy in SAP Fiori launchpad. In Figure 6.2, you can see the visual effect of tiles of all frame types.

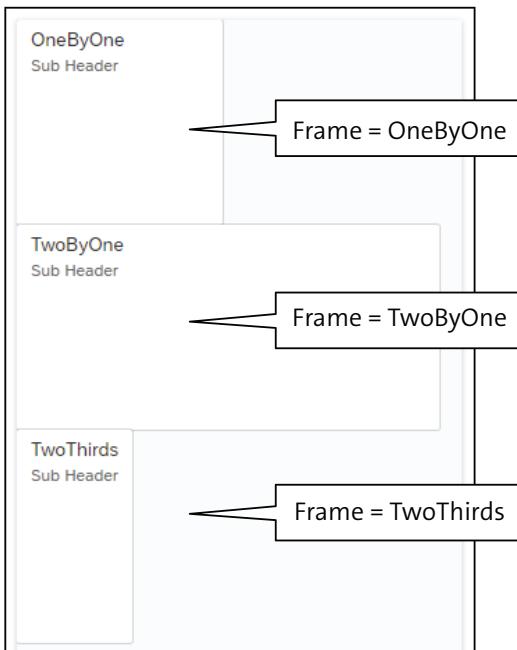


Figure 6.2 Tiles of Different Frame Types

For the tile, you can also set a header image, background image, and other details. For more information, see the detailed document at <https://ui5.sap.com/#/api/sap.m.GenericTile>.

### 6.1.2 Creating a Generic Tile

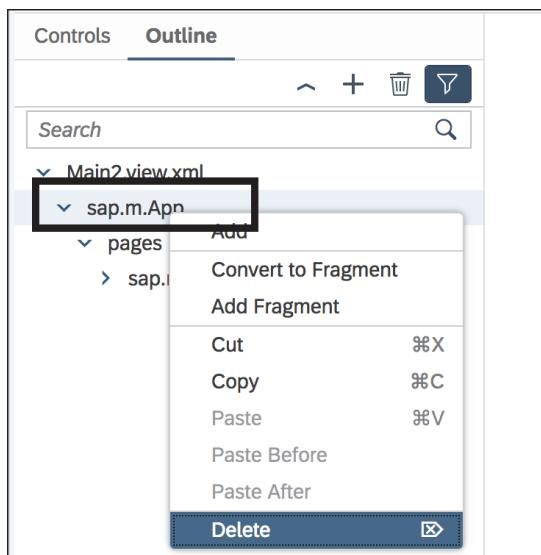
Now it's time for you to create your first tile using the following instructions:

1. Open SAP Web IDE full-stack version.
2. Create a new SAPUI5 project, using the information in [Table 6.1](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | MyTile             |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Main               |

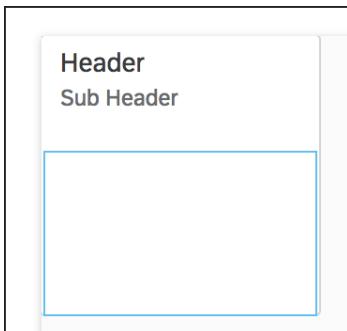
**Table 6.1** MyTile: Project Settings

3. Open the main view using the layout editor.
4. Switch to the **Outline** panel, right-click the app node of the view, and select **Delete**, as shown in [Figure 6.3](#).



**Figure 6.3** Delete App Node of View

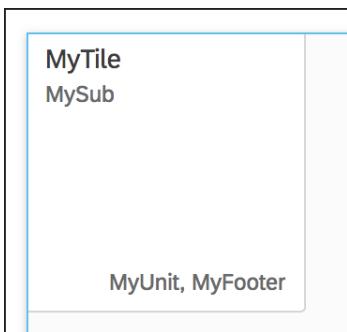
5. Drag a **GenericTile** control into the view, then drag a **TileContent** control into it. Note that you must make sure that the **TileContent** control is within the **GenericTile** control. The result should look like Figure 6.4.



**Figure 6.4** Initial State of Generic Tile with Tile Content

6. Using the layout editor, set the following properties:
  - Set the header of **GenericTile** to “MyTile”.
  - Set the subheader of **GenericTile** tile to “MySub”.
  - Set the footer of **TileContent** control to “MyFooter”.
  - Set the unit of **TileContent** to “MyUnit”.

The result should look like Figure 6.5.



**Figure 6.5** Result of First Custom Tile

7. Save all files, and then run and test the visual effect of the tile in the SAP Fiori launchpad sandbox.

### 6.1.3 Organizing Tile Content

The `TileContent` control accepts any type of control in SAPUI5; however, there is a set of controls designed specifically for the tile, which should be taken into consideration first. [Table 6.2](#) lists all controls designed specifically as content for `TileContent`.

| Control Name   | Usage  |
|----------------|--|
| FeedContent    | Shows the tile containing the text of the feed, a subheader, and a numeric value.                    |
| ImageContent   | Shows <code>ImageContent</code> that can include an icon, a profile image, or a logo with a tooltip. |
| NewsContent    | This control is used to display the news content text and subheader in a tile.                       |
| NumericContent | Shows a number, especially a number with its criticality, trend, and units, which is a KPI.          |

**Table 6.2** Controls Designed Specifically as Content of `TileContent`

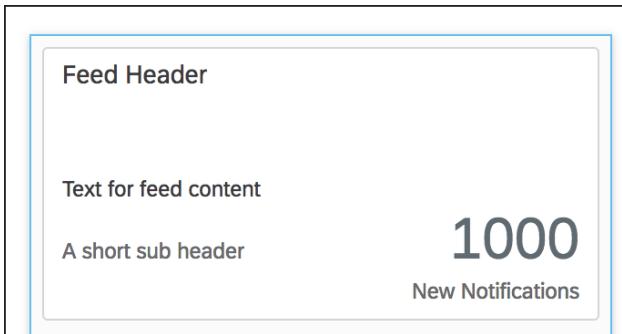
Currently, the layout editor of SAP Web IDE doesn't support those controls. To use them, you need to change the source code of the XML view manually using the code editor, which can be opened by just double-clicking the view. The results of tiles created with each control will look as follows:

- FeedContent

The XML view shown in [Listing 6.1](#) creates a generic tile with `FeedContent`. The result will look like [Figure 6.6](#).

```
<GenericTile header="Feed" frameType="TwoByOne">
  <TileContent footer="New Notifications">
    <FeedContent contentText="Text for feed content"
      subheader="A short sub header" value="1000"/>
  </TileContent>
</GenericTile>
```

**Listing 6.1** Feed Content



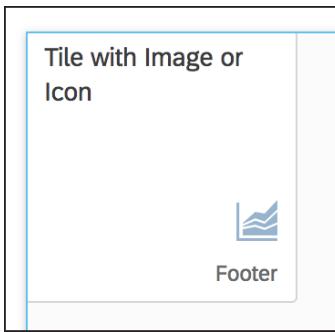
**Figure 6.6** Generic Tile with Feed Content

- **ImageContent**

The XML view shown in [Listing 6.2](#) creates a generic tile with `ImageContent`. The result will look like [Figure 6.7](#).

```
<GenericTile header="Tile with Images or Icon" frameType="OneByOne" >
    <TileContent footer="footer">
        <ImageContent src="sap-icon://area-chart" description="Icon"/>
    </TileContent>
</GenericTile>
```

**Listing 6.2** Image Tile Content



**Figure 6.7** Generic Tile with Image Content

- **NewsContent**

The XML view in [Listing 6.3](#) creates a generic tile with `NewsContent`. The result will look like [Figure 6.8](#).

```
<GenericTile header="Breaking News" frameType="TwoByOne" backgroundImage="css/background.png">
    <TileContent footer="Buy Now!">
        <NewsContent contentText="Want to unreleash the power of Fiori Launchpad?....."
            subheader="Released today by SAP-Press"/>
    </TileContent>
</GenericTile>
```

**Listing 6.3** News Tile Content



**Figure 6.8** Generic Tile with News Content

- NumericContent

The XML view in [Listing 6.4](#) creates a generic tile with NumericContent. The result will look like [Figure 6.9](#).

```
<GenericTile header="Sales" frameType="OneByOne">
    <TileContent unit="EUR" footer="today">
        <NumericContent scale="M" value="1.96" valueColor="Error" indicator="Up"/>
    </TileContent>
</GenericTile>
```

**Listing 6.4** Numeric Tile Content

Another set of commonly used controls are microchart controls. These controls can help you render a microchart in the tile and make sure the performance is good when lots of tiles display on the same page.



Figure 6.9 Generic Tile with Numeric Content

The XML view in [Listing 6.5](#) creates a generic tile with a ColumnMicroChart. The result will look like [Figure 6.10](#).

```
<GenericTile header="Sales by Quarter" frameType="OneByOne">
    <TileContent unit="EUR" footer="This year">
        <c:ColumnMicroChart>
            <c:columns>
                <c:ColumnMicroChartData value="65" color="Error"/>
                <c:ColumnMicroChartData value="90" color="Good"/>
                <c:ColumnMicroChartData value="75" color="Critical"/>
                <c:ColumnMicroChartData value="95" color="Good"/>
            </c:columns>
        </c:ColumnMicroChart>
    </TileContent>
</GenericTile>
```

[Listing 6.5](#) MicroChart in Tile

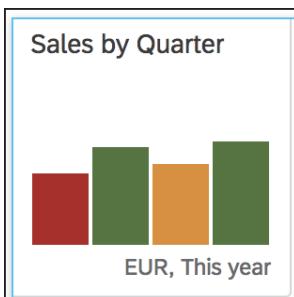


Figure 6.10 Generic Tile with ColumnMicroChart

To see more examples of tiles, go to <https://ui5.sap.com/#/controls>, then find samples under the **Data Visualization and Tile** group.

### Warning

Complex controls may lead to performance issues. As the developer who creates the tile, you have no idea how many occurrences of your tile will be on the initial page of SAP Fiori launchpad.

#### 6.1.4 Creating a Slide Tile

When one tile isn't enough for you, you can create a tile that can slide to show more content. The example in [Listing 6.6](#) combines a tile with `NumericContent` and another tile with `ColumnMicroChart` and slides between them.

```
<SlideTile>
  <tiles>
    <GenericTile header="Sales" frameType="OneByOne">
      <TileContent unit="EUR" footer="today">
        <NumericContent scale="M" value="1.96" valueColor="Error"
          indicator="Up"/>
      </TileContent>
    </GenericTile>
    <GenericTile header="Sales by Quarter" frameType="OneByOne">
      <TileContent unit="EUR" footer="This year">
        <c:ColumnMicroChart>
          <c:columns>
            <c:ColumnMicroChartData value="65" color="Error"/>
            <c:ColumnMicroChartData value="90" color="Good"/>
            <c:ColumnMicroChartData value="75" color="Critical"/>
            <c:ColumnMicroChartData value="95" color="Good"/>
          </c:columns>
        </c:ColumnMicroChart>
      </TileContent>
    </GenericTile>
  </tiles>
</SlideTile>
```

**Listing 6.6** Slide Tile

The example uses the `SlideTile` control and fills two generic tiles in the tiles aggregation.

### Tip

We strongly recommend following through with this example on your computer. It contains an animation that doesn't translate well to the static printed (or electronic) page.

#### 6.1.5 Adding Content to Your Tile

These instructions build off the tile you created in [Section 6.1.2](#). To begin adding content to your tile, follow these instructions:

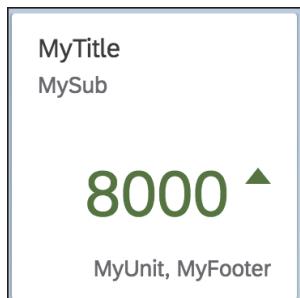
1. Return to SAP Web IDE.
2. Locate **Webapp • View • Main.view.xml**, right-click, then choose **Open Code Editor** from the context menu.
3. Locate the space between `<tileContent>` and `</tileContent>`. The code between them should be as follows:

```
<TileContent id="content0" footer="MyFooter" unit="MyUnit"/>
```

Change the code as follows to add a `NumericContent` control and change the `unit` property of `TileContent` accordingly:

```
<TileContent id="content0" footer="MyFooter" unit="EUR">
<NumericContent value="8000" valueColor="Good" indicator="Up"/>
</TileContent>
```

4. Save and run your tile in your SAP Fiori launchpad sandbox. Your result should look like [Figure 6.11](#).



**Figure 6.11** Result of Customized Tile in SAP Fiori Launchpad Sandbox

## 6.2 Deploying a Custom Tile to SAP Cloud Platform

Before deploying your tile to the SAP Cloud Platform Portal, let's walk through what you need to consider:

1. At first, your tile needs to be accessible from SAP Cloud Platform Portal, which means you must deploy your tile to SAP Cloud Platform and register it as an app to SAP Fiori launchpad.
2. Then you need to set the usage of the tile for other apps in the **Visualization** tab in the configuration page of an app.
3. The content in the tile should be customizable. It should display values or get data from a backend service. The values or URLs of backend services should be provided by the administrator when she configures the app. The tile itself can render against that information dynamically at runtime.
4. Finally, when a user clicks the tile, it should behave just like other standard tiles. The cross-app navigation service should be used to identify and check the availability of the target URL.

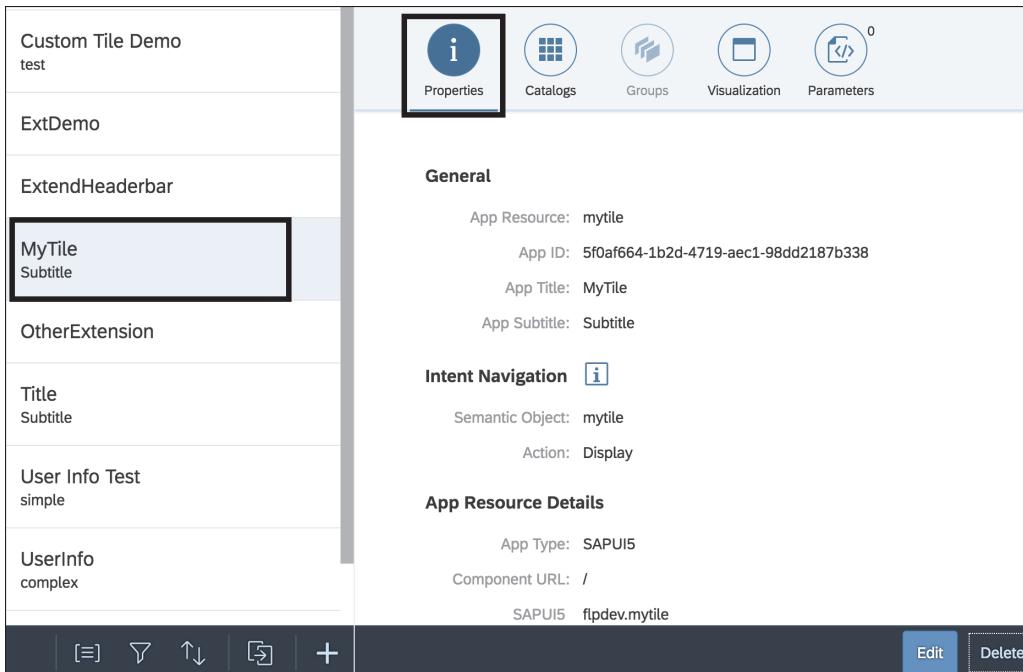
As a result, *deploy*, *apply*, *parameter*, and *navigation* are keywords you need to keep in mind when implementing a custom tile in SAP Cloud Platform Portal. In the following four sections, you'll experience each of them. You'll first develop a simple custom tile with static content and then deploy it into SAP Cloud Platform Portal. Then you'll apply this tile type as a tile for your SAP Fiori application. Finally, you'll turn all the content of this tile into parameters that can be set by your administrator.

### 6.2.1 Deploying Your Tile to SAP Cloud Platform Portal

In this section, you'll simply deploy your new tile developed in [Section 6.1.5](#) to SAP Cloud Platform Portal, as follows:

1. In SAP Web IDE, choose the **MyTile** project, and deploy your project by choosing **Deploy • Deploy to SAP Cloud Platform**.
2. Leave the default options unchanged and click **Deploy**.
3. Wait for this step to complete, then click **Register to SAP Fiori Launchpad** in the pop-up window once it appears.
4. Keep all things default and click **Next** twice, then click **Finish**.
5. Click **OK** to close the pop-up window.

6. Open a new browser window, then open the **Administration** page of SAP Cloud Platform Portal (which you bookmarked in [Chapter 2](#)).
7. Follow menu path **Content Management • Apps** in the left panel.
8. Locate **MyTile**, click it, and click **Edit**, as shown in [Figure 6.12](#).



**Figure 6.12** Edit Application Settings

9. Switch to the **Visualization** tab and choose **No Tile** for the **Tile Type** property, as shown in [Figure 6.13](#). The tile is not like a normal SAPUI5 application. We add the tile here just so it can be accessed via SAP Fiori launchpad, and we choose **No Tile** because it doesn't need a tile display in SAP Fiori launchpad.
10. Click **Save**.
11. Switch to the **Properties** tab, scroll down, find the **App Resources Details** section, and jot down the values for **SAPUI5 Component** and **HTML5 App Name**. You'll use these values when setting visualization options for your app. This tab is shown in [Figure 6.14](#).

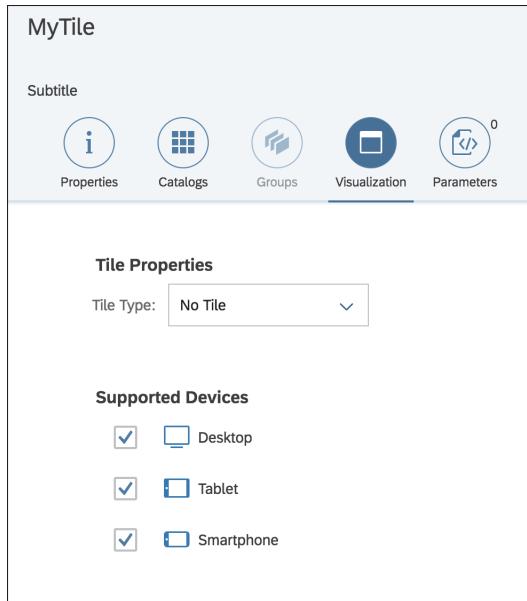


Figure 6.13 Visualization for Custom Tile

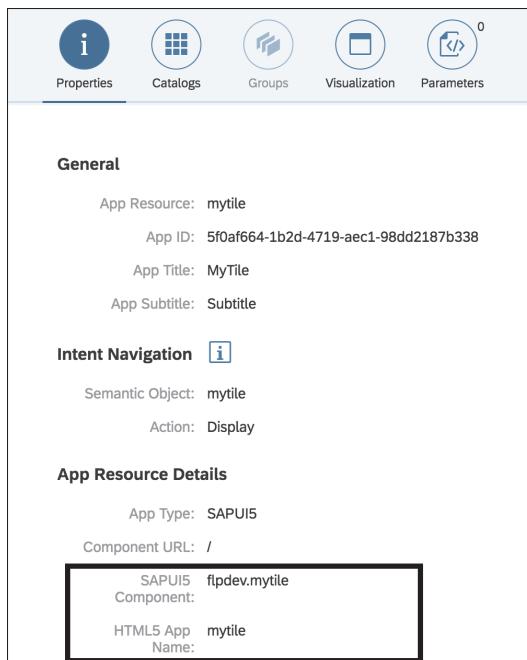


Figure 6.14 Application Properties for Custom Tile

Now you've deployed your tile successfully in SAP Fiori launchpad. Next, you'll use the tile for one of the apps you created in the previous chapters.

### 6.2.2 Applying Your Tile to an SAP Fiori App

In this section, you'll take an app you've deployed into SAP Cloud Platform Portal and change the tile to the one you've just made by following these instructions:

1. Open a new browser window, then open the **Administration** page of SAP Cloud Platform Portal (which you bookmarked in [Chapter 2](#)).
2. Follow menu path **Content Management • Apps** in the left panel.
3. Locate any app you've created before—for example, UserInfo (created in [Chapter 4](#))—click it, and click **Edit**.
4. Switch to the **Visualization** tab and change the values according to [Table 6.3](#). The result should look like [Figure 6.15](#).

| Property    | Value               | Explanation   |
|-------------|---------------------|---|
| Tile Type   | Custom App Launcher | Tell the system you're using your own tile.   |
| Size        | 1×1                 | Frame type.   |
| Title       | <AnythingYouWant>   | Title of the tile.  |
| Module Type | SAPUI5              | It will use a SAPUI5 component. In some cases, you can also use a view and choose corresponding options related to the view type. |
| Name        | flpdev.MyTile       | Namespace of the component.   |
| Prefix      | flpdev.MyTile       | Namespace of the component.   |
| Path        | /sap/fiori/mytile   | This is structured as /sap/Fiori/<UI5 application name>.  |

**Table 6.3** Visualization Parameters for App with Custom Tile

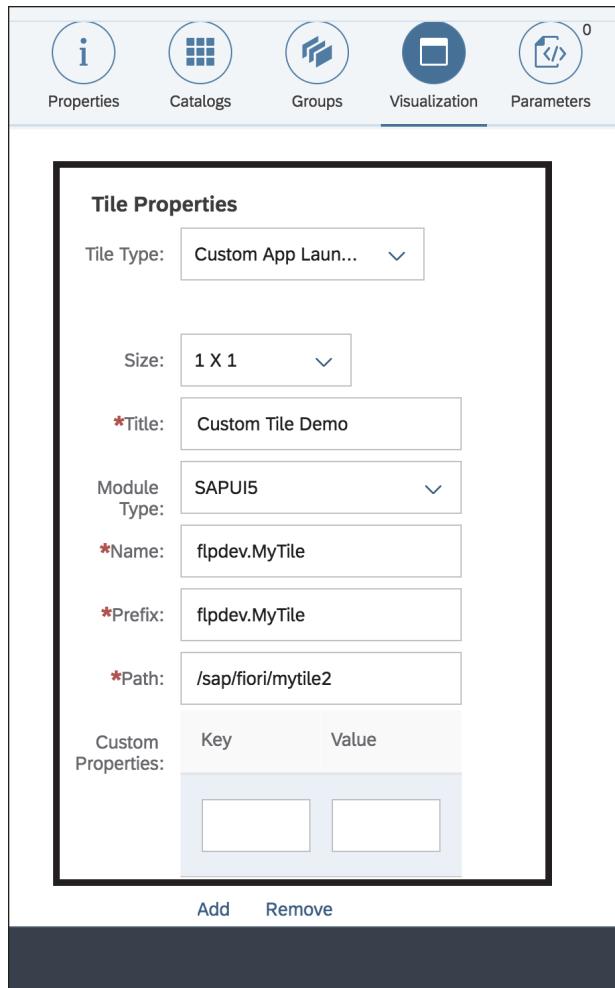


Figure 6.15 Custom Tile Settings for App

5. Save the configuration.
6. Republish your site by clicking the globe button, then check the checkbox for **Clear HTML5 Application Cache** and click **Publish**, as shown in Figure 6.16.

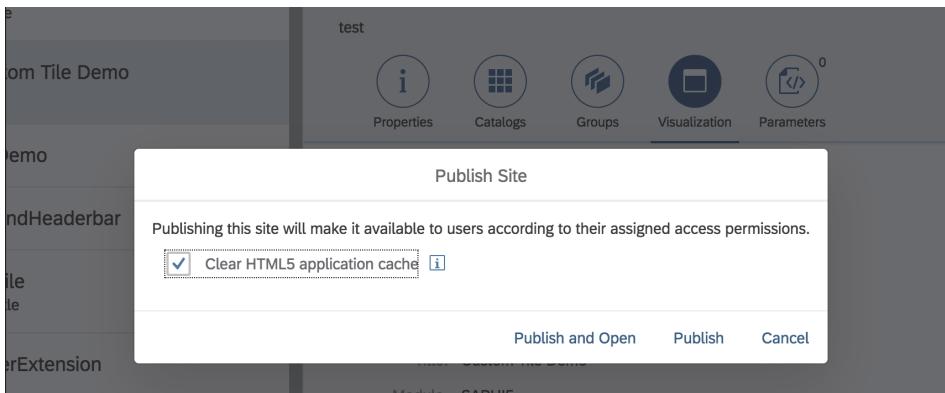


Figure 6.16 Republish SAP Fiori Launchpad Site

7. Open a new tab and open the SAP Cloud Platform Portal page as an end user from your bookmark. If you've currently added the app from the catalog to a group, you'll see your new tile. The result should look like [Figure 6.17](#).

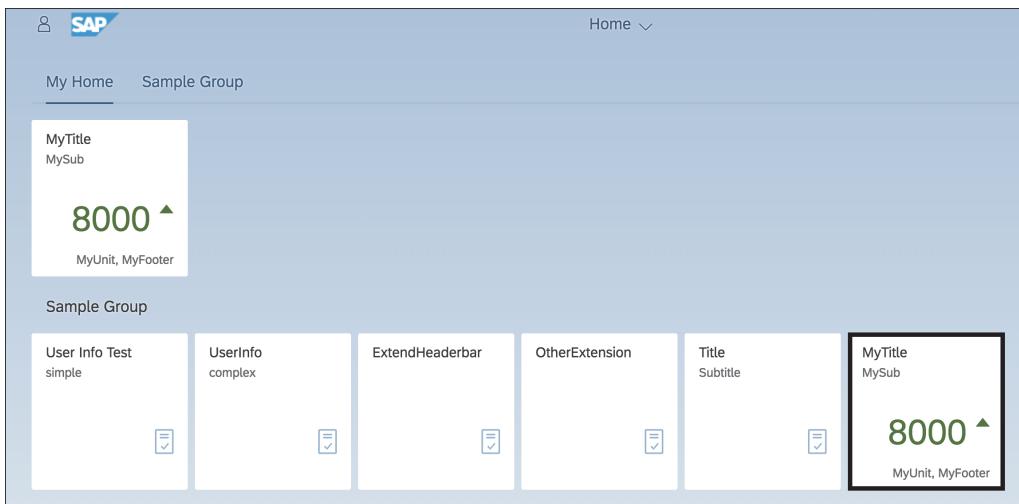


Figure 6.17 Result of Custom Tile

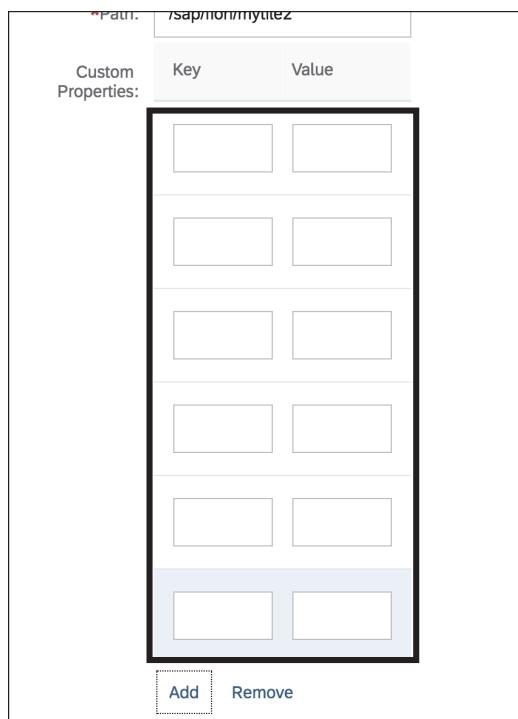
Now you've finished your first tile. However, the tile can't represent the title of your app, and the content is static and can't be changed according to the app. In the next section, you'll learn how to set parameters in the app configuration page and how to add them to your tile.

### 6.2.3 Setting and Parsing Parameters

In this section, you'll learn how to set tile parameters and how to access those parameters in your tile.

First, let's set some parameters, as follows:

1. Open a new browser window, then open the **Administration** page of SAP Cloud Platform Portal.
2. Follow menu path **Content Management • Apps** in the left panel.
3. Locate the app you used in the previous section and click **Edit** to enter edit mode.
4. Switch to the **Visualization** tab.
5. Find the **Add** button and click it five times so that you have six parameters to be filled in (one already exists by default). The result should look like [Figure 6.18](#).



**Figure 6.18** Adding Parameters for Custom Tile

6. You can name and provide values for any number of parameters; for our demo, add the parameters shown in [Table 6.4](#). The result should look like [Figure 6.19](#).

| Name      | Value   |
|-----------|---------|
| subtitle  | demo    |
| number    | 1024    |
| color     | Good    |
| indicator | Down    |
| unit      | USD     |
| footer    | havefun |

**Table 6.4** MyTile: Parameters for Custom Tile

**Tile Properties**

Title Type:

Size:

\*Title:

Module Type:

\*Name:

\*Prefix:

\*Path:

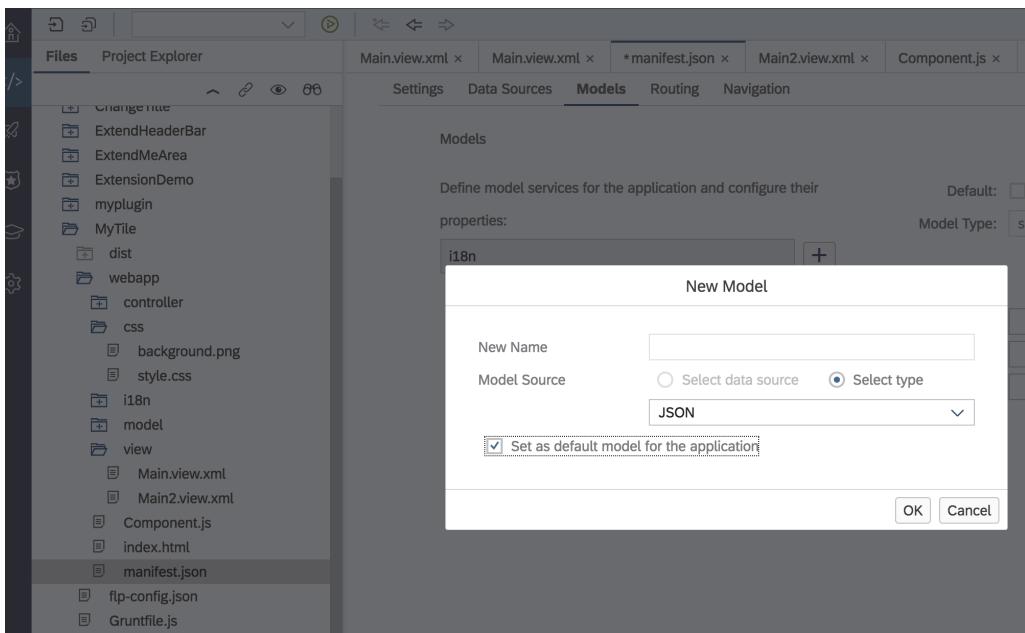
Custom Properties:

| Key       | Value   |
|-----------|---------|
| subtitle  | demo    |
| number    | 1024    |
| color     | Good    |
| indicator | Down    |
| unit      | USD     |
| footer    | havefun |

Add Remove

**Figure 6.19** Result after Setting Parameters

7. Scroll up and change the **Title** to “Custom Tile Demo”.
8. Save your tile.
9. Republish your portal by clicking the globe button, checking the checkbox for **Clear HTML5 Application Cache**, and clicking **Publish**, as was shown in [Figure 6.16](#).
10. Now switch back to SAP Web IDE and continue to edit the source code of your tile.
11. Double-click **webapp • manifest.json** and switch to **Models**.
12. Click the **+** button, and a pop-up window will open.
13. Create a default JSON model, choose the type, and check **Set as Default Model for the Application**. The result should look like [Figure 6.20](#).



**Figure 6.20** Create Default JSON Model

14. Edit the main view using the code editor and change the properties to reflect the data binding as shown in [Listing 6.7](#).

```
<GenericTile header="{/title}" subheader="{/subtitle}" id="tile0">
  <tileContent>
    <TileContent id="content0" footer="{/footer}" unit="{/unit}">
      <NumericContent value="{/number}" valueColor="{/>
```

```
        color}" indicator="/indicator"/>"/>
    </TileContent>
</tileContent>
</GenericTile>
```

**Listing 6.7** MyTile: View with Data Binding to Parameters

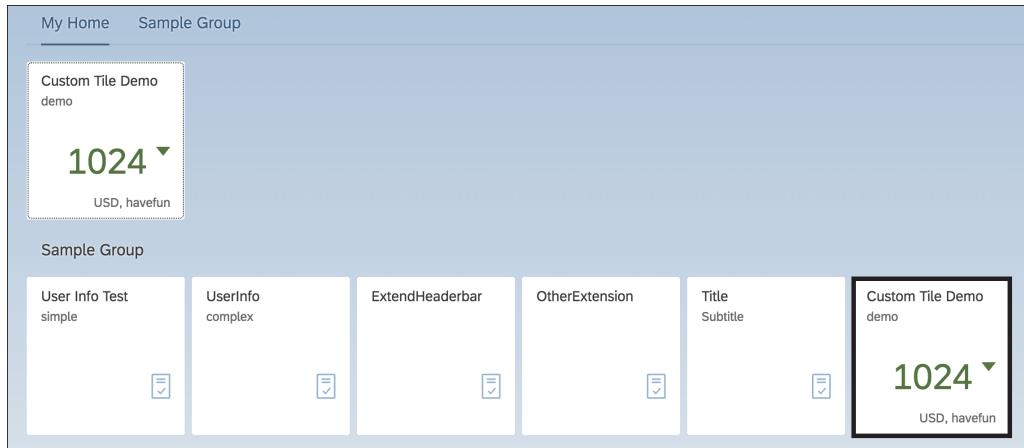
Now that you've bound the view to the JSON model, the next step is to get parameters and use them to fill the JSON model. You can use the `getComponentData` method of `UIComponent` to fetch a JSON object that contains all the customized parameters, as well as predefined parameters such as title. The best place to put this method is in the `onBeforeRendering` hook method of the controller. The tile will be refreshed when a user navigates back from an app.

Edit `Main.controller.js` and add the `onBeforeRendering` method using the code in [Listing 6.8](#).

```
onBeforeRendering:function(){
    //Get reference of the component
    var oComponent = this.getOwnerComponent();
    //Get settings data
    //All custom parameters have same name as you've defined
    //The app title's property name is title
    var oData = oComponent.getComponentData().properties;
    //You can log the data to a console when learning
    //Don't do this in your production code
    console.log(oData);
    //Get the JSON model
    var oModel = oComponent.getModel();
    //Fill model with data
    oModel.setData(oData);
}
```

**Listing 6.8** MyTile: Get Parameters after Tile Is Rendered

After saving all the changes and deploying the app to SAP Cloud Platform again, refresh the SAP Cloud Platform Portal index page to see the results. It should look like [Figure 6.21](#).



**Figure 6.21** Result of Custom Tile with Parameters

It seems everything's finished, but when you click the tile, you'll find that the most important function—jumping to the app—is still missing! You'll implement it using the cross-app navigation service you learned about in [Chapter 3](#).

#### 6.2.4 Implementing Navigation

To implement the navigation logic, you need to invoke the cross-app navigation service when the **press** event of the `GenericTile` control is triggered. And you also need to get the intent information from `componentData`. To begin, follow these steps:

1. Continue editing the `MyTile` project.
2. Open the main view using the layout editor. Select the tile, switch to the **Event** panel, and locate the **Press** event.
3. Create a new function by clicking `:` and selecting **New Function**. Name it “`onTilePress`”.
4. Edit the code by click by clicking `:` and selecting **Open in Editor**.
5. Change the implementation of the `onTilePress` method as shown in [Listing 6.9](#).

```
onTilePress: function (oEvent) {
    //This code was generated by the layout editor.
    //Get component
    var oComponent = this.getOwnerComponent();
    //Get tile configuration data
```

```
var oData = oComponent.getComponentData().properties;
//Get cross-app navigation service
var oService = sap.ushell &&
    sap.ushell.Container &&
    sap.ushell.Container.getService &&
    sap.ushell.Container.getService("CrossApplicationNavi
gation");
//The intent is saved in the intentText property of the config data
oService.toExternal({
    target : { shellHash : oData.intentText }
});
}
```

**Listing 6.9** MyTile: Navigation to Tile Target

6. Save your project and deploy the application to SAP Cloud Platform using the update mode again.
7. Refresh the SAP Cloud Platform Portal index page. This time, if you click the tile, you'll see that the navigation works normally.

### Tips

In this demo, you've set lots of custom parameters. In a real-world case, it's more common to set the URL of an OData service call. Then the tile can get detailed information from the URL. You can use your existing knowledge of OData in SAPUI5 to achieve this easily after finishing this demo.

## 6.3 Deploying a Custom Tile to SAP NetWeaver AS ABAP

In this section, we'll discuss how to deploy a custom tile to the on-premise version of SAP NetWeaver AS ABAP. Because the architecture is different, the requirement for the tile and the deployment steps are different. The tile management is based on the Collaborative Human Interface Part (CHIP) service, which is older than SAP Fiori, so it's more complex and customizable than the options in SAP Cloud Platform.

Using the CHIP service, you can not only define a custom tile but also create a customized configuration page for your tile. We'll guide you through developing and deploying a simple custom tile and then providing a configuration screen.

For this example, you'll develop a custom tile, but this time the tile is in an SAPUI5 view, not a component. You also need to create a CHIP description file to describe the tile.

After deploying this tile, you also need to register this CHIP file in the system. After the tile is generally available, you need to perform tasks such as creating a configuration view and process parameters.

### 6.3.1 Developing a Tile

First let's develop a new tile, following these steps:

1. Create a new SAPUI5 application, using the information shown in [Table 6.5](#).

| Property     | Value              |
|--------------|--------------------|
| Template     | SAPUI5 Application |
| Project Name | MyABAPTile         |
| Namespace    | f1pdev             |
| View Type    | XML                |
| View Name    | Tile               |

**Table 6.5** MyABAPTile: Project Properties

2. Locate the project you created in [Section 6.2.4](#) and open **webapp • view • Main.view.xml** using the code editor. Copy all the source code between the `<View>` and `</View>` tags, without including the tags themselves.
3. Open **webapp • view • Tile.view.xml** in the MyABAPTile project you just created, clear all the code between the `<mvc:View>` and `</mvc:View>` tags, and paste in the code you copied in the previous step. The result should look like [Listing 6.10](#).

```

<mvc:View controllerName="f1pdev.MyABAPTile.controller.Tile" xmlns:html=
"http://www.w3.org/1999/xhtml" xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true" xmlns="sap.m">
<GenericTile header="{/title}" subheader="{/subtitle}" id="tile0" press=
"onTilePress">
    <tileContent>
        <TileContent id="content0" footer="{/footer}" unit="{/unit}">
            <content>

```

```
        <NumericContent value="/number" valueColor="/color" indicator="/indicator"/>
    </content>
    </TileContent>
</tileContent>
</GenericTile>
</mvc:View>
```

**Listing 6.10** MyABAPTile: XML Code for Tile View

4. Edit **controller•Tile.controller.js**. In the `onInit` method, create an OData model and set it as the default model of the view. Then create a new method called `_setPreviewData`, which creates a JSON object with default values, and use this object to fill the default model. Also, you need to call `_setPreviewData` in the `onInit` method. [Listing 6.11](#) shows the code you'll use to do so.

```
onInit:function(){
    //Create a JSON model
    this._oTileModel = new sap.ui.model.json.JSONModel();
    //Set model
    this.getView().setModel(this._oTileModel);
    //Prepare dummy data
    this._setPreviewData();
},
_setPreviewData:function(){
    //A JSON object with a default value
    var oData = {
        title:"Custom Tile",
        subtitle:"Subtitle",
        footer:"Footer",
        unit:"Unit",
        number:"99",
        color:"Good",
        indicator:"Up"
    };
    //Set the default data
    this._oTileModel.setData(oData);
}
```

**Listing 6.11** MyABAPTile: Set Default Data in Controller

5. Run a preview of your tile, and the result should look like [Figure 6.22](#).

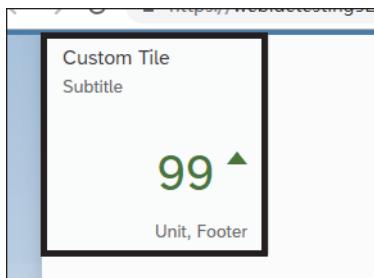


Figure 6.22 Initial State of Custom Tile for ABAP

### 6.3.2 Creating a CHIP Description File

A *CHIP* is an encapsulated, stateful piece of software used to provide functions in collaboration with other CHIPS in a *page builder* page or side panel. All available CHIPS are registered in a library (the CHIP catalog). The CHIP model describes the capabilities of a CHIP and is not based on a specific UI technology. *Client-side-rendering CHIPS* (CSR CHIPS) are CHIPS that use client-side-rendering capabilities, such as SAPUI5.

To register your tile to SAP Fiori launchpad, you should create a CHIP file manually, following these instructions:

1. Right-click the **webapp** folder and choose **Import • File or Project**, as shown in [Figure 6.23](#).

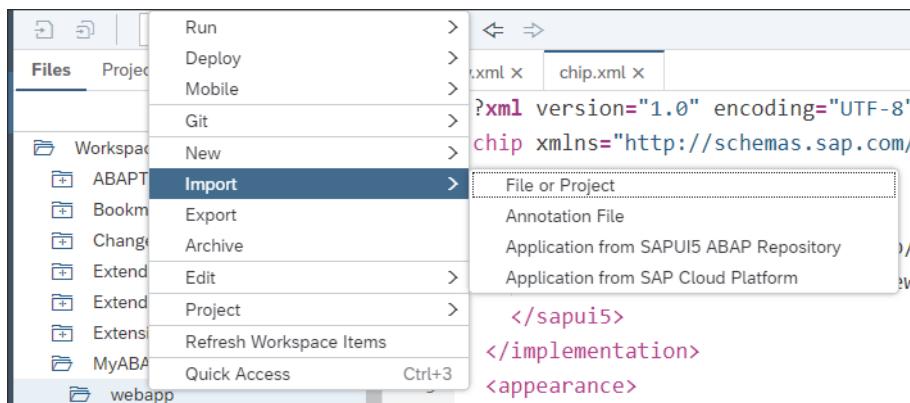


Figure 6.23 Import File

2. Navigate to the location where you saved and extracted the sample files for this chapter (downloadable at [www.sap-press.com/4556](http://www.sap-press.com/4556)), select **chip.xml**, and click **OK**.

3. The *chip.xml* file is imported to your **webapp** folder. It can serve as a template for your custom tiles.
4. Find the `<basePath>` and `<viewName>` tags under `<chip> • <implementation> • <sapui5>`. The `<basePath>` should be the path to your ABAP server where all SAPUI5 applications will be deployed. Change this value to `/sap/bc/ui5_ui5/sap/`.
5. For the view name, use `zmytile/view.Tile.view.xml`. The result should like [Listing 6.12](#).

```
<implementation>
  <sapui5>
    <basePath>/sap/bc/ui5_ui5/sap/</basePath>
    <viewName>zmytile/view.Tile.view.xml</viewName>
  </sapui5>
</implementation>
```

**Listing 6.12** MyABAPTile: CHIP Configuration

#### Note

Let's pause for a moment to explain why you write configuration files like these. If you have experience with SAP Fiori launchpad configuration, you should know that the namespace of a view is not relative to the physical path of your SAPUI5 application—but that's not true for your custom tile. The namespace is resolved only by the physical path, so all you need to provide is the physical path you've planned for this tile. In this case, you must make sure that ZTILE is your ABAP BSP application name when you deploy it.

6. Change the value of `<appearance> • <title>` to My Custom Tile. It will appear as follows in SAP Fiori launchpad designer:

```
<appearance>
  <title>My Custom Tile</title>
</appearance>
```

7. Change the value of the `/UI2/ChipType` parameter to MYCHIP under the last `<parameter>` tag. This value is used when you register your chip file in the SAP NetWeaver AS ABAP system, as follows:

```
<parameters>
  <parameter name="/UI2/ChipType">MYCHIP</parameter>
</parameters>
```

8. Keep all other content unchanged and save the *chip.xml* file.

**Note**

CHIP technology is used in various technologies other than SAP Fiori launchpad, so there will be lots of other configurations available that you don't need for general purposes. If you want to fully understand how CHIPS are configured, visit [https://help.sap.com/viewer/product/SAP\\_NETWEAVER\\_AS\\_ABAP\\_752/7.52.1/en-US](https://help.sap.com/viewer/product/SAP_NETWEAVER_AS_ABAP_752/7.52.1/en-US) and go to **Development Information • SAP NetWeaver User Interface Services Developer Guide**. In this guide, expand **SAP NetWeaver User Interface Services Developer Guide • JavaScript Services • CHIP Development**.

### 6.3.3 Deploying Your Tile as an SAPUI5 Application

Now it's time to deploy the tile. Before doing so, make sure your SAP NetWeaver AS ABAP system and SAP Fiori launchpad are working. You also need to connect SAP Web IDE to the SAP NetWeaver AS ABAP system using SAP Cloud Connector. If you're not familiar with those steps, refer back to [Chapter 2](#).

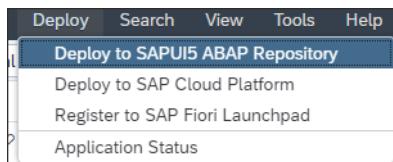
**Note**

To make your app work with the tiles' target resolution mechanism, you need to change all namespaces to `zmytile`, the same way you do with your ABAP BSP application name. The first change is in `view/Tile.view.xml`, where you'll change the `controllerName` property of the view to `zmytile.controller.Tile`. The second change is in `controller/Tile.controller.js`, where you'll change the namespace after `Controller.extend` to `zmytile.controller.Tile`.

The reason we do this after creation is that the SAP Web IDE full stack version does not support creating a project without a namespace; it is not possible to generate an initial project with a namespace like `zmytile.controller.Tile`.

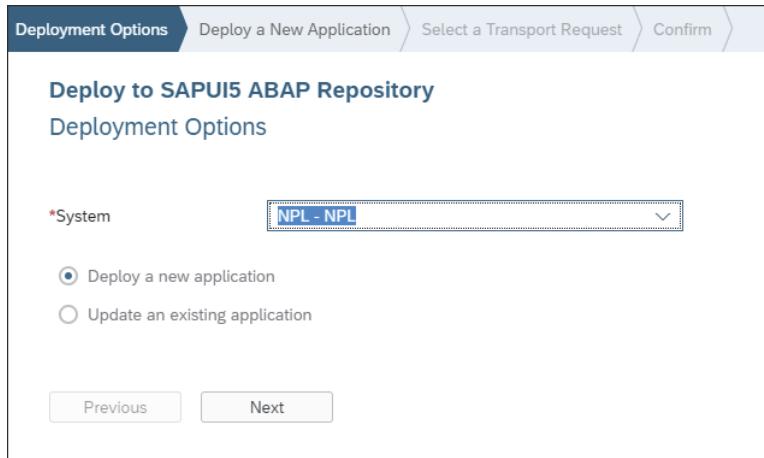
To deploy your tile, follow these steps:

1. Deploy your tile to the SAP NetWeaver AS ABAP system by choosing your app and following menu path **Deploy • Deploy to SAPUI5 ABAP Repository**, as shown in [Figure 6.24](#).



**Figure 6.24** Deploy App to ABAP Repository

2. Select your system—in our case, **NPL - NPL**—and then click **Next**, as shown in [Figure 6.25](#).



**Figure 6.25** Choose Backend System

3. Enter “ZMYTILE” for **Name**, type a **Description**, and select **\$TMP** for the **Package** via the **Browse** button. Note that the search help in the package selection can sometimes take more than one minute. The result should look like [Figure 6.26](#).

The screenshot shows the 'Deploy a New Application' step of the deployment wizard. The title is 'Deploy to SAPUI5 ABAP Repository' and the sub-section is 'Deploy a New Application'. It asks to 'Select a package for the application'. There are three input fields: 'Name' (ZMYTILE), 'Description' (My tile), and 'Package' (\$TMP). A 'Browse' button is available for the package field. At the bottom are 'Previous' and 'Next' buttons.

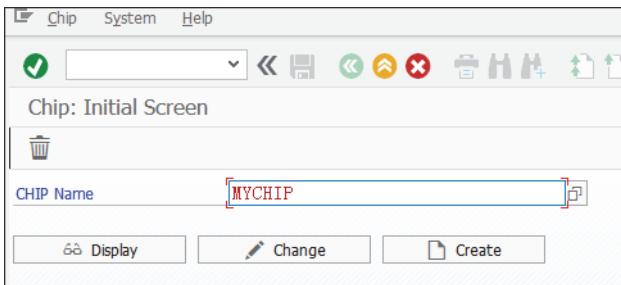
**Figure 6.26** Deployment Information

4. Click **Next**, then click **Finish**. Wait for about one minute, and your app will be deployed on your SAP NetWeaver AS ABAP system.

### 6.3.4 Registering Your Tile

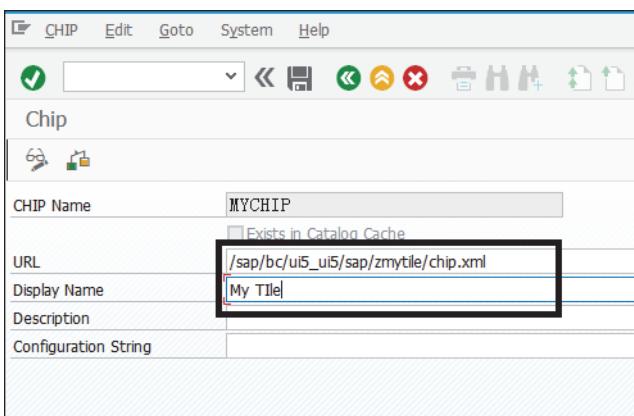
The next step is to make your tile accessible in SAP Fiori launchpad designer. First you need to register your app as a CHIP, then you should add the CHIP to SAP Fiori launchpad, as follows:

1. Logon to your SAP NetWeaver AS ABAP system using SAP GUI.
2. Enter Transaction UI2/CHIP.
3. On the resulting screen, for **CHIP Name**, enter “MYCHIP”, then click **Create**, as shown in [Figure 6.27](#).



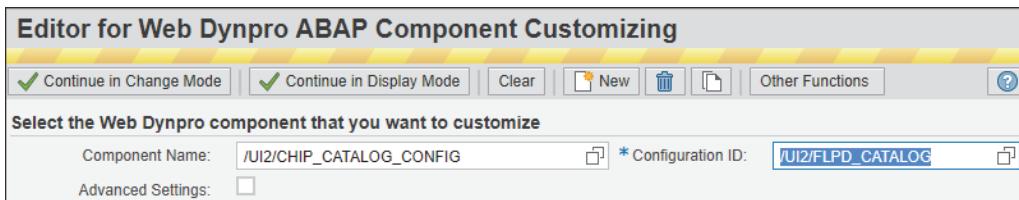
[Figure 6.27 Create CHIP](#)

4. For **URL**, enter “/sap/bc/ui5\_u5/sap/zmytile/chip.xml”. Fill in the **Display Name** and **Description** as desired—for example, “My Tile” for **Display Name**. Then click the **Save** button to save your configuration. The screen should appear as shown in [Figure 6.28](#).



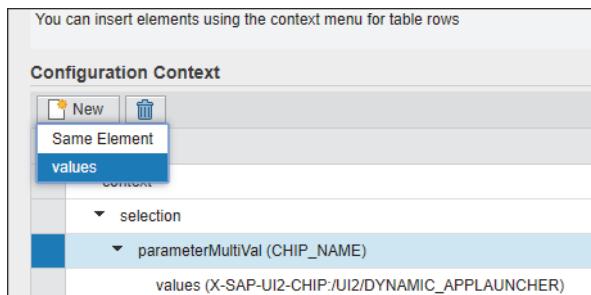
[Figure 6.28 CHIP Details](#)

5. Click **Local Object** or save it in a package.
6. Open your browser and access URL `http://vhcalnplci.dummy.nodomain:8000/sap/bc/webdynpro/sap/customize_component`. If you're using a different system, change the domain and the port accordingly.
7. Provide your name and password and log on to the system.
8. For the **Component Name**, select `/UI2/CHIP_CATALOG_CONFIG` from the value help.
9. For **Configuration ID**, if it's your first time configuring this, enter "`/UI2/FLPD_CATALOG`" and click **New**. If it is not, choose `/UI2/FLPD_CATALOG` from the search help and click **Continue in Change Mode** (see [Figure 6.29](#)).



**Figure 6.29** Web Dynpro Application for Configuring CHIP

10. Click the **parameterMultiVal(CHIP\_NAME)** node, then click **New**, then choose **values** from the drop-down menu, as shown in [Figure 6.30](#).



**Figure 6.30** Adding CHIPs

11. For the **Parameter Value** field, enter "X-SAP-UI2-CHIP:MYCHIP", as shown in [Figure 6.31](#). The first part is static; the second part should be the same as the parameter you set in *chip.xml*.



Figure 6.31 Set CHIP Value

12. Save and close the page.
13. Now enter SAP Fiori launchpad designer, which you bookmarked in [Chapter 2](#), and enter your user credentials.
14. Click + in the bottom-left corner and create a catalog with following information:
  - Title: “Tile Test”
  - ID: “TILETEST”
15. Then click **Save**, as shown in [Figure 6.32](#).

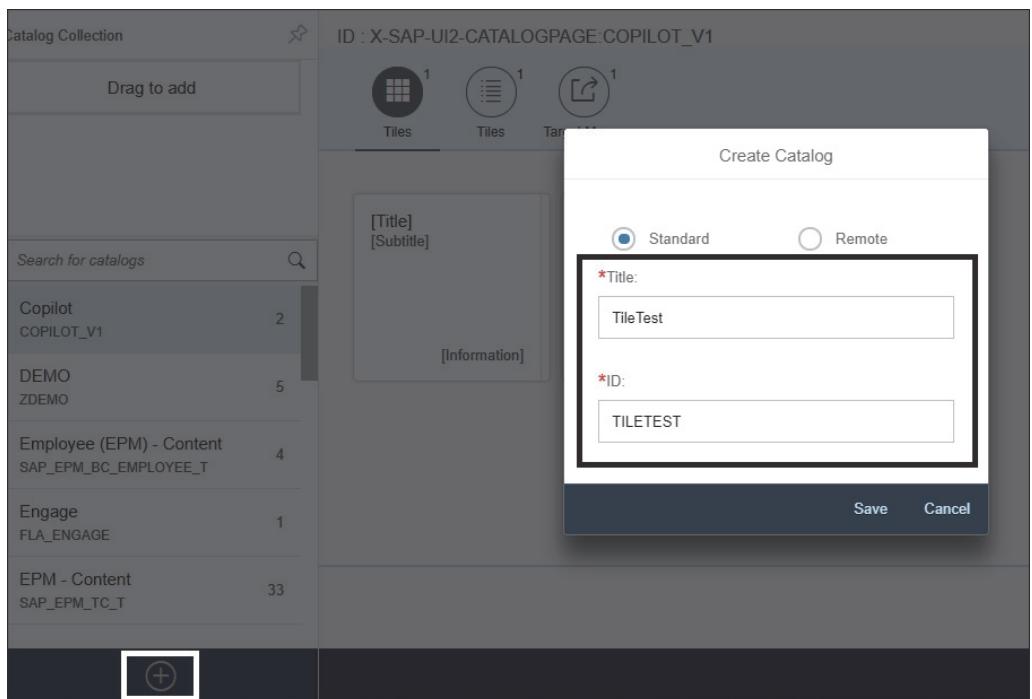
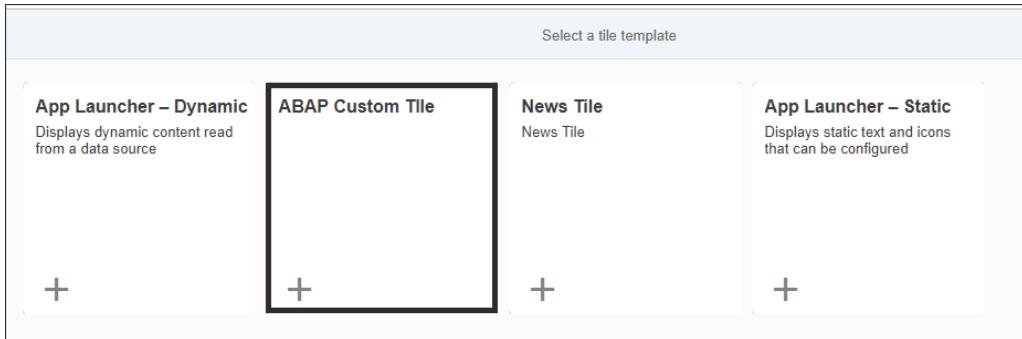


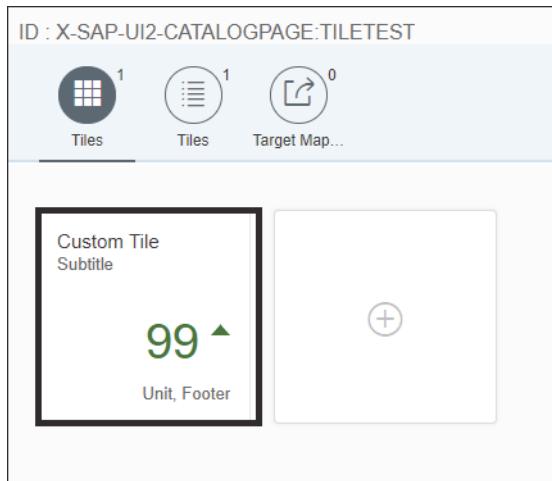
Figure 6.32 Add Tile Catalog

16. Click the + tile on the right side of your window. You'll see a list of available tile types, as shown in [Figure 6.33](#).



**Figure 6.33** List of Available Tile Types

17. Click your custom tile type, and a new tile will appear on the screen with the dummy data you created, as shown in [Figure 6.34](#).



**Figure 6.34** Custom Tile Added

### 6.3.5 Creating a Configuration Screen

As you know, a tile should display information from its configuration. Unlike SAP Cloud Platform Portal, the ABAP platform doesn't provide a default UI for adding

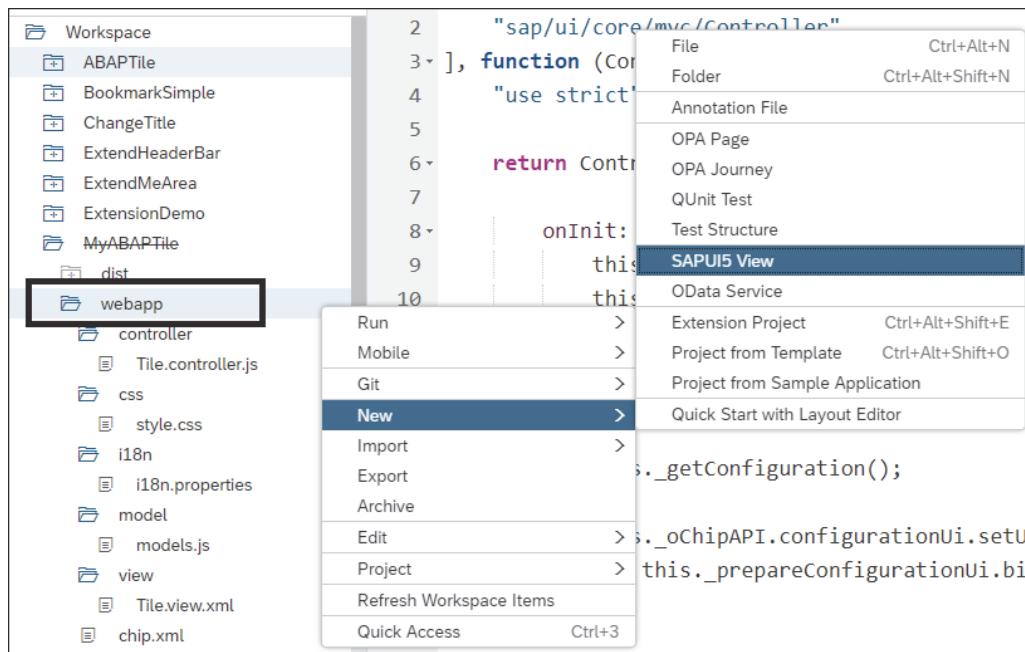
parameters. Instead, you need to provide a configuration screen yourself. This enables much deeper customizations for tiles.

In this section, you'll create a configuration screen using a simple form. We'll avoid creating a complex screen to help you focus on the topics of this chapter. There are three prerequisites you must meet before you begin:

1. A prerequisite for a configuration screen is you should have an SAPUI5 view screen.
2. You should declare your tile support configuration in chip.xml, which we've already done in the template.
3. You need to access the CHIP API and tell it where to find the configuration UI.

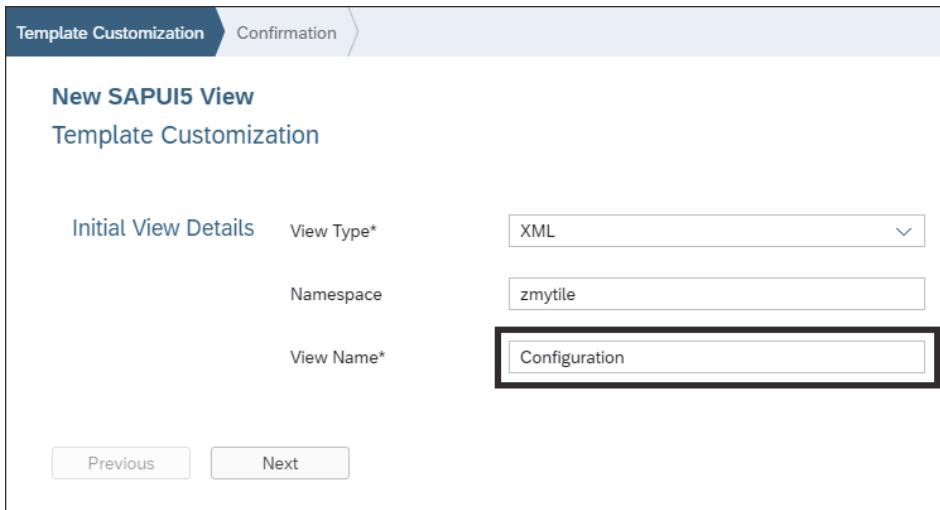
To begin creating a configuration screen, follow these steps:

1. Open SAP Web IDE, right-click the **webapp** folder of your project, and choose **New • SAPUI5 View**, as shown in [Figure 6.35](#).



**Figure 6.35** Menu for Creating New View

2. On the **New SAPUI5 View Template Customization** screen shown in [Figure 6.36](#), leave **View Type** and **Namespace** set to their default values. For **View Name**, enter “Configuration”, then click **Next**.



**Figure 6.36** Setting View Information

3. Click **Finish**.
4. Open the new view with the layout editor.
5. Delete all content from the default view, which you can do by switching to the **Outline** tab, right-clicking the **App** node to open the context menu, and selecting **Delete**.
6. Create a form, using a simple form control and the fields and their binding properties, as shown in [Table 6.6](#). For simplicity’s sake, we used input controls for all fields (using the same procedure detailed in [Chapter 4](#)).

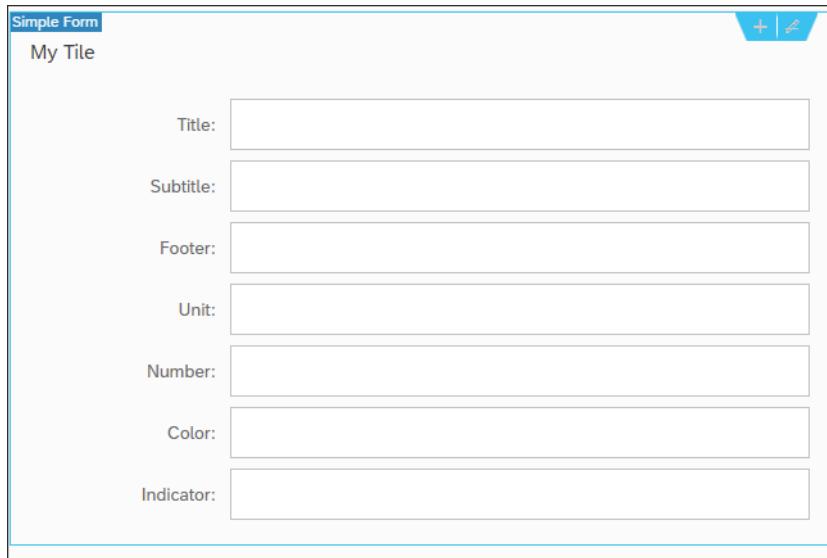
| Sequence | Field    | Binding Syntax |
|----------|----------|----------------|
| 1        | Title    | {/title}       |
| 2        | Subtitle | {/subtitle}    |
| 3        | Footer   | {/footer}      |
| 4        | Unit     | {/unit}        |

**Table 6.6** MyABAPTile: Form Fields in Configuration View

| Sequence | Field     | Binding Syntax |
|----------|-----------|----------------|
| 5        | Number    | {/number}      |
| 6        | Color     | {/color}       |
| 7        | Indicator | {/indicator}   |

**Table 6.6** MyABAPTile: Form Fields in Configuration View (Cont.)

7. The view should look like [Figure 6.37](#).

**Figure 6.37** Preview for Configuration View

8. Next, when your view is loading, you need to tell the CHIP API where to find the configuration view. Modify the `onInit` method of `Tile.controller.js` as shown in [Listing 6.13](#).

```
onInit:function(){
  //Existing code
  //Create a JSON model
  this._oTileModel = new sap.ui.model.json.JSONModel();
  //Set model
  this.getView().setModel(this._oTileModel);
```

```
//Prepare dummy data
this._setPreviewData();

//Code added in this step
//Get API object for CHIP
this._oChipAPI = this.getView().getViewData().chip;
//Provide a function that returns a view object
//Method _prepareConfigurationUi will be mentioned later
this._oChipAPI.configurationUi.setUiProvider(
    this._prepareConfigurationUi.bind(this)
);
},
```

**Listing 6.13** MyABAPTile: Set Configuration Screen when Tile Is Initialing

9. Then you will implement the `_prepareConfigurationUi` method to load the configuration view. The code should be as shown in [Listing 6.14](#).

```
_prepareConfigurationUi: function () {
    //Load the XML view as an object
    var oConfigView = sap.ui.xmlview({
        viewName: "zmytile.view.Configuration"
    });
    oConfigView.setModel(this._oTitleModel)
    //Return the view
    return oConfigView;
}
```

**Listing 6.14** MyABAPTile: Loading and Returning Configuration UI

10. Now you can deploy your SAPUI5 app again using the **Deploy** menu. Again, before you do so, you'll need to change your namespaces to `zmytile`.
11. You may need to run report `/UI2/INVALIDATE_CLIENT_CACHES` to clear caches.
12. Enter SAP Fiori launchpad designer again, find your tile catalog, `TileTest`, and click it. Then click the second icon from the left in the tab bar to show all tiles as a list, as shown in [Figure 6.38](#).
13. Select your custom tile and click **Configure**. The configuration view will display, as shown in [Figure 6.39](#). You'll notice that a page tile and footer bar with **Save** and **Cancel** buttons has already been generated for you.

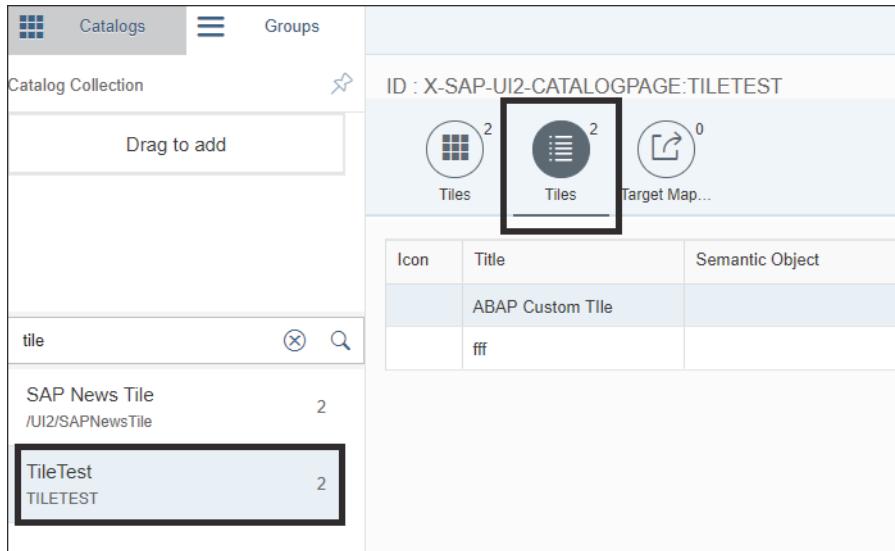


Figure 6.38 Tile List as Table

The screenshot shows a configuration screen for the 'ABAP Custom Tile'. At the top, there's a back arrow, the title 'Configure: 'ABAP Custom Tile'', and an instance ID '0H1UV5UVM1N14IYKON9NX86UJ'. The main area is titled 'My Tile' and contains several input fields: 'Title:' (with a placeholder box), 'Subtitle:' (empty), 'Footer:' (empty), 'Unit:' (empty), 'Number:' (empty), 'Color:' (empty), and 'Indicator:' (empty). At the bottom right, there are 'Save' and 'Cancel' buttons.

Figure 6.39 Configuration Screen

Congratulations! Now you've built your first configuration screen.

### 6.3.6 Setting and Getting Parameters

In this section, we'll make the configuration screen work. We do so using parameters, the storage of which can be handled by the CHIP service. To begin, follow these steps:

1. Return to SAP Web IDE and open *chip.xml* in your project. Add the code in [Listing 6.15](#) between `<consume id="configuration">` and `</consume>`. This code defines seven custom parameters and one default parameter required by the runtime, with `true` as the default value.

```
<parameter name="title"></parameter>
<parameter name="subtitle"></parameter>
<parameter name="footer"></parameter>
<parameter name="footer"></parameter>
<parameter name="unit"></parameter>
<parameter name="number"></parameter>
<parameter name="color"></parameter>
<parameter name="indicator"></parameter>
<parameter name="sap.ui2.launchpage.support">true</parameter>
```

**Listing 6.15** MyABAPTile: Configuration Parameters in CHIP

2. Next, modify the `onInit` method of *Tile.controller.js*, as shown in [Listing 6.16](#).

```
onInit:function(){
    //Existing code
    //Create a JSON model
    this._oTileModel = new sap.ui.model.json.JSONModel();
    //Set model
    this.getView().setModel(this._oTileModel);
    //Comment this method
    //this._setPreviewData();

    //Existing code
    //Get API object for CHIP
    this._oChipAPI = this.getView().getViewData().chip;
    //Provide a function that returns a view object
    //Method _prepareConfigurationUi will be mentioned later
    this._oChipAPI.configurationUi.setUiProvider(
        this._prepareConfigurationUi.bind(this)
    );
```

```
//New code for this demo
//Get existing configuration data before load
this._getConfiguration();
//New code for this demo
//When user clicks save, method this._
saveConfiguration will be invoked
    this._oChipAPI.configurationUi.attachSave(
        this._saveConfiguration.bind(this)
    );

},
```

**Listing 6.16** MyABAPTile: Setting Save and Fetch Actions when Initialzing

3. Create the `_getConfiguration` method to get configuration data from the CHIP API and write it into your JSON model using the code in [Listing 6.17](#).

```
_getConfiguration: function () {
    var confAPI = this._oChipAPI.configuration;
    var oData = {
        title: confAPI.getParameterValueAsString("title") || "",
        subtitle: confAPI.getParameterValueAsString("subtitle") || "",
        footer: confAPI.getParameterValueAsString("footer") || "",
        unit: confAPI.getParameterValueAsString("unit") || "",
        number: confAPI.getParameterValueAsString("number") || "0",
        color: confAPI.getParameterValueAsString("color") || "",
        indicator: confAPI.getParameterValueAsString("indicator") || ""
    };
    this._oTileModel.setData(oData);
}
```

**Listing 6.17** MyABAPTile: Fetching Configuration Data

4. Create the `_saveConfiguration` method to save your settings using the code in [Listing 6.18](#).

```
_saveConfiguration: function () {
    var writeConfAPI = this._oChipAPI.writeConfiguration;
    var oData = this._oTileModel.getData();
    writeConfAPI.setParameterValues(oData, function () {
        //Execute when successful
    }, function () {
```

```
//Execute when failed  
});  
}  
});
```

**Listing 6.18** MyABAPTile: Saving Configuration Data

5. Save the controller.

### Tip

Don't forget to set binding parameters to the configuration view to make sure all input fields are bound to corresponding fields in the JSON model. Setting these bindings will follow the same steps as those in [Section 6.2](#).

6. Deploy your tile again and test it by entering the configuration page. Input some values for the parameters and save. Then refresh the page, at which point you'll see that your settings are in place.

### Best Practices when Creating a Custom Tile

Now you've created a basic custom tile and, most importantly, you know how custom tiles work. But as an experienced developer, you'll find there are still missing details—for example, setting semantic objects and actions.

In general, a custom tile is useful when you want to add some specific function to a standard tile provided by SAP. The best practice is to create a copy of SAP's code and change it. After you complete the demos in this book, you'll know how this works and will have the ability to understand and change the code.

You can download SAP's code from <https://ui5.sap.com>; on that page, click the big **Download** button. After downloading the code, you can find source code for tiles at `sapui5-rt-1.56.5\resources\sap\ushell\components\tiles` on your local machine. The folder structure should look like [Figure 6.40](#).

The `utils.js` file provides some reusable APIs to access the CHIP service; each type of tile is a folder. You can create a new tile based on existing code and deploy it as you've learned to do in this book.

| 1.56.5 > resources > sap > ushell > components > tiles |                 |       |        |
|--|-----------------|-------|--------|
| 名称   | 修改日期            | 类型    | 大小     |
| action   | 2018/8/1 9:04   | 文件夹   |        |
| applauncher  | 2018/8/1 9:04   | 文件夹   |        |
| applaucherdynamic                                      | 2018/8/1 9:04   | 文件夹   |        |
| cdm  | 2018/8/1 9:04   | 文件夹   |        |
| indicatorArea  | 2018/8/1 9:04   | 文件夹   |        |
| indicatorcomparison                                    | 2018/8/1 9:04   | 文件夹   |        |
| indicatorcontribution                                  | 2018/8/1 9:04   | 文件夹   |        |
| indicatordeviation                                     | 2018/8/1 9:04   | 文件夹   |        |
| indicatorDual  | 2018/8/1 9:04   | 文件夹   |        |
| indicatorDualComparison                                | 2018/8/1 9:04   | 文件夹   |        |
| indicatorDualContribution                              | 2018/8/1 9:04   | 文件夹   |        |
| indicatorDualDeviation                                 | 2018/8/1 9:04   | 文件夹   |        |
| indicatorDualTrend                                     | 2018/8/1 9:04   | 文件夹   |        |
| indicatorHarveyBall                                    | 2018/8/1 9:04   | 文件夹   |        |
| indicatornumeric                                       | 2018/8/1 9:04   | 文件夹   |        |
| indicatorTileUtils                                     | 2018/8/1 9:04   | 文件夹   |        |
| generic.js   | 2018/7/30 20:47 | JS 文件 | 29 KB  |
| generic-dbg.js   | 2018/7/30 20:44 | JS 文件 | 47 KB  |
| sbtilecontent.js                                       | 2018/7/30 20:47 | JS 文件 | 2 KB   |
| sbtilecontent-dbg.js                                   | 2018/7/30 20:44 | JS 文件 | 3 KB   |
| utils.js   | 2018/7/30 20:47 | JS 文件 | 45 KB  |
| utils-dbg.js   | 2018/7/30 20:44 | JS 文件 | 112 KB |
| utilsRT.js   | 2018/7/30 20:47 | JS 文件 | 8 KB   |
| utilsRT-dbg.js   | 2018/7/30 20:44 | JS 文件 | 21 KB  |

Figure 6.40 Folder Structure for Tiles

## 6.4 Summary

In this chapter, you learned how to develop your own tile types and integrate them with SAP Fiori launchpad. As an SAPUI5 developer, you've probably experienced cases in which the standard tile types provided by SAP Fiori launchpad can't satisfy your needs. Simply developing your own dashboard page can let you customize your tiles, but they are not easy to manage and don't support standard methods of configuration and personalization. The best part of the techniques mentioned in this chapter is that you can make your tiles customizable and easy to manage as well.

In the next chapter, you'll learn about code-level extensibility features for SAP Fiori launchpad—that is, plug-ins. Using plug-ins, you can run your custom code just after SAP Fiori launchpad is initialized.



# Chapter 7

## Plug-Ins

*Plug-ins help you run custom code just after SAP Fiori Launchpad has been initialized. In this chapter, you'll learn how to develop a plug-in and deploy it on both SAP Cloud Platform and SAP NetWeaver AS ABAP.*

*Plug-ins* are SAPUI5 components without a user interface. They use a special intent, such as *shell plugin*, when deploying on SAP Fiori launchpad. All plug-ins will be loaded, and the initialize code in those plug-ins will be executed after SAP Fiori launchpad has been loaded.

A plug-in is a major container for code that changes SAP Fiori launchpad globally. Plug-ins have the following characteristics:

- Plug-ins are automatically loaded and initialized when SAP Fiori launchpad is started.
- Plug-ins are implemented as SAPUI5 components and provide all standard deployment and lifecycle features of SAPUI5.
- Plug-ins will always be implemented in a platform-independent way, but platform-specific configuration is allowed. On SAP Cloud Platform, you need to configure plug-ins on SAP Cloud Platform Portal. On SAP NetWeaver AS ABAP, you configure them via SAP Fiori launchpad designer.
- On SAP NetWeaver AS ABAP, plug-ins can be enabled and configured dynamically by assigning users to roles.

In this chapter, we'll walk you through developing a plug-in and then deploying it on SAP Cloud Platform or SAP NetWeaver AS ABAP. We'll end the chapter with a quick look at the predefined plug-ins provided by SAP.

### 7.1 Developing a Plug-In

This section will walk you through developing a plug-in, from using templates and adjusting your code through testing the plug-in to ensure it's working properly.

### 7.1.1 Creating a Plug-In Using a Template

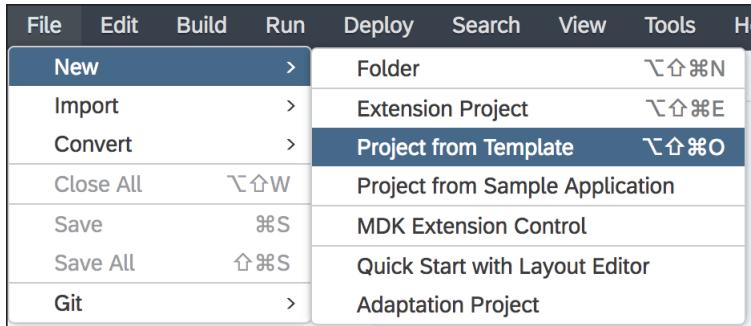
The easiest way to create a new plug-in is by using the template provided by SAP Web IDE full-stack version. Because it's normally not recommended to write complex code in a plug-in, you can always create plug-ins based on the template.

The template not only contains the basic structure and mandatory code for initializing the component but also can help you generate sample code to perform the following activities:

- Adding a button to the SAP Fiori launchpad header
- Adding an SAP Fiori launchpad footer with a button
- Adding buttons to the Me Area

The following procedure will guide you through the basic process of creating a plug-in:

1. Enter SAP Web IDE full-stack version.
2. Follow menu path **File • New • Project from Template**, as shown in [Figure 7.1](#).



**Figure 7.1** Create Project from Template

3. In the **Template Selection** window, select **SAP Fiori Launchpad Plug-In** and click **Next**, as shown in [Figure 7.2](#).
4. In the **Basic Information** window, enter “FlpPlugin” for the **Project Name** and then click **Next**, as shown in [Figure 7.3](#).

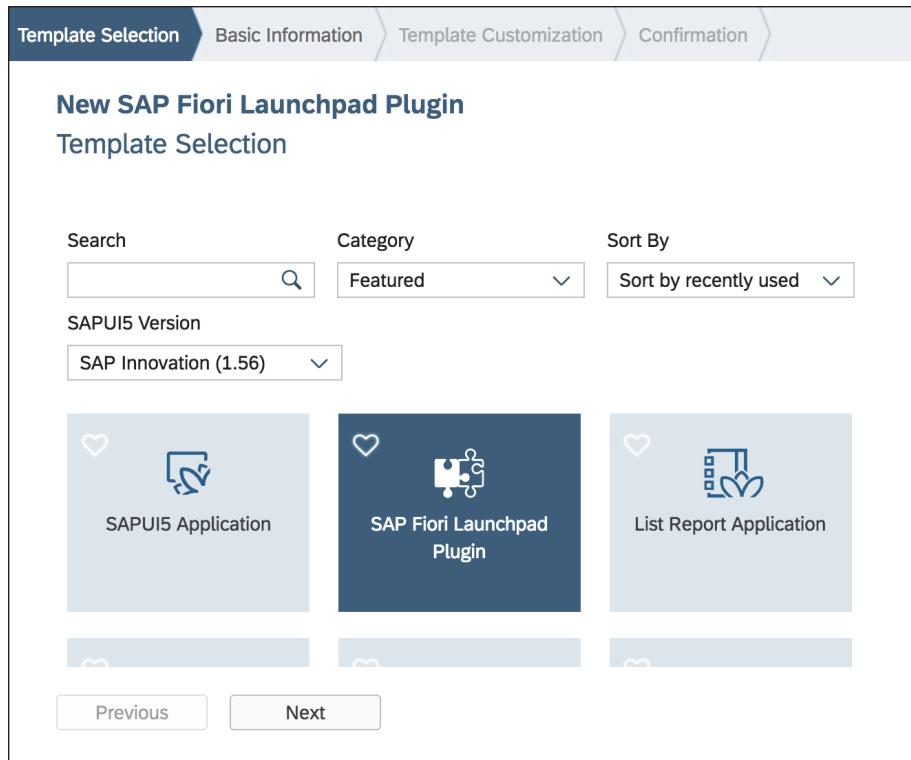


Figure 7.2 Template Selection

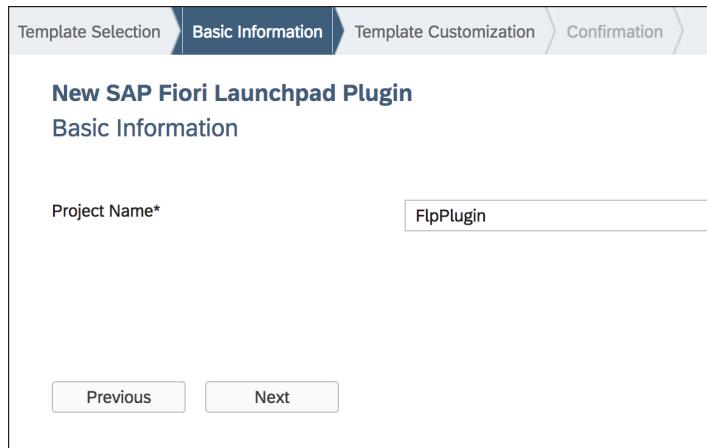
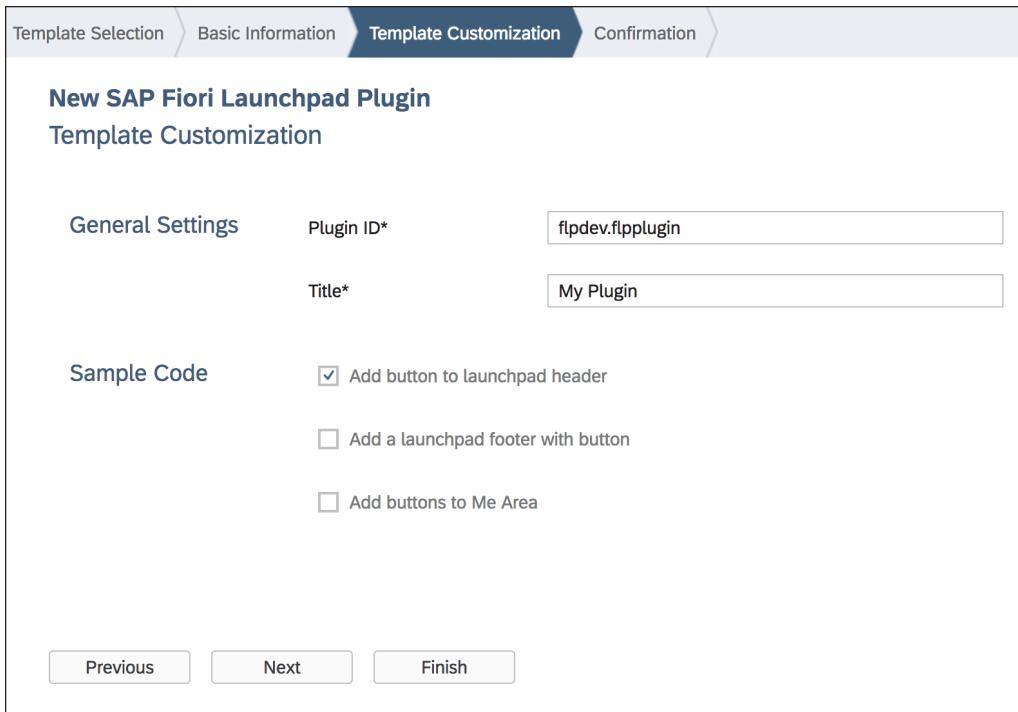


Figure 7.3 Basic Information

5. In the **Template Customization** window, enter “flpdev.flppugin” for **Plugin ID** and any text of your choice for **Title**. Here you can choose the **Sample Code** you need. For this example, select **Add Button to Launchpad Header**, then click **Finish**, as shown in [Figure 7.4](#).



**Figure 7.4** Template Customization

Now that you've created your first plug-in, let's look at the generated code (see [Listing 7.1](#)). The most important file for the plugin is *component.js*.

```
sap.ui.define([
    "sap/ui/core/Component",
    "sap/m/Button",
    "sap/m/Bar",
    "sap/m/MessageToast"
], function (Component, Button, Bar, MessageToast) {

    return Component.extend("flpdev.flppugin.Component", {
```

```
metadata: {
    "manifest": "json"
},

init: function () {
    var rendererPromise = this._getRenderer();

    // This is example code. Please replace with your implementation!

    /**
     * Add item to the header
     */
    rendererPromise.then(function (oRenderer) {
        // Here is the place for your custom code

        oRenderer.addHeaderItem({
            icon: "sap-icon://add",
            tooltip: "Add bookmark",
            press: function () {
                MessageToast.show("This SAP Fiori launchpad has been e
xtended to improve your experience");
            }
        }, true, true);
    });

    /**
     * This method help you get a deferred object which will return the rend
er object
     */
    _getRenderer: function () {
        //You can just ignore the generated code, just use it!
    }
});
```

**Listing 7.1** FlpPlugin: Generated Component

Note the following elements of this file:

- The component is a subclass of `sap.ui.core.Component`. In contrast, other SAPUI5 applications are subclasses of `sap.ui.core.UIComponent`. A plug-in will never have a view or controllers; it inherits from `component` so that the request payload can be minimized.
- A `_getRenderer` method is generated for you. The most common usage for a plug-in is to extend SAP Fiori launchpad, so the renderer is the most used object. Unlike normal applications, a plug-in is executed during the startup of SAP Fiori launchpad, so it's not certain that the renderer object has been loaded when the plug-in is executing. The `_getRenderer` method returns a deferred object that helps you make sure your code is executed after the renderer has been loaded.
- In the `init` method, some example code has been generated; all your code that accesses the SAP Fiori launchpad should be placed in the functions after resolving the `rendererPromise` object.

### 7.1.2 Adjusting Implementation Code

Now let's focus on the core of the code (see [Listing 7.2](#)), in which you can find in `init` method adds a header item.

```
oRenderer.addHeaderItem({  
    icon: "sap-icon://add",  
    tooltip: "Add bookmark",  
    press: function () {  
        MessageToast.show("This SAP Fiori launchpad has been extended to i  
mprove your experience");  
    }  
}, true, true);
```

**Listing 7.2** FlpPlugin: Code Generated for Adding Header Item

As you've already seen, it's not easy to test this because the header item will not display if there isn't enough space. Let's change the code where you found in `init` method to add a header end item, as shown in [Listing 7.3](#).

```
oRenderer.addHeaderEndItem(  
    "sap.ushell.ui.shell.ShellHeadItem", {  
        id: "addbookmarkitem",
```

```

    icon: "sap-icon://add",
    tooltip: "Add bookmark",
    press: function () {
        MessageToast.show("This SAP Fiori launchpad has been extended to i
mprove your experience");
    }
}, true, true);

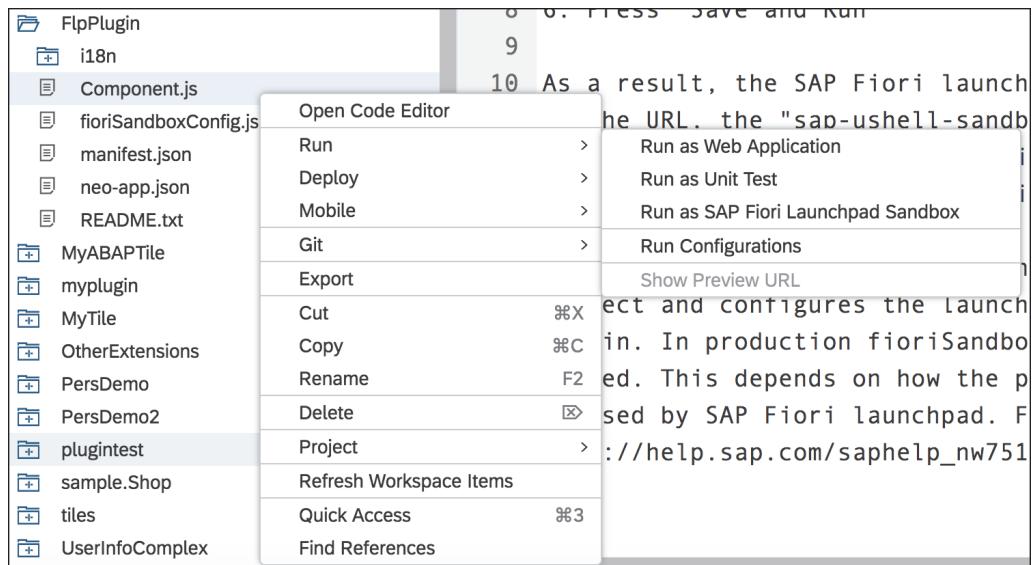
```

**Listing 7.3** FlpPlugin: Add Header End Item

### 7.1.3 Testing Your Plug-In

Testing of a plug-in is a little different compared to testing a normal SAPUI5 application. The following procedure will guide you through the testing process:

1. Right-click **Component.js** and choose **Run • Run Configurations** in the context menu, as shown in [Figure 7.5](#).



**Figure 7.5** Run Configuration

2. Click the **Plus** button and choose **Run as SAP Fiori Launchpad Sandbox** in the popover, as shown in [Figure 7.6](#).

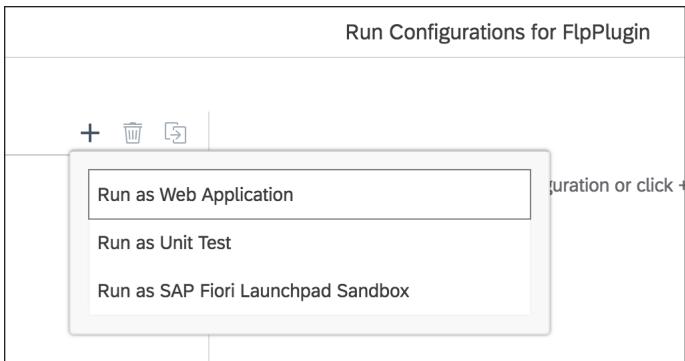


Figure 7.6 Choose Run as SAP Fiori Launchpad Sandbox

3. For the **File Name** field, choose `/fioriSandboxConfig.json` from the dropdown list. Keep the other fields unchanged, as shown in [Figure 7.7](#).

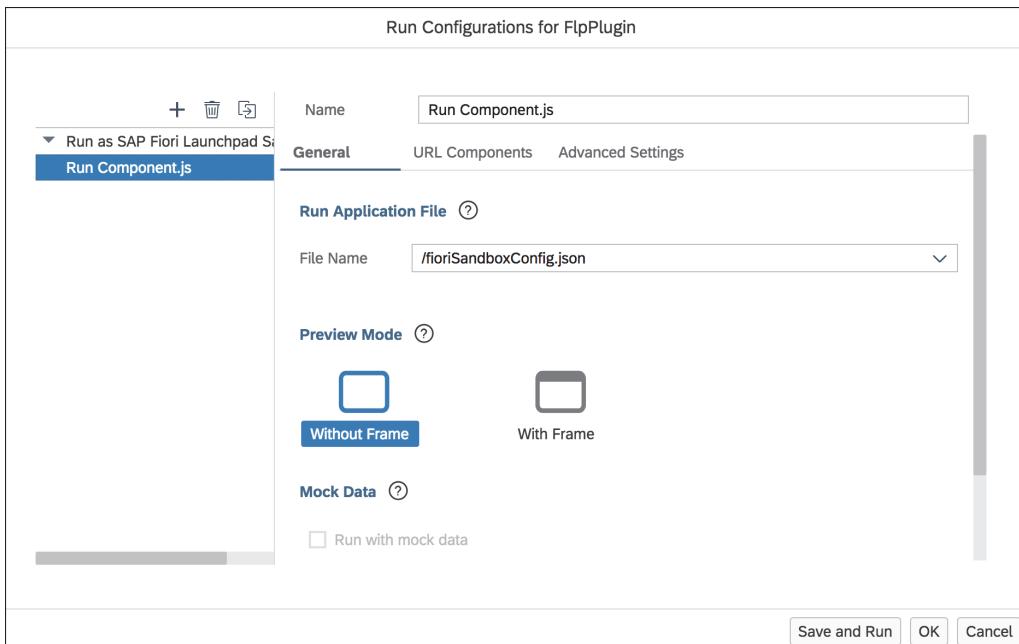


Figure 7.7 Run Configuration: General

4. Switch to the **URL Components** tab and the set value of the **URL Hash Fragment** field to “#Shell-home”, as shown in [Figure 7.8](#).

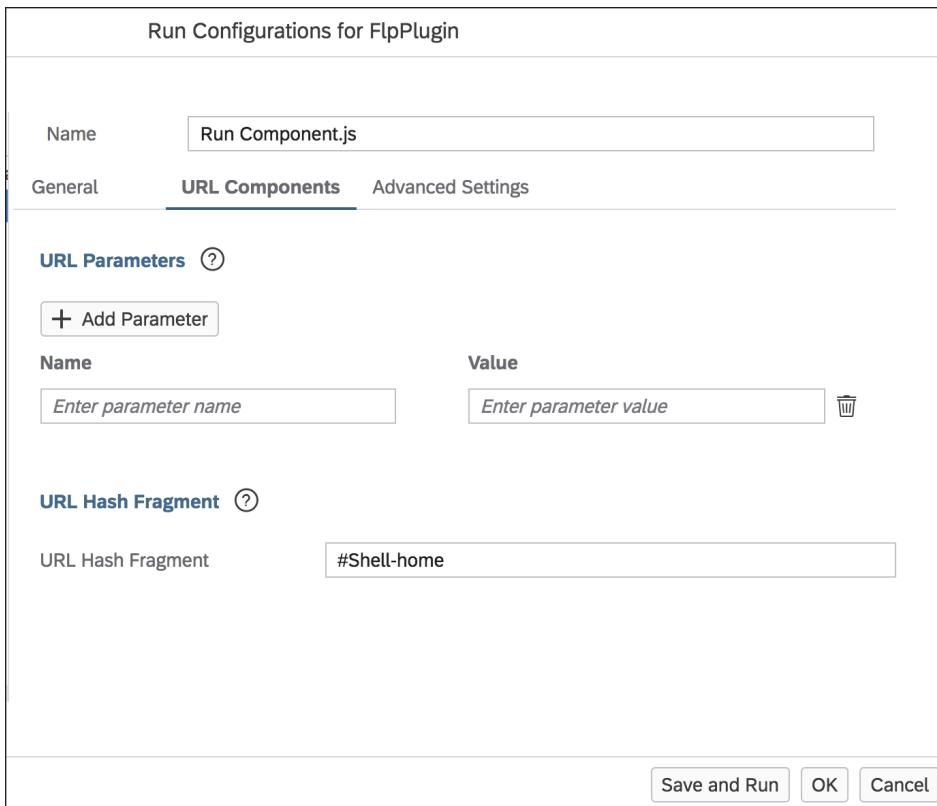


Figure 7.8 Run Configuration: URL Components

5. Click **Save and Run**.

The reason for a test like this is that the plug-in doesn't have a UI. The *fioriSandbox-Config.json* file is a configuration file that loads the plug-in. To make sure the index page displays, you need to specify the intent as #Shell-home in the run configuration.

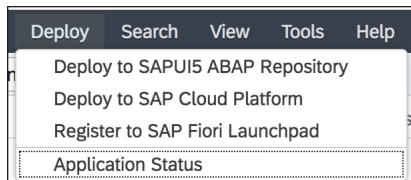
## 7.2 Deploying the Plug-In on SAP Cloud Platform

After development and testing, it's time to deploy your plug-in to SAP Fiori launchpad. First you need to deploy your plug-in as an SAPUI5 application on SAP Cloud Platform, then you need to add an app as a plug-in in SAP Fiori configuration cockpit for your SAP Fiori launchpad site.

### 7.2.1 Deployment and Activation

As we mentioned in earlier chapters, you need to deploy your plug-in as an SAPUI5 application to SAP Cloud Platform. Proceed as follows:

1. In your SAP Web IDE full-stack version, keep your project for the plug-in selected. Choose menu path **Deploy • Deploy to SAP Cloud Platform**, as shown in [Figure 7.9](#).



**Figure 7.9** Deploy to SAP Cloud Platform

2. On the **Deploy Application to SAP Cloud Platform** page, keep everything unchanged and click **Deploy**, as shown in [Figure 7.10](#).

A screenshot of the 'Deploy Application to SAP Cloud Platform' dialog box. The title bar says 'Deploy Application to SAP Cloud Platform'.  
  
The 'Application Details' section contains:

- A radio button group for 'Deploy a new application' (selected) and 'Update an existing application'.
- \*Account: p2000524802trial with a dropdown arrow and a 'Get Accounts' button.
- Project Name: FlpPlugin
- \*Application Name: flpplugin

  
The 'Version Management' section contains:

- \*Version: 1.0.0 with an 'Activate' checkbox checked.

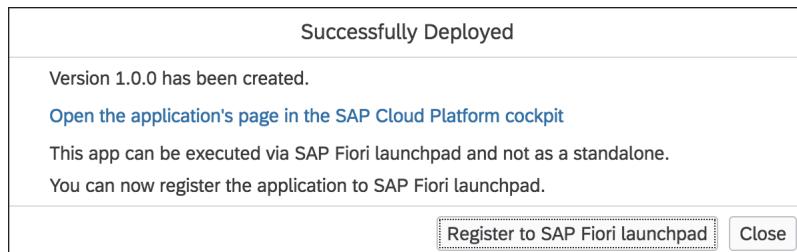
  
At the bottom right are 'Deploy' and 'Cancel' buttons.

**Figure 7.10** Deploy Application to SAP Cloud Platform: Application Details

3. Once you receive the message that your application has deployed successfully, click **Close**, as shown in [Figure 7.11](#).

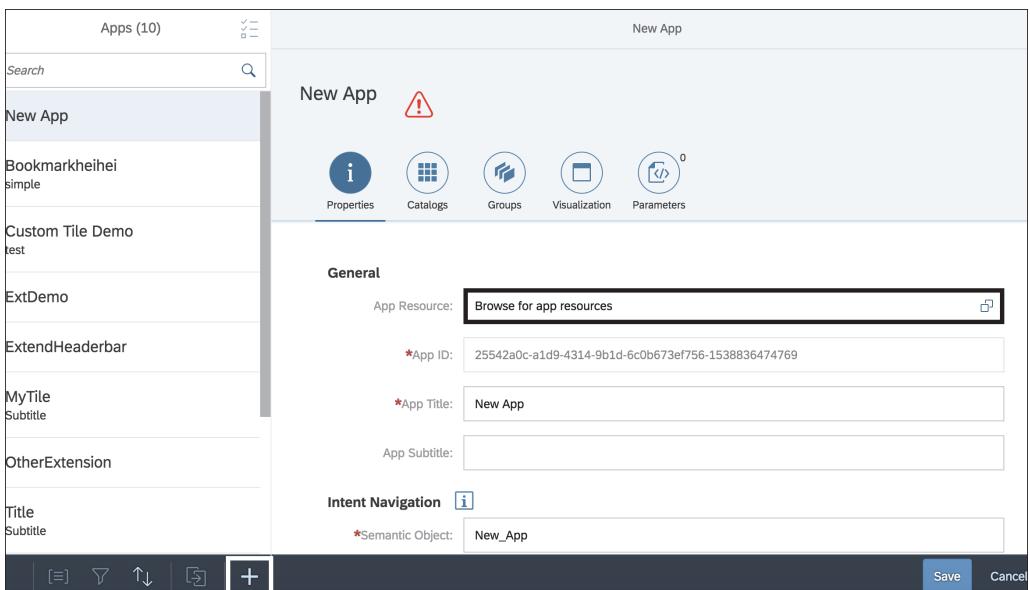
### Note

The plug-in is not an SAPUI5 application, so it can't be registered to SAP Fiori launchpad here. You must do it manually.



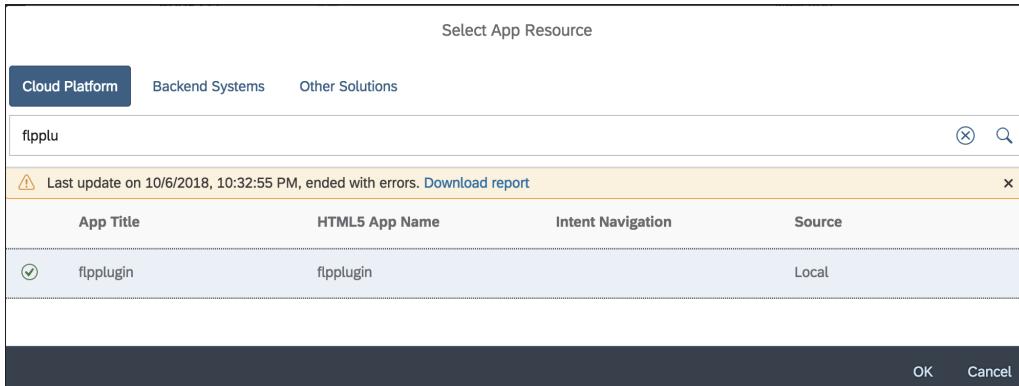
**Figure 7.11** SAP Fiori Application Successfully Deployed

4. Enter the management page of your SAP Fiori launchpad site and follow menu path **Content Management • Apps**.
5. Click the **Plus** button to add an app and open the search help for **App Resource**, as shown in [Figure 7.12](#).



**Figure 7.12** Add New App

6. In the value help dialog, search for “flppugin”, then choose the line that represents your plug-in and click **OK**, as shown in [Figure 7.13](#).



**Figure 7.13** Select Your Plug-In

7. Scroll down to the **App Resources Details** section. For **App Type**, choose **Shell Plugin** from the dropdown list. For **Shell Plugin Type**, choose **Custom**. Leave the other fields unchanged, as shown in [Figure 7.14](#).

The screenshot shows the "App Resource Details" configuration form. It contains five input fields:

- "App Type:" dropdown set to "Shell Plugin".
- "Shell Plugin Type:" dropdown set to "Custom".
- "Component URL:" text input field containing "/".
- "\*SAPUI5 Component :" text input field containing "flpdev.flppugin".
- "HTML5 App Name:" text input field containing "flppugin".

**Figure 7.14** Set App as Plug-In

8. Switch to the **Catalogs** tab and click the **Plus** button to assign the app to a tile catalog, as shown in [Figure 7.15](#).

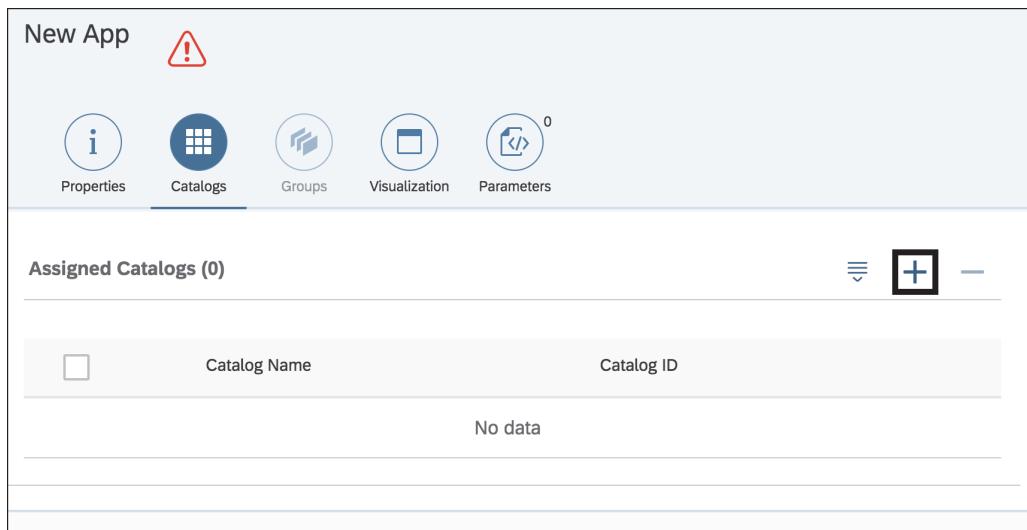


Figure 7.15 Assign Catalog

9. Select **Sample Catalog** and click **OK** in the **Select Catalogs** pop-up shown in Figure 7.16.

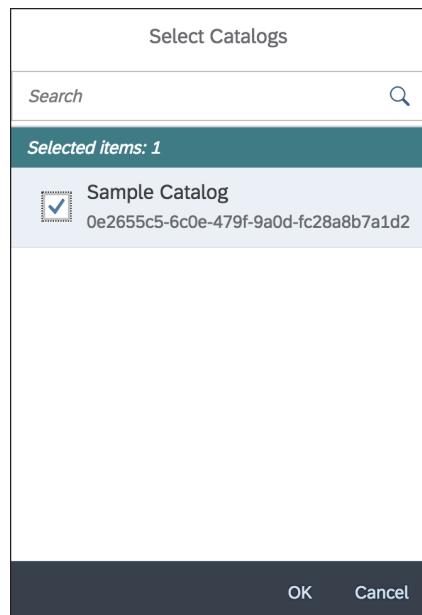


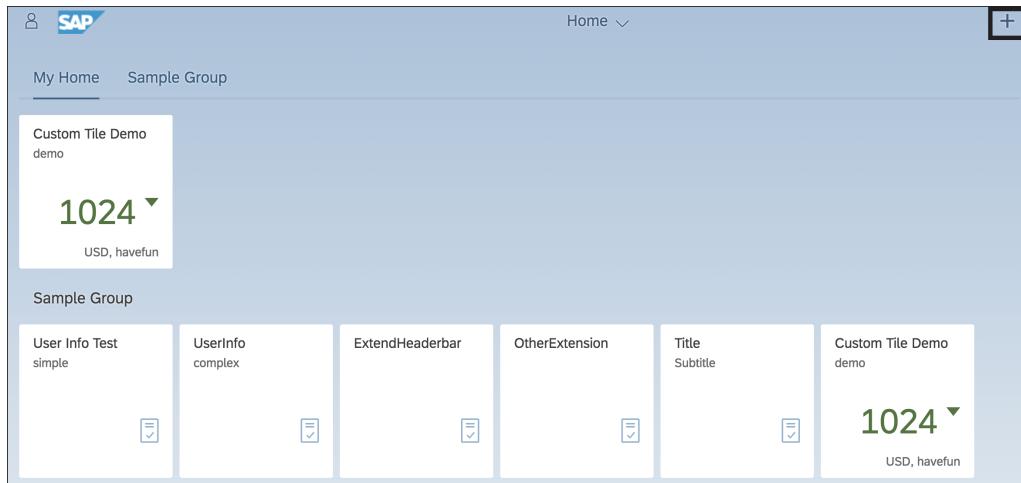
Figure 7.16 Select Catalogs

10. Save all your changes by clicking **Save** on the footer toolbar.
11. Click the globe button in the top-right corner to open the **Publish Site** pop-up window.
12. Check **Clear HTML5 Application Cache**, then click **Publish and Open**, as shown in [Figure 7.17](#).



**Figure 7.17** Publishing Site

13. The result should look like [Figure 7.18](#).

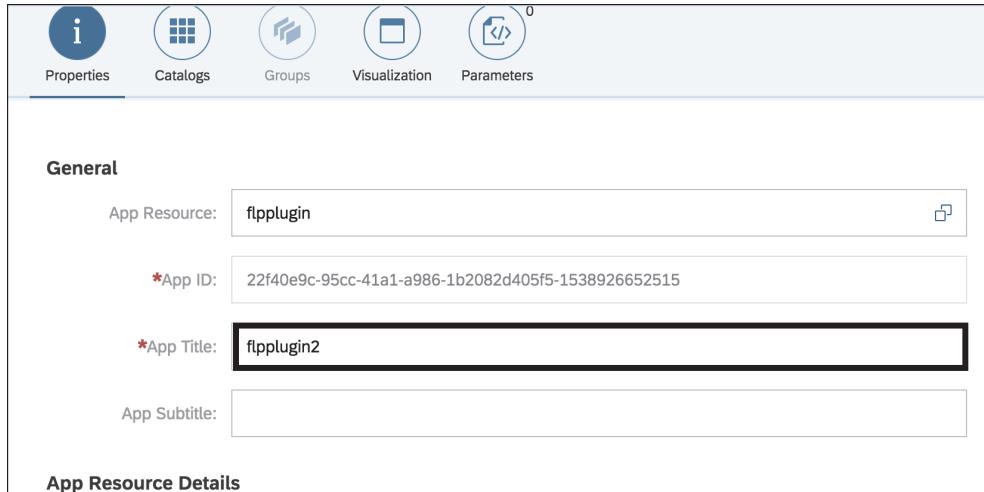


**Figure 7.18** Plug-In in SAP Cloud Platform Portal

### 7.2.2 Avoiding Multiple Code Executions

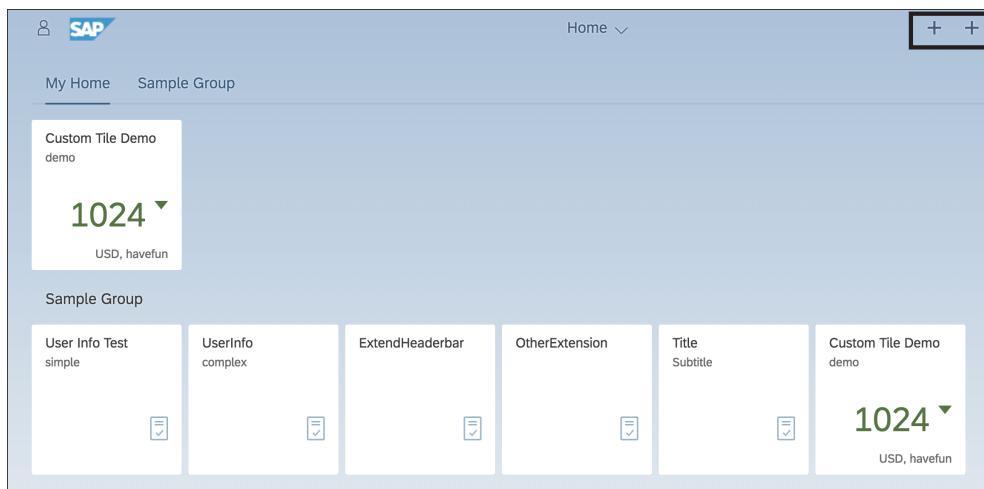
As you can see, the plug-in is easy to configure. But there are potential risks to configuring a plug-in more than once. The following example will configure and deploy the app again, with some unintended consequences:

1. Return to your configuration page for your SAP Fiori launchpad site.
2. Add an app again like in the previous demo. The only difference is that you need to change **App Title** to “flppugin2”, as shown in [Figure 7.19](#).



[Figure 7.19](#) Configure Another App for Same Plug-In

3. Clear the cache and publish your site again; you'll see two items appearing at the end of the shell header, as shown in [Figure 7.20](#).



[Figure 7.20](#) Result of Configuring Plug-Ins Twice

To solve this problem, you need to ensure your code executes only once regardless of how many times the plug-in is executed. A general approach is to check for the existence of the control you want to add before adding it. To do so, switch back to SAP Web IDE full-stack version, then change the `init` method of `component.js` as shown in [Listing 7.4](#).

```
init: function () {
    var rendererPromise = this._getRenderer();
    rendererPromise.then(function (oRenderer) {
        //get reference of the item
        var oItem = sap.ui.getCore().byId("addbookmarkitem");
        //Add item only if the item does not exist
        if (!oItem) {
            oRenderer.addHeaderEndItem(
                "sap.ushell.ui.shell.ShellHeadItem",
                {
                    id: "addbookmarkitem",
                    icon: "sap-icon://add",
                    tooltip: "Add bookmark",
                    press: function () {
                        MessageToast.show("This SAP Fiori launchpad has been extended
to improve your experience");
                    }
                },
                true,
                true);
        }
    });
},
```

**Listing 7.4** FlpPlugin: Code for Avoiding Multiple Executions

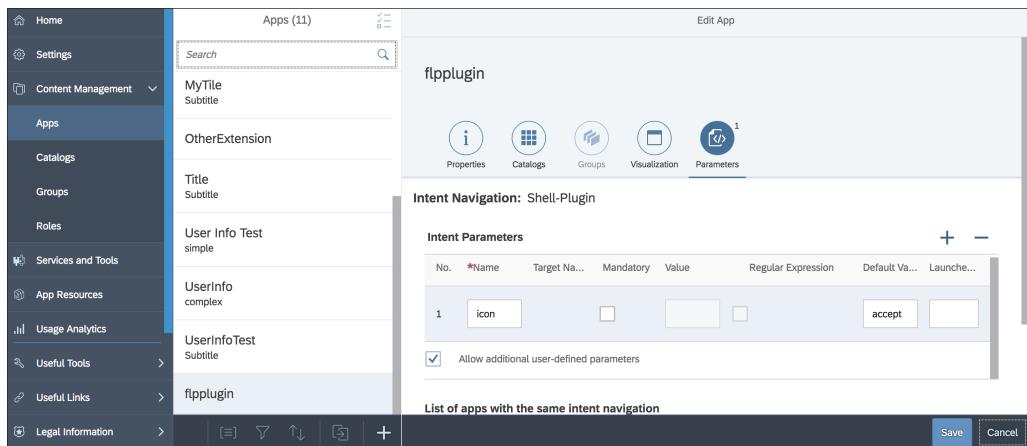
Deploy your plug-in again and publish your site once more. You'll find that the issue has been solved.

### 7.2.3 Working with Configurable Parameters

By working with configurable parameters, you can make your plug-in more flexible. In this section, you'll modify your plug-in to work with configurable parameters, as follows:

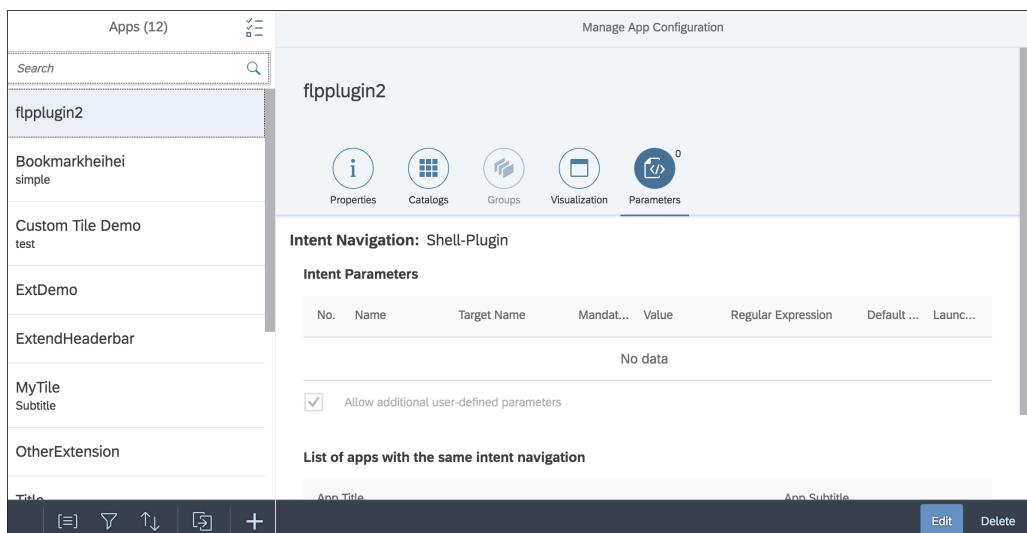
1. Return to your SAP Fiori launchpad site management page. Follow menu path **Content Management • Apps**, then select **flpplugin** in the app list. Enter edit mode by clicking **Edit**.

2. Switch to the **Parameters** tab. Click the **Plus** button on the toolbar of the **Intent Parameters** table. Enter “icon” in the **Name** column and “accept” in the **Default Value** column. Then click **Save** to save all changes. The form should look like [Figure 7.21](#).



**Figure 7.21** Set Parameter for Plug-In

3. To avoid confusion, delete the **flppugin2** app by clicking on it in the apps list and clicking **Delete**, as shown in [Figure 7.22](#).



**Figure 7.22** Delete flppugin2

4. Switch to the SAP Web IDE and change the code of the `init` method of `component.js` as shown in [Listing 7.5](#). Here we get parameter `icon` as the icon's name, after which we can display the icon according to user settings.

```
init: function () {

    var rendererPromise = this._getRenderer();
    //Get configurable parameter and generate the icon URL
    var oComponentData = this.getComponentData();
    var icon = oComponentData.config.icon;
    var iconUrl = "sap-icon://" + icon;
    rendererPromise.then(function (oRenderer) {
        var oItem = sap.ui.getCore().byId("addbookmarkitem");
        if (!oItem) {
            oRenderer.addHeaderEndItem(
                "sap.ushell.ui.shell.ShellHeadItem", {
                    id: "addbookmarkitem",
                    //Change the static parameter to variable
                    icon: iconUrl,
                    tooltip: "Add bookmark",
                    press: function () {
                        MessageToast.show("This SAP Fiori launchpad has been extended to im-
prove your experience");
                    }
                }, true, true);
        }
    });
},
```

**Listing 7.5** FlpPlugin: Read Parameters

5. Deploy your plug-in again to SAP Cloud Platform and republish your site, as shown in [Figure 7.23](#).



**Figure 7.23** Publishing Site

- As a result, the icon for the item you've added will change according to your configuration.

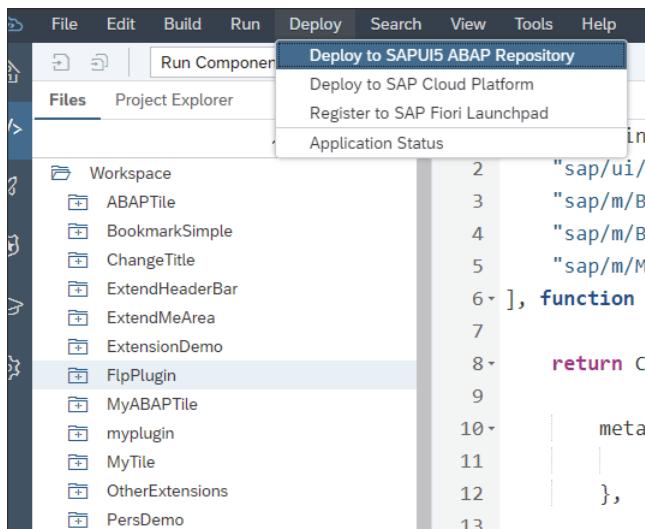
## 7.3 Deploying the Plug-In on SAP NetWeaver AS ABAP

A change in code level isn't needed to deploy your plug-in on SAP NetWeaver AS ABAP. The only difference is the method of deployment and configuration. In this section, we'll guide you through these processes.

### 7.3.1 Deployment

To deploy your plug-in to SAP NetWeaver AS ABAP, you first need to deploy it to the ABAP Repository as a BSP (Business Server Page) application. Then use SAP Fiori launchpad designer to create a target mapping using a predefined intent called shell plugin. To begin the deployment, follow these steps:

- Switch back to SAP Web IDE full-stack version and choose **Deploy • Deploy to SAPUI5 ABAP Repository**, as shown in [Figure 7.24](#).



**Figure 7.24** Menu for Deploying to ABAP

- In the **Deployment Options** step, choose your system and select **Deploy a New Application**. Then click **Next**, as shown in [Figure 7.25](#).

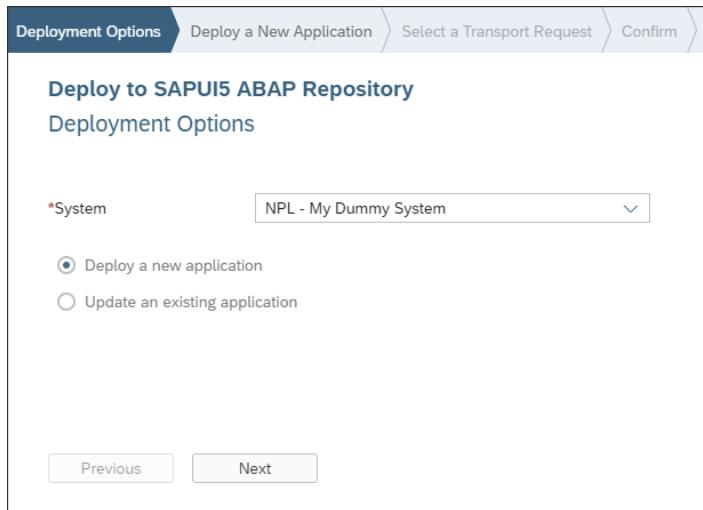


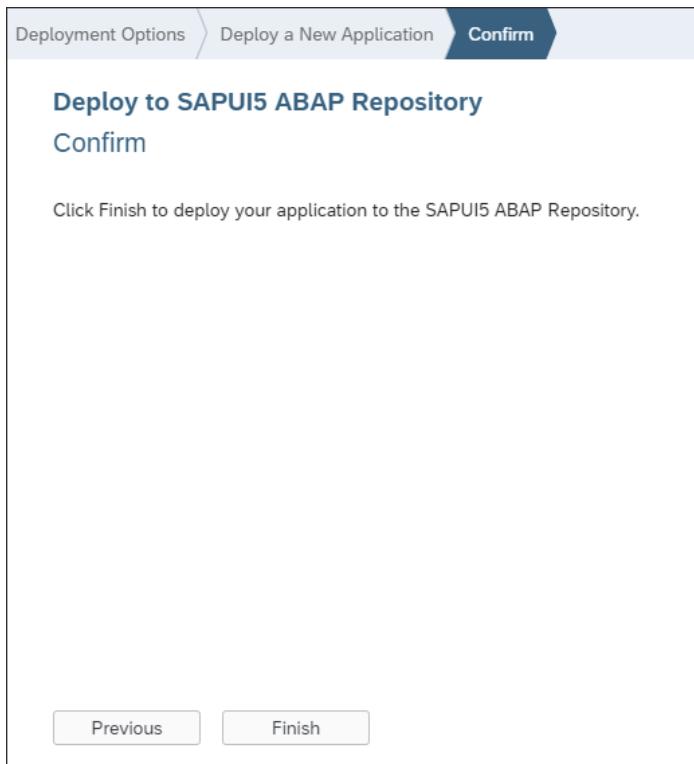
Figure 7.25 Deployment Options

3. In the **Deploy a New Application** step, enter “zplugin” for **Name** and text of your choice for **Description**. Select **\$TMP** by clicking the **Browse** button to the right of the **Package** field. Then click **Next**, as shown in [Figure 7.26](#).

The screenshot shows the 'Deploy a New Application' step of the deployment process. The navigation bar at the top shows the steps: Deployment Options > Deploy a New Application > Select a Transport Request > Confirm. The title 'Deploy to SAPUI5 ABAP Repository' and 'Deploy a New Application' are displayed. A section titled 'Select a package for the application' contains three fields: 'Name' (set to 'zplugin'), 'Description' (set to 'MyPlugin'), and 'Package' (set to '\$TMP'). To the right of the 'Package' field is a 'Browse' button. At the bottom of the screen are 'Previous' and 'Next' buttons.

Figure 7.26 Deploy New Application

- Finally, click **Finish** in the **Confirm** step, as shown in [Figure 7.27](#). Your application has now been successfully deployed!



**Figure 7.27** Confirm Deployment of SAPUI5 Application

### 7.3.2 Configuration

With the plug-in deployed, you can start to configure and activate the plug-in for your user. A predefined intent is reserved for plug-ins: shell plugin. To begin, follow these steps:

- Open SAP Fiori launchpad designer on your local computer.
- Add a tile catalog by clicking the **Add** button in the footer toolbar of the catalog list. Use “MyPlugin” for the **Title** and “ZPLUGIN” for the **ID**. Then click **Save** in the pop-up window, as shown in [Figure 7.28](#).

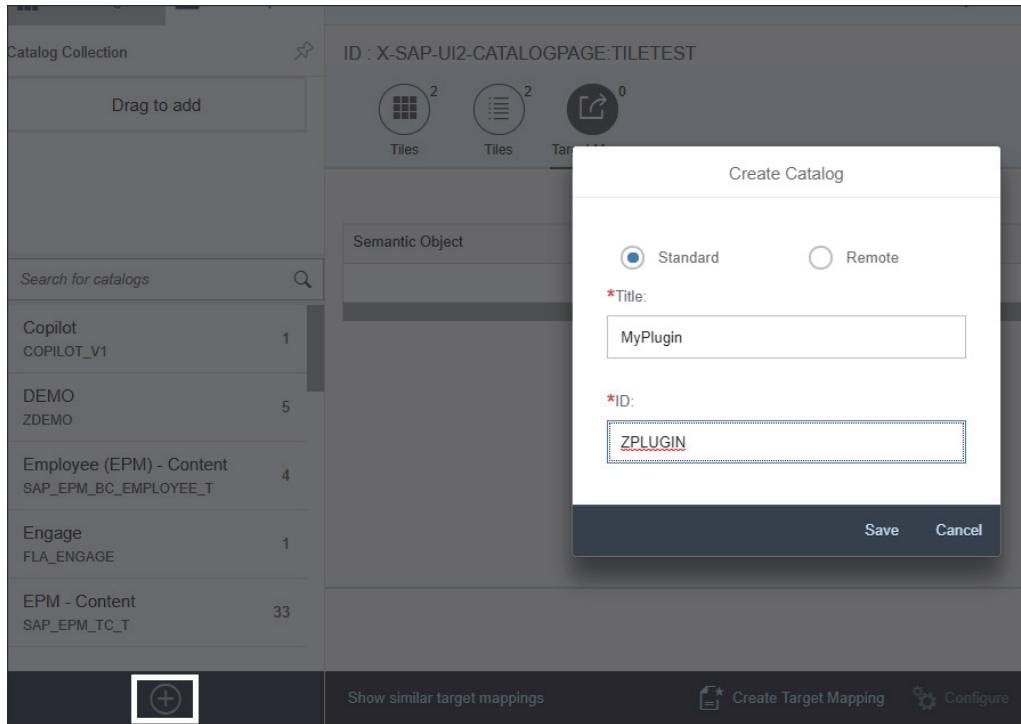


Figure 7.28 Add Catalog

3. Switch to the **Target Mapping** tab and click **Create Target Mapping** in the footer toolbar, as shown in [Figure 7.29](#).

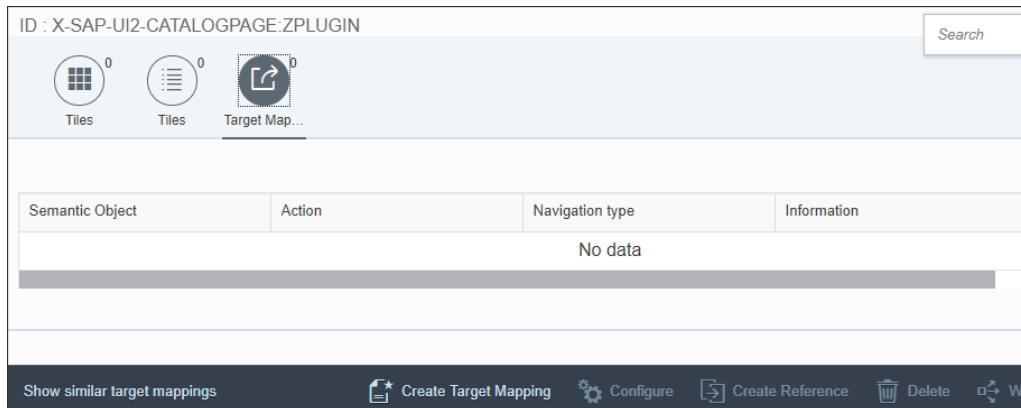


Figure 7.29 Create New Target Mapping

4. Fill in the form using the information in Table 7.1.

| Field            | Value                       |
|------------------|-----------------------------|
| Semantic Object  | Shell                       |
| Action           | Plugin                      |
| Application Type | SAPUI5 Fiori App            |
| Title            | AddHeaderEndItem            |
| URL              | /sap/bc/ui5_ui5/sap/zplugin |
| ID               | fipdev.flppugin             |

**Table 7.1** Properties for Plug-In Target Mapping

5. In the **Parameters** table, add a parameter, entering “icon” for **Name** and “accept” for **Default Value**. The result should look like Figure 7.30.

| Parameter | Mandatory                | Value | Is Regular Expression    | Default Value                              | Target N... |
|-----------|--------------------------|-------|--------------------------|--|-------------|
| Name      | <input type="checkbox"/> |       | <input type="checkbox"/> | <input checked="" type="checkbox"/> accept |             |
| icon      | <input type="checkbox"/> |       | <input type="checkbox"/> |  |             |

**Figure 7.30** Details of Target Mapping

6. Save, and close SAP Fiori launchpad designer.

7. Logon to the SAP GUI and enter Transaction PFCG.
8. Enter “ZROLE\_PLUGIN” for the **Role** name and click the **Single Role** button, as shown in Figure 7.31.

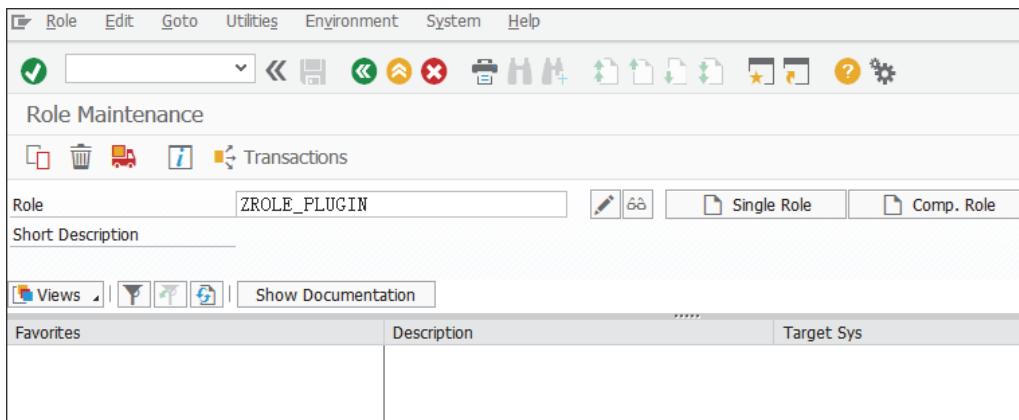


Figure 7.31 Create New Role

9. Provide a **Description** and click **Save**, as shown in Figure 7.32.

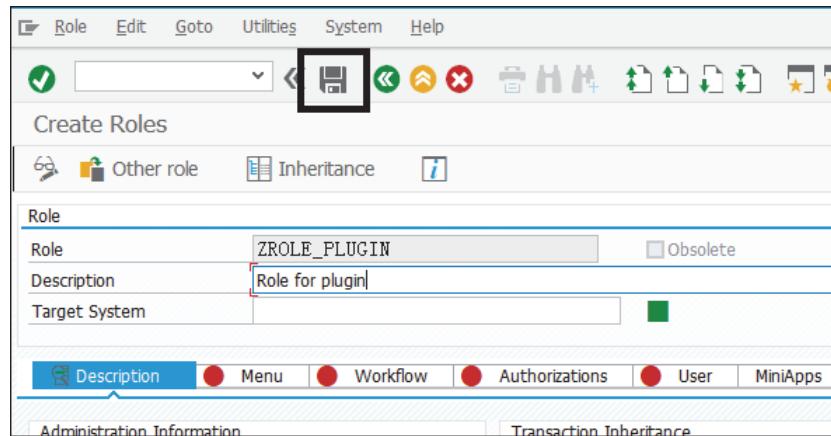


Figure 7.32 Save Role before Continuing

10. Switch to the **Menu** tab, click the small triangle after button **Transaction**, and select **SAP Fiori Tile Catalog**, as shown in Figure 7.33.

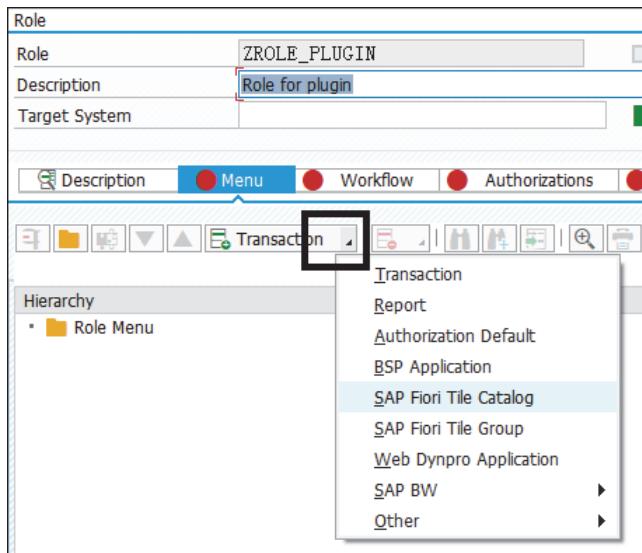


Figure 7.33 Add Tile Catalog as Menu

11. In the pop-up window, enter “ZPLUGIN” for **Catalog ID**, then click the green checkmark to confirm, as shown in Figure 7.34.

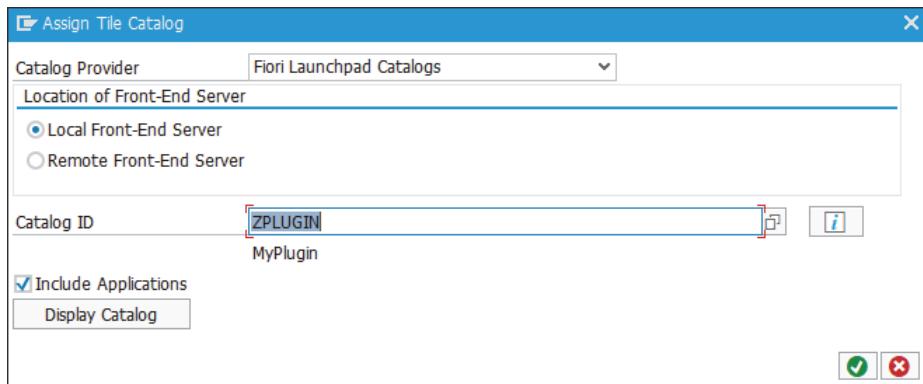
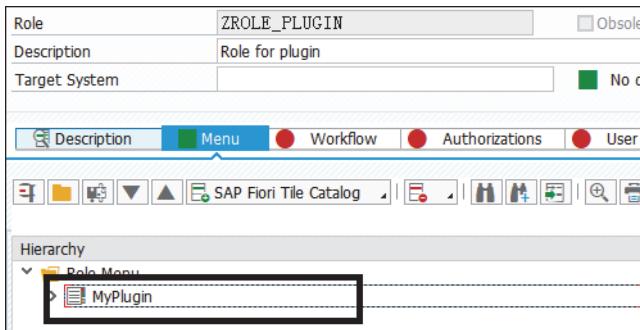


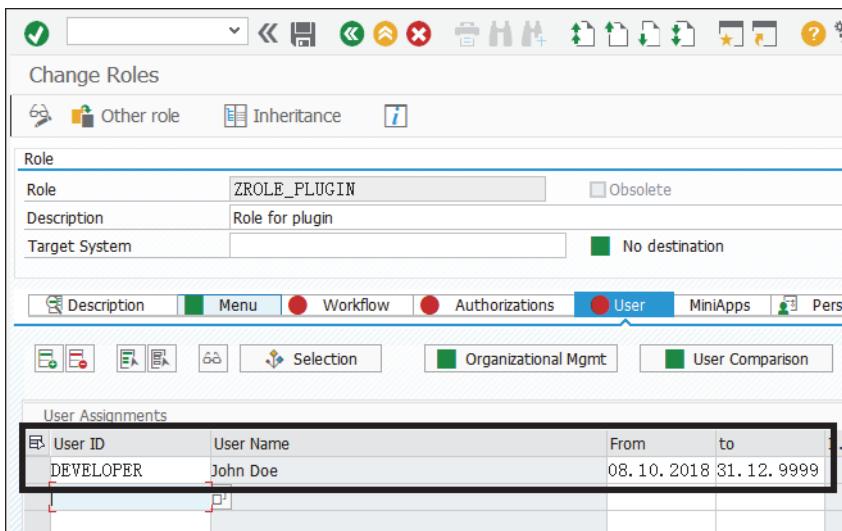
Figure 7.34 Select Catalog

12. The result should look like [Figure 7.35](#).



**Figure 7.35** Result of Adding Tile Catalog

13. Switch to the **User** tab, add your user to the list, and save the role, as shown in [Figure 7.36](#).



**Figure 7.36** Add User and Save

14. Switch back to your browser and enter SAP Fiori launchpad on your SAP NetWeaver AS ABAP system. The SAP Fiori launchpad will now look like [Figure 7.37](#).

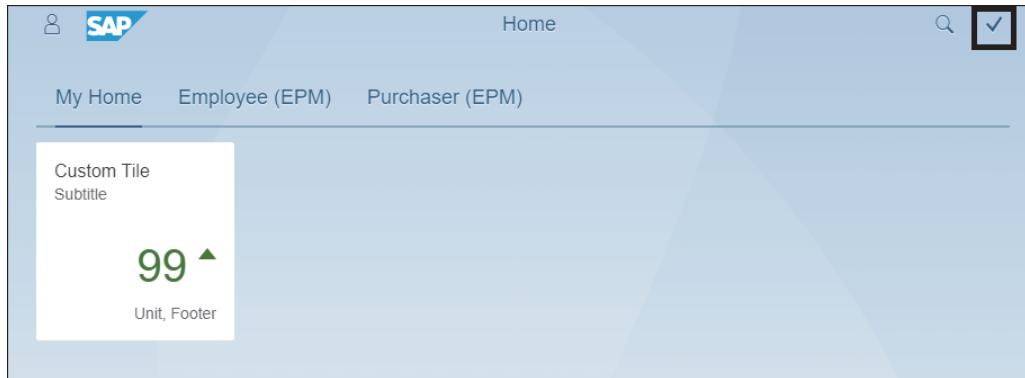


Figure 7.37 Result of Plug-In Activated on SAP NetWeaver AS ABAP

## 7.4 Predefined Plug-Ins

There are predefined plug-ins for special purposes in the system. For example, a plug-in for setting user defaults is available for SAP S/4HANA Finance to help users set default values for common parameters. Another plug-in is for enabling or disabling runtime authoring tools for a specific user.

### 7.4.1 Plug-In for Setting User Defaults

This plug-in adds setting options in the user setting dialog. It consists of common finance parameters like company code.

If you're using SAP Fiori launchpad on SAP NetWeaver AS ABAP and all apps for SAP S/4HANA have been installed, you can find the plug-in in business catalog SAP\_SFIN\_BC\_USER\_PARAM.

For SAP Fiori launchpad on SAP Cloud Platform, you need to perform the following steps:

1. Import the Default Values shell plug-in app using the Transport Manager.

#### Note

This feature is only available if you're subscribed to SAP Fiori Cloud for SAP S/4HANA Finance content.

2. In the Configure Apps editor, select the **Parameters** tab. For each application that was developed to support user defaults as a navigation parameter, populate the parameter fields using the following structures:
  - Parameter name: <parametername>
  - Parameter default value: userdefault.<parametername>

These parameters enable the apps to use the Default Values shell plug-in.

#### 7.4.2 Plug-In for Activating Runtime Authoring

Flexibility is key! Enterprise software must adapt to rapidly changing environments. For example, customers need their apps to fit their processes without long IT projects to adapt them, and cloud providers want to run the same software for everyone to reduce total cost of ownership (TCO). Do you think adapting the user interface of SAP Fiori apps (e.g., by adding, hiding or rearranging fields) is a complex process? Think again! SAPUI5 flexibility services allow for upgrade-safe and modification-free UI changes on different levels (e.g., on the customer side) that can be performed by different users (end users, key users, and developers).

The plug-in for runtime authoring can enable or disable key users from creating their own adaptions of an SAP Fiori application based on the SAPUI5 flexibility service.

In SAP NetWeaver AS ABAP, there's already a tile catalog called SAP\_UI\_FLEX\_KEY\_USER. A plug-in is contained in this catalog, which adds a new item in the Me Area, which is used to adapt SAP Fiori apps at runtime.

### 7.5 Summary

In this chapter, you learned how to develop a plug-in using SAP Web IDE full-stack version and how to deploy it to both cloud and on-premise environments. When you want to call services at startup or extend SAP Fiori launchpad in your system, you can write the code you used in [Chapter 4](#) and [Chapter 5](#) in the plug-ins.

After reading all seven chapters of this book, I hope you've found that SAP Fiori launchpad is a good friend, there to give you a lot of help when you develop your SAPUI5 apps and ready for you to add your own customizations and extensions.

## The Author



**Steve Guo** is an SAP mentor, developer, and trainer who has been working with SAP for more than 8 years. Since 2014 he has been working with SAP's new technologies, including SAP HANA and SAP Fiori, and has helped numerous developers and SAP customers adopt and use SAP software. Steve is the author of three SAP training courses for SAP Fiori and other SAP user interface technologies. He has been recognized as a Gold trainer for his training work by SAP Greater China.



# Index

## A

|   |          |
|---|----------|
| ABAP Development Tools (ADT) .....              | 78       |
| ABAP Repository .....                           | 287      |
| <i>deployment options</i> .....                 | 288      |
| Action .....                                    | 19       |
| Administrators .....                            | 18       |
| API reference .....                             | 140      |
| App descriptor .....                            | 123, 196 |
| <i>setting parameters</i> .....                 | 197      |
| App finder .....                                | 18, 220  |
| App node deletion .....                         | 229      |
| App resource .....                              | 279      |
| <i>details</i> .....                            | 280      |
| Application container .....                     | 98       |
| Application programming interfaces (APIs) ..... | 17       |

## B

|   |                                   |
|---|-----------------------------------|
| Binding parameters .....                          | 266                               |
| Binding path .....                                | 158                               |
| Bookmark service .....                            | 22, 128, 167                      |
| <i>additional functions</i> .....                 | 177                               |
| <i>button</i> .....                               | 177                               |
| <i>existing tiles</i> .....                       | 178                               |
| <i>testing</i> .....                              | 168, 175                          |
| <i>tiles</i> .....                                | 172                               |
| Boot disk file .....                              | 49                                |
| Business Object Processing Framework (BOPF) ..... | 188                               |
| Business Server Page (BSP) .....                  | 287                               |
| Button control .....                              | 194                               |
| Buttons .....                                     | 133, 170, 177, 180, 202, 211, 218 |
| <i>custom</i> .....                               | 191                               |
| <i>disable</i> .....                              | 120                               |
| <i>enable state</i> .....                         | 119                               |
| <i>pressed code</i> .....                         | 163                               |

## C

|                         |     |
|-------------------------|-----|
| Callback function ..... | 177 |
| Callback method .....   | 118 |

|   |                       |
|---|-----------------------|
| Chrome developer tools .....                    | 142                   |
| Client-side rendering CHIPs .....               | 251                   |
| Client-side services .....                      | 22, 127               |
| Code editor .....                               | 193, 231              |
| Collaborative Human Interface Part (CHIP) ..... | 248, 251, 253         |
| <i>API</i> .....                                | 259, 265              |
| <i>configuration parameters</i> .....           | 264                   |
| <i>create</i> .....                             | 255                   |
| <i>details</i> .....                            | 255                   |
| <i>value</i> .....                              | 257                   |
| ColumnMicroChart control .....                  | 234                   |
| Configurable parameters .....                   | 284                   |
| Configuration cockpit .....                     | 20, 25, 277           |
| Configuration screen .....                      | 259                   |
| <i>create</i> .....                             | 258                   |
| <i>parameters</i> .....                         | 264                   |
| <i>tile initialization</i> .....                | 262                   |
| Configure Apps editor .....                     | 296                   |
| Container mode .....                            | 187                   |
| Content aggregation .....                       | 153                   |
| Core data services (CDS) .....                  | 188                   |
| Cross-app navigation ...                        | 112–113, 115–116, 121 |
| Custom tiles .....                              | 227                   |
| <i>add parameters</i> .....                     | 243                   |
| <i>best practices</i> .....                     | 266                   |
| <i>configure</i> .....                          | 262                   |
| <i>republish</i> .....                          | 241                   |
| <i>results</i> .....                            | 242                   |
| <i>SAP Cloud Platform deployment</i> .....      | 237                   |
| <i>SAP NetWeaver AS ABAP deployment</i> ....    | 248                   |
| <i>visualization parameters</i> .....           | 240                   |
| <i>visualizations</i> .....                     | 239                   |

## D

|                              |          |
|------------------------------|----------|
| Dashboard .....              | 36       |
| Data binding .....           | 169, 245 |
| Data changes frequency ..... | 185      |
| Data storage locations ..... | 184      |
| Deferred object .....        | 224      |
| Descriptor editor .....      | 150, 168 |

|  |     |
|--|-----|
| Destination .....                        | 93  |
| <i>configuration</i> .....               | 94  |
| Developer license .....                  | 69  |
| Developers .....                         | 19  |
| Development environment                  |     |
| <i>cloud-based</i> .....                 | 25  |
| <i>connect cloud to on-premise</i> ..... | 80  |
| <i>hardware requirements</i> .....       | 40  |
| <i>on-premise</i> .....                  | 38  |
| <i>on-premise architecture</i> .....     | 39  |
| <i>setup</i> .....                       | 25  |
| Development overview .....               | 21  |
| DNS record .....                         | 64  |
| Dynamic tiles .....                      | 179 |

## E

---

|                                      |                         |
|--------------------------------------|-------------------------|
| End users .....                      | 18                      |
| Error message .....                  | 118                     |
| Event handler .....                  | 133, 160, 163, 181, 196 |
| Event listener .....                 | 198, 212                |
| Events .....                         | 133                     |
| Extensions .....                     | 22, 189, 274            |
| <i>launch page</i> .....             | 211                     |
| <i>managing element states</i> ..... | 210                     |
| <i>Me Area</i> .....                 | 218                     |
| <i>options</i> .....                 | 189                     |
| <i>shell header</i> .....            | 201                     |

## F

---

|                           |               |
|---------------------------|---------------|
| FeedContent control ..... | 231           |
| Fetching data .....       | 223           |
| Firewall .....            | 53            |
| Footer bar .....          | 214           |
| <i>add</i> .....          | 215           |
| Footers .....             | 155, 170, 270 |
| Frame types .....         | 228           |

## G

---

|                           |          |
|---------------------------|----------|
| Generic tiles .....       | 227      |
| <i>create</i> .....       | 229      |
| GenericTile control ..... | 230, 247 |

## H

---

|                                    |          |
|------------------------------------|----------|
| Hardware key .....                 | 73       |
| Header end item .....              | 190      |
| <i>add</i> .....                   | 205–206  |
| <i>pop-up</i> .....                | 207      |
| <i>URL redirect</i> .....          | 206      |
| Header item .....                  | 190, 204 |
| <i>add</i> .....                   | 206, 208 |
| <i>pop-up</i> .....                | 209      |
| <i>show</i> .....                  | 205      |
| <i>with link</i> .....             | 208      |
| Headers .....                      | 274      |
| Hierarchy navigation .....         | 199      |
| Hook method .....                  | 246      |
| Hosts file .....                   | 68       |
| HTML5 app name .....               | 238      |
| HTML5 application repository ..... | 25       |
| HTTP Service Management app .....  | 78       |
| HTTPS certificate .....            | 75       |
| HTTPS protocol .....               | 87       |

## I

---

|                             |                        |
|-----------------------------|------------------------|
| Icon control .....          | 169                    |
| Icons .....                 | 286                    |
| ImageContent control .....  | 232                    |
| Inbound navigation .....    | 109                    |
| index.html .....            | 97                     |
| Input control .....         | 132, 152, 169, 181     |
| Intent .....                | 19, 101, 114, 118, 121 |
| <i>access</i> .....         | 101                    |
| <i>app descriptor</i> ..... | 123                    |
| <i>availability</i> .....   | 117                    |
| <i>configuration</i> .....  | 111                    |
| <i>navigation</i> .....     | 117                    |
| <i>parameters</i> .....     | 108                    |
| <i>propose</i> .....        | 107                    |
| <i>URL</i> .....            | 112                    |

|                               |              |
|-------------------------------|--------------|
| Intent-based navigation ..... | 101–102, 167 |
| <i>benefits</i> .....         | 103          |
| Internal host .....           | 90           |
| IP address .....              | 64           |

**J**

|   |                                       |
|---|---------------------------------------|
| Java Platform, Standard Edition Development Kit (JDK) ..... | 80, 82                                |
| JavaScript APIs .....                                       | 99, 127                               |
| JavaScript promise objects .....                            | 145                                   |
| jQuery object .....   | 222                                   |
| JSON model .....  | 150, 157, 168, 171, 180, 245–246, 265 |
| JSON object .....   | 115, 204, 211, 221, 250               |

**L**

|                              |                              |
|------------------------------|------------------------------|
| Label control .....          | 131, 153                     |
| Languages .....              | 146, 165                     |
| <i>default</i> .....         | 146                          |
| <i>settings</i> .....        | 50                           |
| Launch page .....            | 190                          |
| Layout editor .....          | 131, 150, 158, 194, 229, 260 |
| <i>set properties</i> .....  | 230                          |
| License administration ..... | 72                           |
| Local disk .....             | 185                          |

**M**

|                                |                                 |
|--------------------------------|---------------------------------|
| Me Area .....                  | 18, 98, 191, 204, 218, 221, 270 |
| <i>add button</i> .....        | 220                             |
| <i>add settings</i> .....      | 221, 223                        |
| <i>event handlers</i> .....    | 219                             |
| <i>settings</i> .....          | 220                             |
| Memory size .....              | 46                              |
| Microchart controls .....      | 233                             |
| Microservices .....            | 21                              |
| Multiple code executions ..... | 282, 284                        |

**N**

|                                  |               |
|----------------------------------|---------------|
| Namespace .....                  | 252           |
| nano .....                       | 58, 63        |
| <i>install</i> .....             | 58            |
| Navigation .....                 | 112, 118, 247 |
| <i>configuring targets</i> ..... | 121           |
| <i>flow</i> .....                | 120           |
| <i>services</i> .....            | 115           |
| <i>target</i> .....              | 113, 125      |
| <i>testing</i> .....             | 117           |
| <i>to previous app</i> .....     | 119           |

|                              |     |
|------------------------------|-----|
| NewsContent control .....    | 232 |
| Notification center .....    | 18  |
| NumericContent control ..... | 233 |

**O**

|                                  |     |
|----------------------------------|-----|
| OData service .....              | 127 |
| OData service call .....         | 248 |
| onInit method .....              | 250 |
| onPress method .....             | 112 |
| onTilePress method .....         | 247 |
| openSUSE Linux .....             | 39  |
| <i>download</i> .....            | 41  |
| <i>install</i> .....             | 49  |
| Operating system .....           | 49  |
| <i>change hosts file</i> .....   | 68  |
| <i>network settings</i> .....    | 62  |
| <i>prepare for ABAP</i> .....    | 55  |
| Outbound navigation target ..... | 122 |
| Outline structure .....          | 153 |

**P**

|   |         |
|---|---------|
| Package selection .....                         | 254     |
| Page builder .....                              | 251     |
| Page control .....                              | 98      |
| Partition settings .....                        | 51      |
| Password .....                                  | 85      |
| Personalization service .....                   | 22, 179 |
| <i>complex data</i> .....                       | 185     |
| <i>fetch data</i> .....                         | 183     |
| <i>initialization with container mode</i> ..... | 186     |
| <i>initialize</i> .....                         | 182     |
| <i>restrict usage</i> .....                     | 187     |
| <i>save data</i> .....                          | 183     |
| Plug-ins .....                                  | 23, 269 |
| <i>add button</i> .....                         | 270     |
| <i>add header</i> .....                         | 274     |
| <i>basic information</i> .....                  | 270     |
| <i>characteristics</i> .....                    | 269     |
| <i>configuration</i> .....                      | 289     |
| <i>create with template</i> .....               | 270     |
| <i>development</i> .....                        | 269     |
| <i>generated component</i> .....                | 273     |
| <i>parameters</i> .....                         | 285     |
| <i>predefined</i> .....                         | 295     |
| <i>SAP Cloud Platform deployment</i> .....      | 277     |
| <i>SAP NetWeaver AS ABAP deployment</i> ....    | 287     |

|                                      |   |
|--------------------------------------|---|
| Plug-ins (Cont.)                     |   |
| <i>template customization</i>        | 272   |
| <i>templates</i>                     | 270   |
| <i>testing</i>                       | 275   |
| Port forwarding                      | 67  |
| Port number                          | 81  |
| Press event                          | 133, 160, 170, 247                                |
| Promise object                       | 196   |
| Promise type return object           | 171–172, 174                                      |
| Property target                      | 115   |
| <br><b>R</b>                         |   |
| Register app                         | 138   |
| Related apps                         | 200   |
| Renderer object                      | 201–202, 212, 219                                 |
| Report /UI2/INVALIDATE_CLIENT_CACHES | 262   |
| Resource bindle                      | 189   |
| Roles                                | 33–34, 292, 294                                   |
| Router                               | 167, 198  |
| Runtime authoring                    | 296   |
| <br><b>S</b>                         |   |
| SAP Belize Plus                      | 166   |
| SAP Cloud Connector                  | 40, 80, 92, 253                                   |
| <i>define subaccount</i>             | 86  |
| <i>download</i>                      | 44  |
| <i>install</i>                       | 81, 83  |
| <i>set up</i>                        | 84  |
| SAP Cloud Platform                   | 20, 84, 106, 108, 199                             |
| <i>create destination</i>            | 92  |
| <i>deploy app</i>                    | 110, 137, 278                                     |
| <i>index page</i>                    | 26  |
| <i>log on</i>                        | 35  |
| <i>registration</i>                  | 27  |
| <i>trial account</i>                 | 26  |
| <i>trial index</i>                   | 29  |
| <i>update app</i>                    | 147   |
| SAP Cloud Platform Cockpit           | 27, 92  |
| SAP Cloud Platform Portal            | 20, 23, 25, 135, 137, 165, 199, 237               |
| <i>activate</i>                      | 31  |
| <i>admin space</i>                   | 36  |
| <i>administration page</i>           | 238   |
| <i>app deployment</i>                | 237   |
| <i>authorization</i>                 | 32  |
| SAP Cloud Platform Portal (Cont.)    |   |
| <i>create site</i>                   | 36  |
| <i>enable</i>                        | 31  |
| <i>index page</i>                    | 246   |
| <i>plug-in</i>                       | 282   |
| <i>publish site</i>                  | 37  |
| <i>site template</i>                 | 36  |
| <i>testing</i>                       | 146   |
| SAP Community                        | 41  |
| SAP Enterprise Portal                | 21  |
| SAP Fiori Cloud                      | 295   |
| SAP Fiori launchpad designer         | 77, 252, 257, 289                                 |
| SAP Fiori launchpad sandbox          | 106, 135–136, 163, 184, 236, 275                  |
| SAP Fiori launchpad sites            | 75  |
| SAP Gateway                          | 87  |
| SAP GUI                              | 66, 73  |
| <i>logon</i>                         | 71  |
| SAP HANA XS Advanced                 | 21  |
| SAP JVM                              | 80  |
| <i>download</i>                      | 44  |
| SAP NetWeaver                        | 19  |
| SAP NetWeaver AS ABAP                | 23, 38–39, 84, 106, 253, 294                      |
| <i>commands</i>                      | 66  |
| <i>download</i>                      | 41  |
| <i>install components</i>            | 64  |
| <i>post-installation</i>             | 66  |
| SAP S/4HANA                          | 103, 295  |
| SAP S/4HANA Cloud                    | 21  |
| SAP S/4HANA Finance                  | 295   |
| SAP Screen Personas                  | 97  |
| SAP Smart Business                   | 227   |
| SAP Web IDE                          | 25, 29, 39, 97, 103, 129, 205, 229, 236, 245, 270 |
| <i>Chrome settings</i>               | 136   |
| <i>service page</i>                  | 30  |
| SAP_UI component                     | 20  |
| SAPUI5                               | 17, 97, 262                                       |
| <i>app creation</i>                  | 129, 148, 167, 179, 193                           |
| <i>app testing</i>                   | 135, 163  |
| <i>app title</i>                     | 106   |
| <i>application descriptor</i>        | 108   |
| <i>architecture</i>                  | 97, 100   |
| <i>component</i>                     | 238   |
| <i>embedding apps</i>                | 22, 103   |
| <i>flexibility service</i>           | 296   |

|                                 |               |
|---------------------------------|---------------|
| SAPUI5 (Cont.)                  |               |
| <i>SDK</i>                      | 142           |
| <i>template customization</i>   | 105           |
| <i>templates</i>                | 104           |
| <i>testing apps</i>             | 103           |
| Search                          | 18            |
| Secondary header title          | 203           |
| Select controls                 | 159           |
| Semantic object                 | 19            |
| Service alias                   | 192–193       |
| Shared folder                   | 59, 64        |
| <i>add</i>                      | 60            |
| Shell                           | 98            |
| <i>container</i>                | 99            |
| <i>header</i>                   | 190, 201, 283 |
| <i>plugin</i>                   | 269, 280      |
| <i>renderer</i>                 | 99            |
| ShellUIService                  | 192           |
| Simple form control             | 152–153       |
| Slide tiles                     | 235           |
| <i>numeric and microchart</i>   | 235           |
| SlideTile control               | 236           |
| SSH service                     | 53            |
| Startup failures                | 82            |
| States                          | 210           |
| <i>parameters</i>               | 211           |
| Storage file                    | 47            |
| Subheader                       | 190, 213      |
| <i>add</i>                      | 214           |
| System mapping                  | 88–89         |
| Tiles (Cont.)                   |               |
| <i>development</i>              | 249           |
| <i>footer</i>                   | 227           |
| <i>get parameters</i>           | 246           |
| <i>header image</i>             | 228           |
| <i>navigation</i>               | 248           |
| <i>parameters</i>               | 243           |
| <i>perform count</i>            | 175           |
| <i>register</i>                 | 251, 255      |
| <i>update</i>                   | 173           |
| <i>XML code</i>                 | 250           |
| Time zone settings              | 52            |
| Title                           | 189           |
| <i>change</i>                   | 196, 198      |
| <i>context menu</i>             | 198           |
| <i>information extensions</i>   | 192           |
| <i>set hierarchy</i>            | 199           |
| Title control                   | 155           |
| Tool area item                  | 215           |
| <i>expandable</i>               | 215, 217      |
| <i>simple</i>                   | 216           |
| Toolbar control                 | 155           |
| Transaction                     |               |
| <i>/UI2/PERS_EXPIRED_DELETE</i> | 185           |
| <i>PFCG</i>                     | 292           |
| <i>SE16</i>                     | 185           |
| <i>SICF</i>                     | 78            |
| <i>UI2/CHIP</i>                 | 255           |
| Transport Manager               | 295           |

## T

|  |                        |
|--|------------------------|
| Target mapping                         | 19, 101, 103, 290      |
| Templates                              | 129                    |
| Test environment                       | 112                    |
| Themes                                 | 146, 166               |
| Tile catalog                           | 19, 257, 262, 280, 293 |
| <i>add</i>                             | 290                    |
| Tile types                             | 23, 227, 258           |
| <i>none</i>                            | 238                    |
| TileContent control                    | 230–231                |
| Tiles                                  | 114, 208               |
| <i>add content</i>                     | 236                    |
| <i>content</i>                         | 231                    |
| <i>create</i>                          | 172                    |
| <i>delete</i>                          | 173                    |
| <i>deploy as SAPUI5 app</i>            | 253                    |
| UI add-on for SAP NetWeaver            | 20                     |
| Universally unique identifiers (UUIDs) | 55                     |
| URL components                         | 276                    |
| User defaults                          | 295                    |
| User experience                        | 31                     |
| User ID                                | 33, 44, 140            |
| User info service                      | 22, 128, 135           |
| <i>API</i>                             | 140                    |
| <i>methods</i>                         | 143, 148               |
| <i>promise object</i>                  | 145                    |
| <i>testing</i>                         | 143                    |
| <i>testing methods</i>                 | 144                    |
| User input                             | 171, 208               |
| User perspectives                      | 18                     |
| User settings                          | 164, 223–224           |
| Users                                  | 294                    |

## U

|                               |               |                       |             |
|-------------------------------|---------------|-----------------------|-------------|
| uuidd .....                   | 55            | Virtual port .....    | 89          |
| <i>check commands</i> .....   | 62            | VirtualBox .....      | 39, 59      |
| <i>check service</i> .....    | 61            | <i>download</i> ..... | 40          |
| <i>install</i> .....          | 56            | <i>install</i> .....  | 45          |
|                               |               | Virtual machine ..... |             |
|                               |               | <i>set up</i> .....   | 46          |
| <b>V</b>                      |               |                       |             |
| VBox .....                    | 202, 212, 219 |                       |             |
| <i>control</i> .....          | 194           | <b>W</b>              |             |
| Versions .....                | 19            | Web Dynpro .....      | 17, 97, 256 |
| Views .....                   | 131           |                       |             |
| <i>create new</i> .....       | 259           | <b>X</b>              |             |
| <i>information</i> .....      | 260           | XML view .....        | 177         |
| Virtual hard disk .....       | 47            | XTerm .....           | 62          |
| <i>size</i> .....             | 48            |                       |             |
| Virtual host .....            | 88            | <b>Y</b>              |             |
| Virtual machine .....         | 45            | Yast UI .....         | 56          |
| <i>network settings</i> ..... | 67            |                       |             |
| <i>reboot</i> .....           | 61            |                       |             |
| <i>start</i> .....            | 49            |                       |             |

# Service Pages

The following sections contain notes on how you can contact us.

## Praise and Criticism

We hope that you enjoyed reading this book. If it met your expectations, please do recommend it. If you think there is room for improvement, please get in touch with the editor of the book: [meaganw@rheinwerk-publishing.com](mailto:meaganw@rheinwerk-publishing.com). We welcome every suggestion for improvement but, of course, also any praise!

You can also share your reading experience via Twitter, Facebook, or email.

## Supplements

If there are supplements available (sample code, exercise materials, lists, and so on), they will be provided in your online library and on the web catalog page for this book. You can directly navigate to this page using the following link: [www.sap-press.com/4556](http://www.sap-press.com/4556). Should we learn about typos that alter the meaning or content errors, we will provide a list with corrections there, too.

## Technical Issues

If you experience technical issues with your e-book or e-book account at SAP PRESS, please feel free to contact our reader service: [support@rheinwerk-publishing.com](mailto:support@rheinwerk-publishing.com).

## About Us and Our Program

The website <http://www.sap-press.com> provides detailed and first-hand information on our current publishing program. Here, you can also easily order all of our books and e-books. Information on Rheinwerk Publishing Inc. and additional contact options can also be found at <http://www.sap-press.com>.

# Legal Notes

This section contains the detailed and legally binding usage conditions for this e-book.

## Copyright Note

This publication is protected by copyright in its entirety. All usage and exploitation rights are reserved by the author and Rheinwerk Publishing; in particular the right of reproduction and the right of distribution, be it in printed or electronic form.

© 2019 by Rheinwerk Publishing, Inc., Boston (MA)

## Your Rights as a User

You are entitled to use this e-book for personal purposes only. In particular, you may print the e-book for personal use or copy it as long as you store this copy on a device that is solely and personally used by yourself. You are not entitled to any other usage or exploitation.

In particular, it is not permitted to forward electronic or printed copies to third parties. Furthermore, it is not permitted to distribute the e-book on the Internet, in intranets, or in any other way or make it available to third parties. Any public exhibition, other publication, or any reproduction of the e-book beyond personal use are expressly prohibited. The aforementioned does not only apply to the e-book in its entirety but also to parts thereof (e.g., charts, pictures, tables, sections of text).

Copyright notes, brands, and other legal reservations as well as the digital watermark may not be removed from the e-book.

## Digital Watermark

This e-book copy contains a **digital watermark**, a signature that indicates which person may use this copy. If you, dear reader, are not this person, you are violating the copyright. So please refrain from using this e-book and inform us about this violation. A brief email to [info@rheinwerk-publishing.com](mailto:info@rheinwerk-publishing.com) is sufficient. Thank you!

## **Trademarks**

The common names, trade names, descriptions of goods, and so on used in this publication may be trademarks without special identification and subject to legal regulations as such.

All of the screenshots and graphics reproduced in this book are subject to copyright © SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany. SAP, the SAP logo, ABAP, Ariba, ASAP, Concur, Concur Expenses, Concur Tript, Duet, SAP Adaptive Server Enterprise, SAP Advantage Database Server, SAP Afaria, SAP ArchiveLink, SAP Ariba, SAP Business ByDesign, SAP Business Explorer, SAP BusinessObjects, SAP BusinessObjects Explorer, SAP BusinessObjects Lumira, SAP BusinessObjects Roambi, SAP BusinessObjects Web Intelligence, SAP Business One, SAP Business Workflow, SAP Crystal Reports, SAP Early-Watch, SAP Exchange Media (SAP XM), SAP Fieldglass, SAP Fiori, SAP Global Trade Services (SAP GTS), SAP GoingLive, SAP HANA, SAP HANA Vora, SAP Hybris, SAP Jam, SAP Max-Attention, SAP MaxDB, SAP NetWeaver, SAP PartnerEdge, SAPPHIRE NOW, SAP Power-Builder, SAP PowerDesigner, SAP R/2, SAP R/3, SAP Replication Server, SAP S/4HANA, SAP SQL Anywhere, SAP Strategic Enterprise Management (SAP SEM), SAP SuccessFactors, The Best-Run Businesses Run SAP, TwoGo are registered or unregistered trademarks of SAP SE, Walldorf, Germany.

## **Limitation of Liability**

Regardless of the care that has been taken in creating texts, figures, and programs, neither the publisher nor the author, editor, or translator assume any legal responsibility or any liability for possible errors and their consequences.