



SAP® Cloud Platform Integration

The Comprehensive Guide

- Set up and use SAP Cloud Platform Integration to connect cloud and hybrid systems
- Model synchronous, asynchronous, and B2B integration scenarios
- Learn to monitor, maintain, and secure integration flows

2nd edition, updated and expanded

Bilay · Gutsche
Krimmel · Stiehl



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>.

Herzig, Heitkötter, Wozniak, Agarwal, Wust

Extending SAP S/4HANA: Side-by-Side Extensions with the SAP S/4HANA Cloud SDK

2018, 618 pages, hardcover and e-book

www.sap-press.com/4655

Bögelsack, Baader, Prifti, Zimmermann, Krcmar

Operating SAP in the Cloud: Landscapes and Infrastructures

2016, 435 pages, hardcover and e-book

www.sap-press.com/3841

John Mutumba Bilay, Roberto Viana Blanco

SAP Process Orchestration: The Comprehensive Guide (2nd Edition)

2017, 908 pages, hardcover and e-book

www.sap-press.com/4431

Alborghetti, Kohlbrenner, Pattanayak, Schrank, Sboarina

SAP HANA XSA: Native Development for SAP HANA

2018, 605 pages, hardcover and e-book

www.sap-press.com/4500

John Mutumba Bilay, Peter Gutsche, Mandy Krimmel,
Volker Stiehl

SAP® Cloud Platform Integration

The Comprehensive Guide

Dear Reader,

On the cover of this book is a modern, futuristic-looking suspension bridge. Against a striking sky, this engineering marvel combines form and function, providing something that is used daily, even by members of the SAP PRESS team! As all of us live in the greater Boston area, we commute to the same place, to create and refine books that cater to the SAP audience, covering SAP from every direction. Cables connecting to a singular hub.

Like the bridges getting us to our jobs, SAP Cloud Platform Integration connects your cloud and on-premise applications! Even more, this star author team of John Mutumba Bilay, Peter Gutsche, Mandy Krimmel, and Volker Stiehl all came together to provide you the be-all, end-all guide to SAP Cloud Platform Integration. In these pages is the information that you need, as solid as a suspension bridge.

What did you think about *Controlling with SAP Cloud Integration: The Comprehensive Guide*? 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!

Will Jobst

Editor, SAP PRESS

willj@rheinwerk-publishing.com

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:

Copy No. j9rf-t2xn-pm4h-awcb
for personal use of
Albert-Daniel Kenel
angel.kenel@gmail.com

Imprint

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

Editor Will Jobst

Acquisitions Editor Hareem Shafi

Copyeditor Julie McNamee

Cover Design Graham Geary

Photo Credit iStockphoto.com/518857661/© gregobagel

Production E-Book Hannah Lane

Typesetting E-Book SatzPro, Krefeld (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: Bilay, John Mutumba, author.

Title: SAP cloud platform : the comprehensive guide / John Bilay, Peter Gutsche, Mandy Krimmel, Volker Stiehl.

Other titles: SAP HANA cloud integration

Description: 2nd edition. | Bonn ; Boston : Rheinwerk Publishing, 2018. |

Revised edition of: SAP HANA cloud integration / John Mutumba Bilay, Peter Gutsche, Volker Stiehl. 2016. | Includes index.

Identifiers: LCCN 2018024335 (print) | LCCN 2018026190 (ebook) | ISBN 9781493217076 (ebook) | ISBN 9781493217069 (alk. paper)

Subjects: LCSH: Cloud computing. | SAP HANA (Electronic resource)

Classification: LCC QA76.585 (ebook) | LCC QA76.585 .B55 2018 (print) | DDC 004.67/82--dc23

LC record available at <https://lccn.loc.gov/2018024335>

ISBN 978-1-4932-1706-9 (print)

ISBN 978-1-4932-707-6 (e-book)

ISBN 978-1-4932-1708-3 (print and e-book)

© 2018 by Rheinwerk Publishing, Inc., Boston (MA)

2nd edition 2018

Contents

Foreword by Björn Goerke	17
Preface	19
Acknowledgments	27

1 Introduction to SAP Cloud Platform Integration 31

1.1 The Role of SAP Cloud Platform Integration in a Cloud-Based Strategy	32
1.2 Use Cases	35
1.2.1 Point-to-Point versus Mediated Communication	36
1.2.2 Message-Based Process Integration	36
1.2.3 Cloud-to-Cloud Integration	38
1.2.4 Cloud-to-On-Premise Integration	39
1.2.5 On-Premise-To-On-Premise Integration	40
1.2.6 Hybrid Usage of Cloud and On-Premise Integration Solutions	41
1.3 Capabilities	42
1.3.1 Integration Platform-as-a-Service	43
1.3.2 Message Processing Step Types (Integration Capabilities)	44
1.3.3 Connectivity Options	45
1.3.4 Prepackaged Integration Content	48
1.3.5 Security Features	48
1.3.6 High Availability	49
1.3.7 Integration Design and Monitoring Tools	50
1.4 Editions	50
1.5 Summary	51

2 Getting Started 53

2.1 Architecture Overview	53
2.1.1 Virtual and Clustered Integration Platform	54
2.1.2 Detailed Structure of a Cluster	59

2.1.3	Secure Communication	63
2.1.4	Implementation of Message Flows	63
2.1.5	Architecture Summary	68
2.2	Tools and Processes	71
2.2.1	Tools	71
2.2.2	Processes	78
2.3	Running Your First Integration Scenario	80
2.3.1	Demo Scenario and Landscape	80
2.3.2	Prerequisites	81
2.3.3	Set Up the Landscape and the Technical Connections	81
2.3.4	Develop the Integration Flow	84
2.3.5	Creating and Deploying a User Credentials Artifact	99
2.3.6	Import Certificate Required by the Mail Server into Keystore	101
2.3.7	Send the SOAP Message	103
2.3.8	Monitor the Message	106
2.4	Summary	111

3 Integration Content Catalog

3.1	Introduction to the Integration Content Catalog	113
3.2	Terms and Conditions of Using Prepackaged Integration Content	117
3.2.1	Quick Configure versus Content Edit	117
3.2.2	Notify about Update (Manual Update)	118
3.2.3	Automatic Update	120
3.3	Consuming Prepackaged Content	121
3.3.1	Search in the Integration Content Catalog	122
3.3.2	Import Prepackaged Integration Content	127
3.3.3	Modify or Configure the Integration Package	129
3.3.4	Deploy Content	136
3.4	Prepackaged Content Provided by SAP	137
3.4.1	Content for SAP SuccessFactors	138
3.4.2	Content for SAP Cloud for Customer	140
3.4.3	Content for Integrating with SAP C/4HANA	141

3.4.4	Content for Integrating with the Ariba Network	144
3.4.5	Content for Globalization Scenarios	146
3.5	Creating Your Own Content Package	147
3.6	Summary	150

4 Basic Integration Scenarios

4.1	Working with SAP Cloud Platform Integration's Data Model	153
4.1.1	Message Processing: The Apache Camel Framework	155
4.1.2	Exercise: Working with Camel's Message Model	158
4.1.3	Connecting and Configuring a Sender with an Integration Flow	161
4.1.4	Adding and Configuring Steps in the Integration Flow	164
4.1.5	Checking Configuration Using the Problems View	168
4.1.6	Running the Integration Flow	170
4.1.7	Troubleshooting	173
4.2	Using Externalization to Enable Easy Reuse of the Integration Flow	175
4.2.1	Externalize	176
4.2.2	Configure and Run the Scenario	181
4.3	Content Enrichment by Invoking an OData Service	183
4.3.1	The Target Scenario	184
4.3.2	Invoking an OData Service	185
4.3.3	Configuring the OData Connection	187
4.3.4	Creating the Resource Path Using the Query Editor	190
4.3.5	Using the Content Enricher Step	195
4.4	Working with Mappings	201
4.4.1	The Scenario	203
4.4.2	Adding and Using Resources via the Resources View	205
4.4.3	Applying the Mapping Step in the Message Processing Chain	210
4.4.4	Using Value Mapping to Enhance Your Scenario	218
4.5	Defining and Providing an OData Service	224
4.5.1	The Target Scenario	224
4.5.2	Providing an OData Service	225

4.6 Working with an Aggregator	238
4.6.1 Sample Scenario	240
4.6.2 Sending Messages via SoapUI	243
4.6.3 Viewing the Aggregated Message	249
4.7 Summary	250

5 Advanced Integration Scenarios

5.1 Message Routing	251
5.1.1 The Scenario	252
5.1.2 Configuration of the Content-Based Router	254
5.1.3 Running the Content-Based Router Scenario	259
5.2 Working with Lists	262
5.2.1 The Scenario	262
5.2.2 Configuring the Integration Flow	264
5.2.3 Running the Integration Flow	275
5.2.4 Enriching Individual Messages with Additional Data	279
5.3 Asynchronous Message Handling	281
5.3.1 Synchronous versus Asynchronous Communication from SAP Cloud Platform Integration's Perspective	283
5.3.2 Adding an Asynchronous Receiver	295
5.3.3 Routing a Message to Multiple Receivers Using the Multicast Pattern	299
5.4 Reliable Messaging Using the JMS Adapter	308
5.4.1 Asynchronous Decoupling of Inbound Communication	308
5.4.2 Configure Retry for Multiple Receivers	322
5.4.3 Configure Explicit Retry with Alternative Processing	330
5.5 Summary	338

6	Special Topics in Integration Development	341
6.1	Timer-Based Message Transfer	341
6.1.1	The Scenario	342
6.1.2	Configuring a Timer-Based Integration Flow	343
6.1.3	Running the Integration Flow	348
6.2	Using Dynamic Configuration via Headers or Properties	349
6.2.1	An Integration Flow with a Dynamically Configured Attribute	351
6.2.2	Monitoring Dynamically Configured Attributes at Runtime	356
6.2.3	Using Predefined Headers and Properties to Retrieve Specific Data Provided by the Integration Framework	360
6.3	Structuring Large Integration Flows Using Local Processes	365
6.3.1	Taking Hold of Complexity by Modularization	366
6.3.2	Configuring the Collaboration between Parent and Child Processes	368
6.3.3	Using Exception Subprocesses	375
6.4	Connecting Integration Flows Using the ProcessDirect Adapter	378
6.4.1	Use Cases for the ProcessDirect Adapter	380
6.4.2	A Simple Example	381
6.4.3	Dynamic Endpoint Configuration with the ProcessDirect Adapter	383
6.5	Versioning and Migration of Integration Flows	388
6.5.1	Integration Flow Component Versions	388
6.5.2	Upgrading an Integration Flow Component	391
6.5.3	Downgrading Integration Content for SAP Process Orchestration	395
6.6	Transporting Integration Packages to Another Tenant	403
6.6.1	Manually Transporting Integration Packages	403
6.6.2	Transporting Integration Packages Using CTS+	405
6.6.3	Transporting Integration Packages Using the Cloud-Based Transport Management Service	405
6.7	Using the Adapter Development Kit	411
6.7.1	The Adapter Development Kit (ADK)	411
6.7.2	Installing the Adapter Development Kit	412
6.7.3	Developing a Sample Adapter	415
6.8	Best Practices for Integration Flow Development	427
6.9	Summary	429

7	B2B Integration with SAP Cloud Platform Integration	431
7.1	B2B Capabilities in SAP Cloud Platform Integration: Overview	432
7.2	Defining Interfaces and Mappings in the Integration Content Advisor	434
7.2.1	Create Message Implementation Guidelines	435
7.2.2	Configure Mapping Guidelines	447
7.2.3	Generate the Runtime Content	450
7.3	Configure a B2B Scenario with AS2 Sender and IDoc Receiver Adapters	451
7.3.1	Create an Integration Flow Using a Template	451
7.3.2	Configure AS2 Sender Channel to Receive EDI Messages	460
7.3.3	Add an IDoc Receiver	466
7.3.4	Configure Acknowledgement Handling	471
7.4	Using the Partner Directory for Partner-Specific Configuration Data	482
7.4.1	Concept of Partner Directory	482
7.4.2	Use a Receiver Endpoint URL Dynamically in the Integration Flow	484
7.4.3	Store the Partner-Specific Endpoint URL in Partner Directory	488
7.5	Summary	492
8	SAP Cloud Platform Integration Operations	493
8.1	Operations: Overview	494
8.2	Monitoring Integration Content and Message Processing	496
8.2.1	Manage Integration Content	498
8.2.2	Log Configuration	502
8.2.3	Monitor Message Processing	503
8.2.4	Managing Tiles	519
8.3	Manage Security	522
8.3.1	Maintain Security Material	524
8.3.2	Manage the Keystore	527
8.3.3	Maintain Certificate-to-User Mappings	536
8.3.4	Test Outbound Connectivity	538

8.4	Manage Temporary Data	549
8.4.1	Monitor Data Stores	550
8.4.2	Monitor Variables	553
8.4.3	Maintain Message Queues	555
8.4.4	Maintain Number Ranges	563
8.5	Access Logs	565
8.5.1	Monitor Audit Log	566
8.5.2	Check System Log Files	568
8.6	Manage Locks	570
8.7	Summary	573

9 Application Programming Interfaces 575

9.1	Introduction	575
9.2	Java APIs Provided by SAP Cloud Platform Integration	576
9.3	Using the Java API in a User-Defined function	579
9.4	Using the Script Step	584
9.4.1	Target Scenario	585
9.4.2	Enhancing the Integration Flow	586
9.5	OData API	590
9.5.1	SAP API Business Hub	594
9.5.2	Cross-Site Request Forgery Token Handling	602
9.5.3	Monitoring Message Flows Using the API	605
9.5.4	Managing Deployed Integration Content Using the API	611
9.5.5	Managing Log Files Using the APIs	614
9.5.6	Managing Message Store Entries Using APIs	616
9.5.7	Managing Security Material Using the API	620
9.5.8	Managing the Partner Directory Using the API	621
9.6	Using SAP Cloud Platform Integration with SAP Cloud Platform API Management	623
9.6.1	Establish a Connection between SAP Cloud Platform Integration and SAP API Management	625

9.6.2	Provision Application Programming Interfaces	629
9.6.3	Consume the Application Programming Interface	638
9.7	Summary	641

10 SAP Cloud Platform Integration Security

10.1	Technical System Landscape	644
10.1.1	Architecture	644
10.1.2	Network Infrastructure	648
10.1.3	Data Storage Security	650
10.1.4	Data Protection and Privacy	651
10.1.5	Physical Data Security	654
10.2	Processes	655
10.2.1	Software Development Process	655
10.2.2	Provisioning and Operating SAP Cloud Platform Integration Clusters by SAP	656
10.2.3	Setting Up Secure Connections between the Tenant and Remote Systems	657
10.3	User Administration and Authorization	657
10.3.1	Technical Aspects of User Management	658
10.3.2	Personas, Roles, and Permissions	658
10.3.3	Managing Users and Authorizations for a SAP Cloud Platform Integration Subaccount	660
10.4	Data and Data Flow Security	665
10.4.1	Basic Cryptography in a Nutshell	666
10.4.2	Transport-Level Security Options	671
10.4.3	Authentication and Authorization	673
10.4.4	Securely Connecting a Customer System to SAP Cloud Platform Integration (through HTTPS)	683
10.4.5	Setting Up a Scenario Using OAuth with the Twitter Adapter	692
10.4.6	Message-Level Security Options	699
10.4.7	Designing Message-Level Security Options in an Integration Flow ...	703
10.5	Keystore Management	721
10.5.1	Using X.509 Security Material for SAP Cloud Platform Integration	722

10.5.2	Managing Security Material in the Tenant Keystore	725
10.5.3	Managing the Lifecycle of Keys Provided by SAP	729
10.6	Summary	736

11 Productive Scenarios Using SAP Cloud Platform Integration

11.1	Integration of SAP Cloud for Customer and SAP ERP	737
11.1.1	Technical Landscape	738
11.1.2	Example Adapter Configurations	740
11.2	Integration of SAP Cloud for Customer with SAP S/4HANA Cloud	743
11.3	Integration of SAP Marketing Cloud and Various Applications	744
11.4	Integration of SAP SuccessFactors and SAP ERP	745
11.4.1	Technical Landscape	746
11.4.2	SAP SuccessFactors Adapter	747
11.5	Integration of SAP Applications with the Ariba Network	750
11.5.1	Technical Landscape	752
11.6	Summary	752

12 Summary and Outlook

12.1	Multi-Cloud Support	756
12.2	Integration Content Management	758
12.3	Predefined Integration Content	759
12.4	New Connectivity Options	759
12.5	Business-to-Business Support	760
12.5.1	Further Adapters	760
12.5.2	Enhancements of the Integration Content Advisor	760
12.5.3	Trading Partner Management	761

12.6 SAP Cloud Platform Integration API	761
12.7 Connectivity	762
12.8 Security	762
12.8.1 Tenant Keystore Management	762
12.8.2 Compliance with Security Standards	763
12.9 Harmonization of SAP Cloud Platform Integration with Other Services ...	763
12.10 Summary	764

Appendices

A Abbreviations	767
B Literature	775
C The Authors	779

Index	781
Service Pages	I
Legal Notes	II

Foreword by Björn Goerke

Undoubtedly, the rapid succession of innovations we see in the field of information technology is fundamentally changing how we do business, how we communicate with each other, even how we live. Software is increasingly driving the world and everything is digitizing. Self-driving cars, refrigerators that order food on their own, robots that take over complex tasks previously only performed by humans—just a few years ago, this would have sounded like science fiction, but now these things are discussed seriously and debated publicly. The future of enterprise IT has to be agile, yet robust. Distributed, while interconnected. Open, yet secure. Simple, but intelligent.

When the world's master of the Go game, the Chinese Ke Jie, was defeated by Google's AlphaGo algorithm in 2017, software capable of learning without being explicitly programmed for every eventuality clearly moved from the realm of theory to practical reality, highlighting the fact that such technological changes will inevitably shape all our futures.

In times such as these, it is no longer an option for enterprises to simply keep pace with new technology. Instead it is imperative to forge ahead, embrace innovations and build new businesses and processes that maximize the advantages they offer. A new IT era is upon us.

No IT company “knows” business processes better than SAP, the world's largest vendor of business software. And no one knows better how crucial it is to have a reliable, experienced and responsive partner to help you make the most of every innovation and opportunity. Agility will be key to winning in this new era. With SAP Cloud Platform (SAP CP), you can benefit from THE agile platform-as-a-service that provides you with the capabilities to develop, extend, and build intelligent applications using SAP's cloud infrastructure.

This new era will lead to new competitive dynamics and business models, and will introduce new disruptive players, who will be quick to harness innovation. To supercharge our continuous journey to innovate the future, we will have to embrace extreme openness in our technology stack and culture.

Of course, *integration* was and still is a key challenge facing forward-looking enterprises. Companies must be able to run, monitor, and control business processes distributed over heterogeneous system landscapes—both in the cloud and on premise.

Integration will only become more significant as technological developments bring new software systems to connect to and integrate.

SAP Cloud Platform Integration—Cloud Integration—as SAP’s integration service on top of SAP CP, is the way to go. It comes with a rich set of tools that enable you to design, build, and operate integration scenarios based on the exchange of messages within distributed landscapes. It supports various protocols and standards to connect SAP and non-SAP systems securely, in the cloud and on premise. Cloud Integration is a unique platform that enables you to adopt the cloud world. Furthermore, it provides you with a rich variety of predefined integration content that supports out-of-the-box integration of SAP systems with each other as well as with third-party systems.

The platform is continuously innovating, with updates released every two weeks with zero downtime for your business. Therefore, since the release of the first edition of this book, the Cloud Integration team has enhanced the portfolio of the product by many new features.

Curious to find out more? This new edition explains all these innovations in detail, along with an in-depth introduction to the product portfolio and the architecture underlying it. However, like the first edition, the book is suited for the beginner who wants to dive in and get started with the product, with easy-to-follow instructions on how to set up and run your first integration scenario. At the same time, this second edition offers a deeper look into the details of integration design and touches on more sophisticated scenarios. As a result, the advanced user also can rely on it as a valuable reference.

This book is sure to be a good companion on your way to becoming an integration expert. Enjoy reading!

Björn Goerke

Chief Technology Officer at SAP and President of SAP Cloud Platform

Preface

The IT landscapes in today's companies are getting more complex every day. With the advent of cloud computing, the need for integration between on-premise applications and cloud solutions, or between cloud applications, becomes apparent. By reading this book, you will learn how SAP Cloud Platform Integration can help you solve your integration challenges.

Enterprise application integration (EAI) has a long history. The need for easy data exchange came up early, with the first computer systems. Whether it was the transfer of master data, such as customer or product data, or the transmission of transactional data, such as orders or invoices, systems of importance always required some sort of communication and integration. Interestingly, this communication wasn't restricted to systems and applications *within one company*. The demand for inter-company data transfer increased the integration challenge even further.

With the arrival of cloud computing, we face a new dimension to the integration domain, as on-premise and cloud applications need to work seamlessly with each other. Companies selling standard software for both worlds, on-premise and on-demand, face an additional challenge: their customers expect seamless data exchange between their applications out of the box. As SAP's strategy clearly focuses on becoming *the cloud-company* for business software, the need to supply an easy-to-use integration solution running in the cloud became apparent. SAP had already gathered experience in building integration software with SAP Process Integration. However, for this new integration era, the good old SAP Process Integration solution wasn't suitable, as it didn't support typical cloud qualities such as multi-tenancy, data isolation, rolling software updates, zero downtime, or cloud elasticity. Instead of further investing in SAP Process Integration and trying to make it "cloud-ready," SAP decided to deliver a completely new integration solution called SAP Cloud Platform Integration. SAP Cloud Platform Integration was developed from scratch on top of the SAP Cloud Platform, with typical cloud integration scenarios in mind. With this approach, SAP is combining the best of both worlds: on-premise integration with its

rock-solid SAP Process Integration product and cloud-based integration with SAP Cloud Platform Integration. With this separation, customers have the freedom of choice when considering their particular needs.

Structure of the Book

This book introduces SAP Cloud Platform Integration and covers a wide range of topics. We will begin with [Chapter 1](#) and a discussion of how SAP Cloud Platform Integration, as the prime integration point between disparate cloud and on-premise systems, fits into SAP's overall strategy to become *the cloud-company* for business software. In addition, we will present the main use cases of SAP Cloud Platform Integration, such as cloud-to-cloud or cloud-to-on-premise integration. [Chapter 1](#) closes with an explanation of SAP Cloud Platform Integration's major capabilities.

As we wanted to deliver a practical book with lots of examples, [Chapter 2](#) dives directly into the hands-on experience. After introducing the architecture of SAP Cloud Platform Integration and the new vocabulary you need to get used to, we guide you step-by-step through a very simple scenario. You will work with the *Web UI* as the central SAP Cloud Platform Integration tool used to model, deploy, run, and monitor your integration scenarios, or *integration flows* in Cloud Integration's nomenclature. This chapter lays the foundation upon which all further chapters and exercises will be built.

Besides modeling integration scenarios from scratch, SAP Cloud Platform Integration also comes with prepackaged integration content. Remember, SAP's customers expect a smooth message exchange between SAP's on-premise and cloud solutions, as a single vendor ships both types of products. This data exchange can only work if SAP delivers pre-configured integration content running on top of SAP Cloud Platform Integration out-of-the-box. The delivered integration flows seamlessly glue the systems together without running costly integration projects, which were once a necessity. This prepackaged integration content is already available in the *Integration Content Catalog*. [Chapter 3](#) explains how you can benefit from the Integration Content Catalog and describes in detail, for example, the integration content for seamlessly connecting SAP SuccessFactors and SAP ERP Human Capital Management. At the end of the chapter, we will also explain how you can develop your own integration content and make it available in the Integration Content Catalog—an offering that is of special interest for SAP's partner ecosystem.

Chapter 4, **Chapter 5**, and **Chapter 6** focus on developing custom integration flows in case the prepackaged content provided in the Integration Content Catalog isn't sufficient for your business needs. In particular, we'll use step-by-step guides on how to apply the various step types that are part of SAP Cloud Platform Integration's web-based modeling environment. Over the course of these three chapters, you will learn how to:

- Work with SAP Cloud Platform Integration's data model.
- Enrich incoming data with data from an external OData source.
- Map data between different message interfaces.
- Route messages to the right receiver depending on the message's content (i.e., *content-based routing*).
- Influence the evaluation sequence of conditions for the content-based router.
- Handle messages comprising a list of items (such as order items); splitting and merging the individual list items are of particular interest in this section.
- Influence the execution of synchronous and asynchronous scenarios.
- Use Java Message Service (JMS) queues to temporarily store messages on the Cloud Integration platform and, that way, to asynchronously decouple inbound from outbound communication.
- Schedule timer-based integration flows, which run at pre-defined times and/or at pre-defined intervals.
- Dynamically configure integration flow parameters using headers and properties.
- Structure large integration scenarios based on modularization and the usage of local integration flows.
- Exchange data between parent and child integration flows.
- Directly connect integration flows by using the ProcessDirect adapter.
- Understand the versioning concept of the cloud-based integration flow modeler and migrate integration flows to another version.
- Transport integration content across tenants.
- Develop your own adapters by using the *Adapter Development Kit* (ADK).
- Last but not least, get an overview of some best practices for integration flow design to make sure to optimize reliability and performance of message processing.

Chapter 7 covers the capabilities of SAP Cloud Platform Integration to support business-to-business (B2B) integration. SAP has developed various new components to support such use cases, such as the Integration Content Advisor, the Partner Directory, and various new adapters and integration flow step types. This chapter guides you along one end-to-end scenario through the usage of these components and tools.

Chapter 8 continues our journey with the operational aspects of SAP Cloud Platform Integration. As the previous chapters have shown you how to design integration flows to cover use cases with increasing complexity, you—the integration developer—like to monitor how these integration flows actually process messages on your SAP Cloud Platform tenant. Monitoring messages is one task. But there are additional tasks you can accomplish using the Monitor section of the Web UI, such like managing security artifacts (for example, the keys and certificates contained in the tenant keystore), and data storages on your tenant. This chapter covers all these aspects in detail.

So far, we have in detail described how people can work with the user interface of SAP Cloud Platform Integration (basically, the Web UI) to perform tasks such like integration flow development or monitoring messages. However, SAP Cloud Platform Integration provides also the option to access integration-related artifacts based on an application programming interface (API). The available APIs are described in detail in **Chapter 9**. This chapter also shows you how you can use Cloud Integration APIs together with SAP Cloud Platform API Management, a dedicated service of SAP Cloud Platform that helps you to develop, manage, and publish your own APIs.

So far, we have completely left out security aspects from our discussion. However, when it comes to cloud computing, security is one of the top-ranked requirements from customers, whether they are addressing the service provider hosting SAP Cloud Platform Integration or the integration developer running scenarios on top of it. We have decided to collect all security-relevant topics in one chapter, rather than spread them across the book. Therefore, **Chapter 10** will be your one-stop shop for all security-related questions. The chapter summarizes the measures taken by SAP to protect your data at the highest level and shows what you can do to maximize the security level of your integration scenarios. Keeping with our habit of providing hands-on examples, this chapter also contains guides that show you how to build simple integration flows that contain features such as digital encryption or authentication.

We now approach the end of the book. [Chapter 11](#) looks at already-running productive scenarios using SAP Cloud Platform Integration. We will mainly focus on the specifics of the following scenarios:

- Integration of SAP Cloud for Customer and SAP ERP
- Integration of SAP Cloud for Customer with SAP S/4HANA Cloud
- Integration of SAP Marketing Cloud and various applications
- Integration of SAP SuccessFactors and SAP ERP
- Integration of SAP applications with the Ariba Network

Once finished with this chapter, you will understand how of SAP Cloud for Customer plays a crucial role as part of SAP's classical and new business applications.

Finally, [Chapter 12](#) concludes the book with an outlook comprising the roadmap for SAP Cloud Platform Integration. By reading this chapter, you will receive an impression of how of SAP Cloud for Customer will evolve over time and how of SAP Cloud for Customer grows even more important for SAP's overall company strategy.

Sample Applications

Over the course of this book, we will be developing many sample applications that demonstrate the key concepts of of SAP Cloud for Customer in context. Some of these applications require certain artifacts such like files that contain web service descriptions. To make it easy for you to set up these applications, we provide these files ready-to-download in the supplemental materials of this book. You can download them at www.sap-press.com/4650.

You can generally find instructions for installing and deploying these applications within the chapters that cover them. If you run into any problems with the examples, you can email the authors directly at: johnbilay@rojoconsultancy.com, mandy.krimmel@sap.com, and peter.gutsche@sap.com.

The downloaded archive—when unpacked—will have a directory structure which is oriented along the chapters where the associated files are required:

- [Chapter 2](#): The file `SendOrder_Async.wsdl` contains the Web Services Description Language (WSDL) file that defines an input message (asynchronous interface) used in various sample integration flows.

- **Chapter 4:** The file **GetOrderShipDetails_Sync.wsdl** contains the Web Services Description Language (WSDL) file that defines an input message (synchronous interface) used in a sample integration flow.
- **Chapter 5:** The file **SendOrderList_Async.wsdl** contains the Web Services Description Language (WSDL) file that defines an input message (asynchronous interface) used in a sample integration flow.
- **Chapter 7:** The following files are required to set up the B2B scenario:
 - The archive **EDI_IDoc_Template.zip** is the template to be used to create the integration flow.
 - The folder **ASC_X12_to_SAP_IDOC_Purchase_Order_Mapping** contains *.xsd and *.xsl files generated from Integration Content Advisor.
 - The file **GroovyScript.txt** contains the coding for the groovy script flow step to retrieve configuration data from partner directory.
 - File **850 - Purchase Order.txt** contains the test payload for the scenario.
 - File **850 - Purchase Order - Technically incorrect.txt** contains an incorrect payload for the scenario to test the error case.
- **Chapter 10:** The file **SendOrder_Async.wsdl** (same file as used for Chapter 2) contains the Web Services Description Language (WSDL) file that defines an input message (asynchronous interface) used in various sample integration flows.

Who This Book Is For

This book addresses a broad audience, from integration architects, integration consultants, and integration developers, to technical-oriented business users, project leaders, and managers who want to understand how SAP Cloud Platform Integration can support either their journey into the cloud or how it can solve integration challenges related to integrating cloud solutions (on-premise-to-cloud or cloud-to-cloud). The reader, ideally, should be familiar with basic concepts regarding enterprise application integration and messaging, as this book will not cover those concepts. In addition, a basic understanding of Java, scripting languages, and enterprise integration patterns is beneficial. However, note that no knowledge of SAP Cloud Platform Integration or SAP Process Integration is required. You will receive everything you need to begin productive work with Cloud Integration, from designing and running a simple integration flow, up to implementing complex integration patterns from this book.

Now that you have an understanding of the book's contents, and for whom the book was written, we don't want to lose any more time getting started. We wish you an enjoyable ride!

Acknowledgments

Writing a book is always a challenging task. And it would be close to impossible without the help of many good friends and colleagues. This chapter is for those who supported us in one way or another.

We would like to start by thanking the team at Rheinwerk Publishing for all their support throughout the project. We are very thankful for your support and your team spirit. You have made it all possible. We would like, in particular, to thank Hareem Shafi from Rheinwerk Publishing for encouraging us to take up the task of writing a second edition of the book and for helping us set up the project. We would like to sincerely thank Will Jobst for accompanying us throughout the whole project. Will was always there when we needed support and quickly answered our many questions. Without his help during the whole writing process, it would not have been possible to bring the project to a successful end. Thank you for that!

We would like to thank Björn Goerke for recognizing this book with his foreword.

Sindhu Gangadharan encouraged us to take over the task to write a second edition of the book and, like she did for the first edition, supported our project.

Anette Asmus and Anita Riegel critically read some text passages and helped us to refine them at the final stage of writing this book.

We would like also to thank Volker Stiehl, coauthor of the first edition of the book, for his dedication in shaping the initial version of the book and for his insights into the topic of integration. Unfortunately, you couldn't join us in working on the second edition, Volker, but your contributions were invaluable in making the book what it is today.

We would also like to sincerely thank all the people who have contributed and reviewed different chapters of this SAP Cloud Platform Integration book. We would also like to acknowledge many others that we did not mention their name for their direct and indirect support and interest on the book.

John Mutumba Bilay

I have enjoyed the teamwork and collaborative spirit during this book project. It has been a true honor to work alongside Peter and Mandy. Your detailed and critical look at every topic made a difference. Thank you for that.

I would like to personally thank and express my gratitude to the many people who contributed to this book in different ways:

- Sabarish T. S. and Deepak Govardhanrao Deshpande for their support with questions related to OData provisioning. Your contribution is highly appreciated.
- Sujit Hemachandran for his valuable contribution and review of prepackaged content.
- Sandra Voges for her valuable inputs and review of [Chapter 9](#).
- My gratitude also goes to the entire team of Rojo Consultancy B.V. for their help and consideration during the book writing process.

Finally, a big thank you to my wife Hermien and my boys Ralph, Luc, and Ruben for their loving support, patience and encouragement. You have made me stronger, better and more fulfilled than I could have ever imagined. During the process of writing this book, you have allowed me to skip some playtime to work on the book instead! I love you all to the moon and back.

Peter Gutsche

Like when writing for the first edition of this book, I experienced continuous and professional support from my colleagues in the SAP Cloud Platform Integration development teams, the user assistance team, and the product management team. Without the support of all of you this project would have not been possible. However, listing all your names would go beyond the scope of this section.

I would like in particular to thank the following colleagues who were involved in discussions or text reviews directly related to the book project:

- Peter Goebel helped me for any technical question related to the SAP Cloud Platform Integration system.
- I would like to thank Gunther Stuhec for sharing his knowledge of business-to-business integration in a couple of inspiring coffee corner chats.
- Franz Forsthofer, Christian Becker, Frantisek Deglovic, and Martin Matejcek helped my clarifying questions related to SAP Cloud Platform Integration security.
- Aurelian Stratica was a critical reader of the first edition of the book and shared his feedback.
- Finny Babu reviewed [Chapter 6](#) and provided valuable input and feedback, in particular, for the topic of versioning and migration.

- Ralf Belger, Boris Zarske, VishnuPrasath Dhayanithi, Sabarish T. S., and Dimitar Aleksandrov helped me to find my way through the topic of integration content transport. I could count on your professional support when setting up a scenario using the beta version of the SAP Cloud Platform Transport Management Service.
- For the adapter development topic, I got input and support from Kumar Amar, Gopalkrishna Kulkarni, Appala Naidu Uppada, and Sharath Sasi.
- Markus Beier and Stephan Siano helped my clarifying some questions related to Camel headers and properties.
- Udo Paltzer and Andreas Quenstedt were there for all my questions about the product strategy and sharing their perspective on the outlook of the product.
- Divya Mary provided valuable feedback to [Chapter 12](#). Thank you!
- Abinash Nanda helped clarifying questions related to integration scenarios including SAP Cloud for Customer.

I would like to thank my co-authors for this second book edition, Mandy and John, for their continuous support and team spirit. It was always possible to contact you and get your feedback in time when I came over a question. Your input and feedback to the individual chapters I wrote were invaluable. It was a real pleasure to work with you!

My manager Stefanie Schmitt supported me throughout the whole time of writing this book. Thank you for that!

Sherry Föhr was always there when I had questions related to the nuances of the English language and helped out with her expertise.

Finally, and of capital importance: I would like to thank Isabelle Krys for her encouragement and forbearance throughout the process of writing. Without you, it would have been hard for me to keep up with the project. Thank you!

Mandy Krimmel

Working on a book project like this requires helpful colleagues and understanding friends in the background. Working at SAP, I have the advantage of having many experienced, helpful and motivated people around to drill with questions. I would like to express my special thanks to the following colleagues for their support:

- The B2B chapter would not have been possible without the help of the development team of the Integration Content Advisor. Special thanks to Prashantha Halmuttur Lakshminarayana for helping to get the EDI template running. Jörg

Ackermann, Gunther Stuhec, and Marton Luptak supported me in understanding and describing the features of the Integration Content Advisor.

- Appala Naidu Uppada supported me with his expert knowledge of the AS2 adapter and the EDI flow steps. He helped me getting the Mendelson AS2 tool setup and running and had to answer lots of B2B-specific questions.
- Franz Forsthofer contributed with his comprehensive blogs about the partner directory and reviewed the respective section in the book.
- Stephan Siano and Jörg Kessler answered countless questions about the JMS functionality and JMS resources.
- Ines Ahrens and Maik Keller supported me with their expert knowledge in the operations area of Cloud Integration.
- Peter Göbel often saved my day by helping me with the setup and maintenance of my test cluster.

Gabriela Vittek, my manager, encouraged me to bring my knowledge into this book project. She supported me throughout the whole project and always had open ears and constructive suggestions when something did not run as planned.

Not to forget my two co-authors: Peter and John! It was a pleasure working with you! I enjoyed the constructive and goal-oriented discussions and the helpful attitude. I'm proud that I was part of this team!

Above all, none of this would have been possible without my husband Stefan and my daughter, who supported me throughout, even when deprived of my time and attention. Thank you!

Finally, we want to thank you, our readers, for choosing and purchasing this book. You want to learn more about SAP Cloud Platform Integration, and we hope you receive all the information needed in the following chapters to drive your personal education forward. Enjoy reading, and we hope you benefit from SAP Cloud Platform Integration in your next cloud integration project.

John Mutumba Bilay, Peter Gutsche, and Mandy Krimmel

Chapter 1

Introduction to SAP Cloud Platform Integration

This chapter explains how SAP Cloud Platform Integration fits into SAP's overall cloud strategy and its main use cases.

It's no secret anymore: the world is going digital! The convergence of trends, such as cloud computing, social networks, mobility, the Internet of Things, blockchain, hash-graph, and the resulting big data, is changing how we conduct business. We're increasingly experiencing this change in our personal lives, but these trends are quickly moving into the enterprise world, as well. Just take a look at how software is consumed today: more and more businesses are choosing to have their software delivered as a service and hosted in huge data centers. While on-premise environments aren't going anywhere soon, the industry is definitely seeing a shift to increased cloud adoption. Subscription-based licensing and central hosting are simply how people want to receive software these days. In this changing world of software delivery, the importance of simplicity can't be understated. All of these converging technological forces are creating a much more complex world. Hence, the drive to simplify experiences for software users is of paramount importance.

SAP, the world's largest provider of business software, has recognized the technological shift and is continuously expanding its offering of cloud software. However, SAP customers who have invested in on-premise landscapes in recent years can't be expected to immediately abandon their solutions and move to the cloud. This is where SAP Cloud Platform Integration comes into play, as the solution of choice for companies that are looking to bring their cloud and on-premise technologies together.

That brings us to the topic of the upcoming section: the role SAP Cloud Platform Integration plays in a company's cloud strategy. We'll also discuss the main use cases of SAP Cloud Platform Integration and provide an introduction to its various capabilities in this chapter.

Note

SAP Cloud Platform Integration is the new name for a product which was previously known as SAP HANA Cloud Integration (SAP HCI). The product was renamed in the beginning of 2017 as part of a general rebranding of SAP's cloud offering.

1.1 The Role of SAP Cloud Platform Integration in a Cloud-Based Strategy

As SAP moves to the cloud, hybrid deployments are going to play an increasingly major role in businesses. Most customers who run businesses today must, without a doubt, maintain existing IT landscapes. It's simply unreasonable to assume that cloud deployments will completely replace current environments. Consequently, successful companies look for solutions that will help them combine their current environments, which are typically on premise, with cloud-based applications (where it makes sense). Hybrid deployments will be the solution of choice for most companies in the years to come. This is where the market is heading, and this is where SAP sees investments in IT while moving to the cloud. However, hybrid landscapes automatically mean integration needs: on-premise and cloud applications need to exchange data with each other. This is exactly where an integration platform such as SAP Cloud Platform Integration fills the gap: it's responsible for reliable message exchange between systems. However, before we dive into the details of SAP Cloud Platform Integration and its strategic role in SAP's future plans, let's understand what motivates companies to implement cloud strategies.

The move to the cloud is made for good reasons. [Figure 1.1](#) summarizes key business benefits of cloud computing, categorized into four major pillars.

When we talk about the cloud, we're actually thinking about the benefits shown in [Figure 1.1](#). First and foremost, it's all about faster deployments. By deployment, we mean the first setup and customizing of the software that makes it ready for productive usage. Deployment in the old days used to take months, even years. With cloud solutions, deployment time is reduced drastically, sometimes to weeks or days. This trend is accompanied by a mobile user experience. The cloud is consumer-grade, it's beautiful, and it's flexible. Today's users expect to use software instantly and find it easy to adapt to without yesterday's long-lasting training sessions and expensive courses.

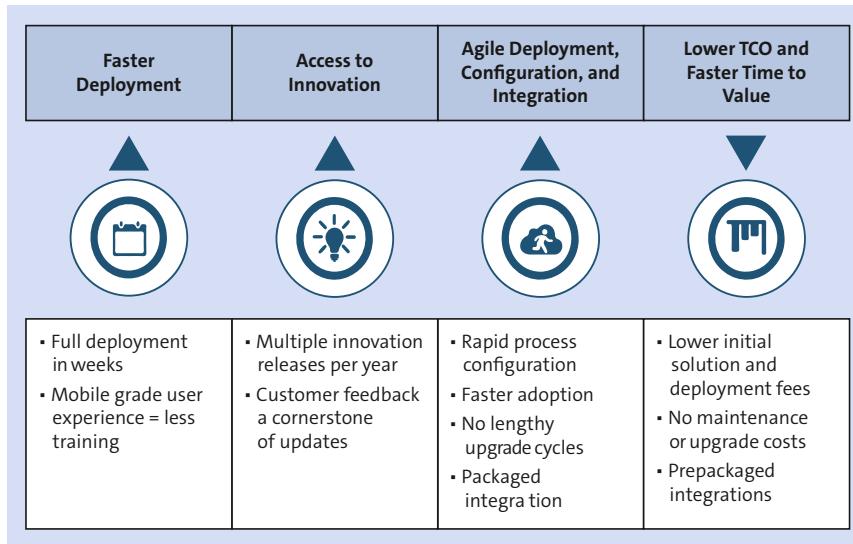


Figure 1.1 Business Benefits of Cloud Computing

Access to innovation, the second pillar in [Figure 1.1](#), is another important benefit of cloud computing. We're moving away from upgrade cycles that once took years, to at least quarterly upgrade cycles with most of today's software-as-a-service (SaaS) offerings. In many solutions, the innovation cycle may even be reduced to a monthly update. This results in the delivery of quarterly (or monthly) innovations to all users of a solution at the same time, and allows the software to always remain up to date. Consequently, all users are consistently working with the latest version of their software.

This leads directly to the next advantage of cloud computing: because all users are working with the same software at the same time, a short feedback cycle is ensured. Potentially millions of people are working with the software and will report their experiences directly to the vendor. This regular feedback cycle helps the software become better and better each day, as the vendor can react to issues almost immediately. Compare this cycle to the old days: after the vendor sold the solution, the deployment, as well as the upgrade, depended on the customer's planning, and the vendor never really received feedback about the software's usage. Today, cloud solutions allow innovations to be implemented faster, driving the quality of the software forward.

In summary, deployment and upgrade processes are becoming more and more agile, resulting in rapid configurations for faster adoption of the software without lengthy upgrade cycles. However, one problem needs to be solved in this new world of cloud-based software delivery: How do the systems residing in the cloud and on premise talk to each other? Fortunately, prepackaged integration content comes to the rescue. As the vendors of these solutions know their software best, they also know how to best connect them with each other. This is the home turf of SAP Cloud Platform Integration, as we'll see in a moment. Integrations between cloud systems, or between cloud and on-premise systems, are increasingly being delivered prepackaged. This means, in essence, the way those systems communicate and exchange data with each other can be defined in advance by the software vendor so that the consumer needs just a few configuration steps to make them executable: plug and play at its best. Customers have the choice of moving only certain functionality to the cloud. Out-of-the-box integrations are then used to allow for a smooth transition from the on-premise world to the cloud world at the speed the customer chooses.

Finally, we've heard a lot about lower total cost of ownership (TCO) and faster time to value of cloud solutions. This is the result of the advantages pointed out before: customers don't have to spend money on hardware and software hosted in their own data centers anymore. They can now outsource these tasks to the solution provider. Customers optimize their TCO due to lower initial solution and deployment fees, as well as no longer needing to shoulder the maintenance and upgrade costs. So, if you think about TCO, typically cloud solutions do very well in that regard.

The advantages are obvious: these benefits of cloud deployments free up time, money, and energy that can be spent on new innovations and new engagement models with your customers, partners, suppliers, and employees. You can now dedicate more time to serving your customers best. So remember, when you think about the cloud, it's not only about functional benefits but also about further innovations that become possible because you have more resources to dedicate to what really matters for your business.

Now that we understand the advantages that the cloud brings to the table, the need for an integration platform weaving together the loose ends between various combinations of cloud and on-premise solutions becomes even more obvious. SAP Cloud Platform Integration is itself a cloud solution based on the SAP Cloud Platform. It benefits from all of the advantages just outlined and manages reliable message exchanges between all participants. Whether we're talking about SAP on-premise/

cloud applications, non-SAP on-premise/cloud solutions, or business-to-business integrations (B2B), SAP Cloud Platform Integration is the solution of choice to connect them. In other words: a cloud strategy without SAP Cloud Platform Integration would be impossible. The solution is therefore SAP's strategic integration platform on the road to becoming a successful cloud company.

SAP Cloud Platform Integration has another big advantage when compared to the competition: because SAP knows its own applications better than any other vendor, SAP Cloud Platform Integration is shipped with preconfigured integration content that just needs a few configuration steps to become productive for most businesses. This approach reduces time and money for integration projects to the absolute minimum.

But what does a typical integration scenario look like? To sharpen the picture of a universal integration platform, let's take a look at some concrete use cases.

1.2 Use Cases

Business processes, in many cases, require different applications and software systems to exchange data with each other. For example, clicking on the **Buy an article** button in a seller's web catalog typically triggers subsequent processes that involve complex data flows in a landscape of software systems, often distributed across the boundaries of different organizations. Such landscapes are usually heterogeneous in the sense that the systems that communicate with each other use different technical communication protocols and store data in different individual formats and data structures.

An integration expert who is in charge of implementing such complex distributed processes faces the challenge of cross-linking a large number of systems and data sources in the right way, and of making sure that the relevant data is exchanged during the operation of a business process correctly.

In the following sections, we'll first discuss how such integration challenges can be addressed by SAP Cloud Platform Integration. In this context, we'll introduce the concept of mediated communication. We'll then describe the various use cases that are covered by SAP Cloud Platform Integration in general. Over the course of this discussion, we'll also briefly touch on another integration solution provided by SAP, called SAP Process Orchestration.

1.2.1 Point-to-Point versus Mediated Communication

Let's first assume that our integration expert solves the integration challenge by implementing direct connections between each of the components that are to exchange data with each other. The resulting method these components use to exchange data with each other is referred to as *point-to-point communication*.

For example, if the components are different SAP systems, *point-to-point connections* can be implemented by remote function calls (RFCs). In this case, all components are tightly coupled with each other, as illustrated in [Figure 1.2](#), on the left. This setup has a number of disadvantages. For example, if one component, A, is upgraded, all connections where A is involved also have to be adapted. In the case of a large number of components, upgrade and maintenance tasks could easily spiral out of control, because the number of connections grows to the square of the number of components.

A more reliable and efficient approach to solving this integration challenge is to use a central integration platform—or an *integration bus*—that is interconnected between the involved systems. This setup is illustrated in [Figure 1.2](#), to the right. All integration-related tasks are managed by the integration bus. This approach is called *mediated communication*, and it ensures that the number and arrangement of connections remains manageable.

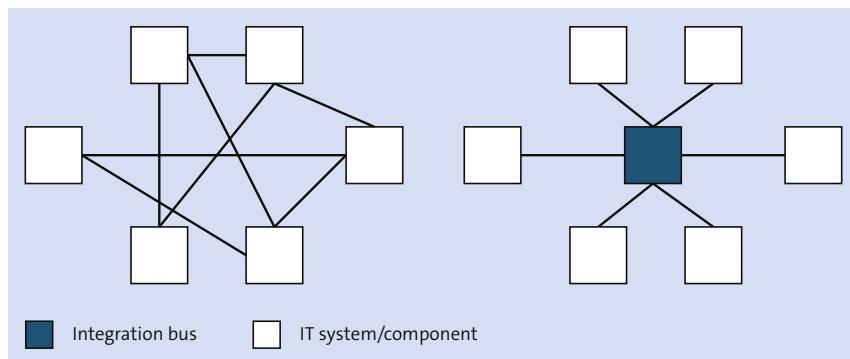


Figure 1.2 Point-to-Point Communication (Left) with a “Spaghetti-like” Arrangement of Connections, Compared to Mediated Communication (Right)

1.2.2 Message-Based Process Integration

Mediated communication is typically based on the exchange of messages. Let's examine what this means with the help of a few simple examples.

The integration bus makes sure, for example, that a message sent from system A to system B in a certain process step is transformed in a way that system B can interpret and further process the data contained in the message. If a message is to be forwarded from system A to more than one receiver, the integration platform manages proper routing of the message. Other process steps, in turn, might require that messages are split into smaller chunks, each of which might be forwarded to a different receiver, and so forth.

The integration bus approach makes sure that integration-related information and processes are centrally maintained and that tasks such as maintaining and updating the integration software are kept separate from integrated business applications.

An additional paradigm is to design the integration in a way that the applications to be integrated are loosely coupled with each other; that is, each individual application runs independently. This approach makes the overall business process less error-prone and reduces dependencies.

By addressing such integration challenges and approaches, SAP Cloud Platform Integration enables you to integrate processes that span different applications, organizations, or enterprises. This includes systems and applications of any kind, including non-SAP systems.

It's evident that different kinds of business processes require different means of exchanging messages between connected components. SAP Cloud Platform Integration supports a large number of these *Enterprise Integration Patterns*.

The term enterprise integration pattern—also referred to as *messaging pattern*—has been shaped and made popular by Gregor Hohpe and Bobby Woolf in their book *Enterprise Integration Patterns* (Addison-Wesley, 2004). The book defines and documents a number of integration patterns and how robust integration solutions can be designed. Each of the 65 patterns are illustrated in an intuitively understandable visual notation. The description is general enough to allow for an implementation with many different integration technologies.

A key capability of SAP Cloud Platform Integration—if not the most important one—is that it supports the implementation of enterprise integration patterns. As we'll explain in detail in [Chapter 2](#), SAP Cloud Platform Integration uses the integration framework Apache Camel for this.

Integration Pattern Examples

One example of an Enterprise Integration Pattern is the *content-based router*. Imagine that a sender is connected to multiple receiver systems, and the business process requires that a message from the sender is forwarded to a particular receiver system depending on the content of the message (e.g., a customer ID). The content-based router makes such forwarding possible.

Another example is the *splitter*, which defines that a single message is split into multiple partial messages that can be processed individually.

To support different integration patterns, SAP Cloud Platform Integration offers a wide range of integration capabilities and connectivity options, as described in [Section 1.3](#). However, let's first go into the different use cases that are supported by SAP Cloud Platform Integration.

1.2.3 Cloud-to-Cloud Integration

SAP Cloud Platform Integration is a cloud-based integration platform that provides on-demand process integration services (see [Figure 1.3](#)). Customers can use the platform's resources flexibly by paying a monthly fee. They don't need to install any integration middleware in their own landscape. All software processes that deal with message exchange run fully on SAP servers.

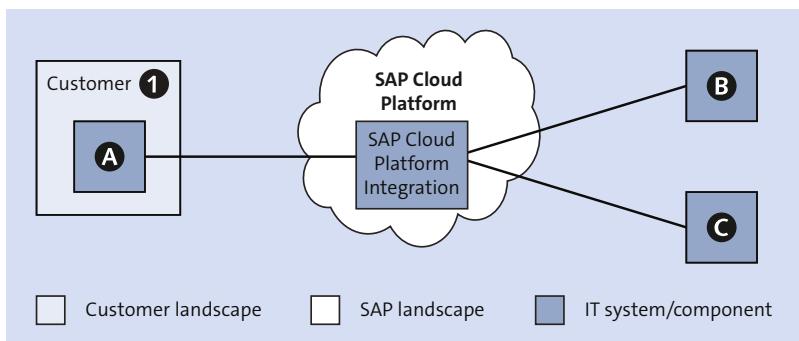


Figure 1.3 SAP Cloud Platform Integration as the Integration Platform Based on the SAP Cloud Platform

To run an integration scenario that requires message exchange between the three IT systems outlined in [Figure 1.3](#), connections between these systems and SAP Cloud

Platform Integration are required. All integration-related processes run in the SAP landscape (as part of SAP Cloud Platform).

Figure 1.3 doesn't specify which kind of component or systems can be connected with each other. As our example, let's assume that component **A** belongs to the system landscape of customer **1**. The other components to be connected aren't further specified.

Components or applications that are subject to an integration scenario can, in principle, be differentiated according to the following criteria:

- **On premise**

The component or application is installed and maintained in the landscape of an integration solution customer, on the premises, or locally, in the customer landscape.

- **Cloud**

The component or application runs in the cloud—for example, in SAP's cloud—and can be used by the customer on demand.

SAP Cloud Platform Integration, first and foremost, is suitable for the integration of cloud applications, that is, for cloud-to-cloud integration.

1.2.4 Cloud-to-On-Premise Integration

More and more cloud applications appear on the market and replace, in part, applications that until recently have only been available as on-premise solutions. However, large enterprises have invested much in their on-premise landscape and want to keep this investment. They only want to source out a part of their business into the cloud. These enterprises look for integration solutions that support the integration of their existing on-premise applications with new cloud applications.

SAP Cloud Platform Integration also supports this cloud-to-on-premise integration use case, whether you're using SAP or non-SAP systems.

Predefined Integration Content

SAP allows SAP Cloud Platform Integration customers to quickly implement a range of integration solutions out of the box. For this, customers can choose among a set of predefined integration packages in the Integration Content Catalog. The available integration packages cover the integration with a number of SAP solutions. We'll go into more detail on this in Chapter 3.

1.2.5 On-Premise-To-On-Premise Integration

SAP has successfully distributed SAP Process Orchestration for a number of years and continues to do the same today. SAP Process Orchestration is a combined package of SAP Process Integration and SAP Business Process Management (SAP BPM) capabilities that allow you to design business processes.

Note

Refer to *SAP Process Orchestration: The Comprehensive Guide* (SAP PRESS, 2017) for a comprehensive introduction to SAP Process Orchestration.

The SAP Process Integration part of SAP Process Orchestration is an on-premise integration middleware that, like SAP Cloud Platform Integration, can be used as an integration bus and that addresses the integration challenges mentioned in [Section 1.2.1](#) and [Section 1.2.2](#). In this context, we only refer to these integration bus capabilities and therefore don't consider the overall SAP Process Orchestration solution.

From a use case perspective, and having introduced SAP Cloud Platform Integration as the solution of choice for cloud-based integration, we want to emphasize that SAP Process Integration is the recommended solution for pure on-premise-to-on-premise integration.

Note that SAP Process Integration itself is an on-premise middleware that requires customers to install the integration software within their own landscape.

[Figure 1.4](#) outlines the differences between a cloud-based integration solution and an on-premise integration solution by providing a high-level comparison of the general technical landscapes for SAP Process Integration and SAP Cloud Platform Integration. In the example, two customers run integration scenarios that involves message exchanges between three IT systems (**A**, **B**, and **C**), where system **A** is part of the customer's landscape. Systems **B** and **C** aren't further specified. On the left side of [Figure 1.4](#), SAP Process Integration is installed in the landscape of customer **①**. This means that, at runtime, the message exchange among systems **A**, **B**, and **C** is performed by an integration bus hosted *within* the customer's landscape. On the right side of [Figure 1.4](#), customer **②** is using SAP Cloud Platform Integration as an integration bus to facilitate message exchange among IT systems **A**, **B**, and **C**. Here SAP Cloud Platform Integration is *outside* the customer's landscape (i.e., in the cloud).

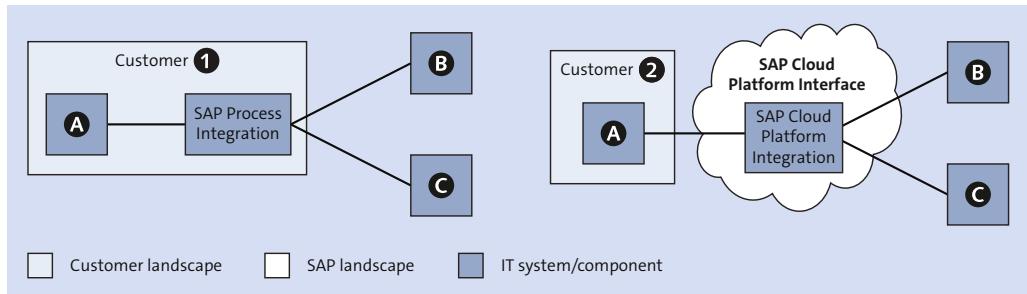


Figure 1.4 General Use Case and Component Setup for SAP Process Integration (Left) and SAP Cloud Platform Integration (Right)

Federated Process Integration

In more complex integration scenarios that span different enterprises, entire landscapes of various customers need to communicate with each other. Such cases typically require the installation of several SAP Process Integration instances at the site of each involved customer. Such federated scenarios fall outside the scope of this book.

Although SAP Cloud Platform Integration and SAP Process Integration are both integration platforms, note that SAP Cloud Platform Integration doesn't replace SAP Process Integration. SAP Cloud Platform Integration is a new solution that runs on SAP Cloud Platform, and it's designed as an integration platform-as-a-service (PaaS). We'll explain this in more detail in [Section 1.3.1](#).

1.2.6 Hybrid Usage of Cloud and On-Premise Integration Solutions

From a use case perspective, SAP Cloud Platform Integration is suitable for those business cases where customers like to outsource their integration-related processes—or parts of them—into the cloud.

Obviously, SAP Cloud Platform Integration is the solution of choice when you need to integrate processes that run within a landscape of cloud applications. As such, SAP Cloud Platform Integration is a complementary solution, as compared to SAP Process Integration. In many cases, the best strategy for customers might be to use SAP Cloud Platform Integration in combination with an already existing SAP Process Integration installation. For example, let's assume that you, the customer, have already

invested in an *on-premise integration solution* (e.g., SAP Process Integration) and would like to keep these investments, outsourcing only parts of your integration-related processes to the cloud. [Figure 1.5](#) shows schematically how the landscape of such a *hybrid* scenario could appear. To connect SAP Cloud Platform Integration with SAP Process Integration, you can use the SAP Cloud Platform Connectivity service (hereafter, SAP Cloud Platform Connectivity) ([Section 1.3.3](#)).

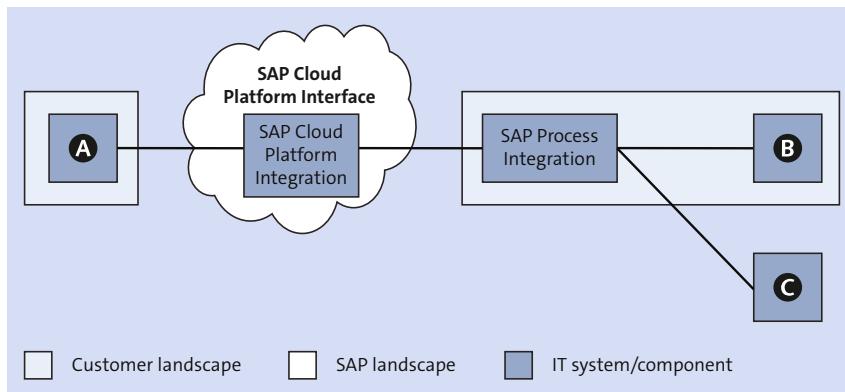


Figure 1.5 Hybrid Use Case Where Parts of the Landscape Are Integrated with SAP Cloud Platform Integration and Other Parts by SAP Process Integration

In this section, we've given an overview of the general use cases that can be addressed with SAP Cloud Platform Integration. To show how such use cases are implemented in practice, [Chapter 11](#) will describe a few examples of how SAP Cloud Platform Integration is used in real life.

1.3 Capabilities

Now that you understand how SAP Cloud Platform Integration can be used to benefit your business, let's delve deeper into the solution's capabilities. In this section, we provide a high-level overview of SAP Cloud Platform Integration's main features. We'll briefly touch on the aspect that SAP Cloud Platform Integration is an integration platform that is operated in the cloud and summarize the benefits of this setup. We'll then give an overview of the different ways messages can be processed by SAP Cloud Platform Integration and of the connectivity options supported by the platform. We'll also succinctly discuss integration content, security, and high availability,

and, finally, introduce the tools that come with SAP Cloud Platform Integration, as well as the different editions of the software that can be purchased.

In the subsequent chapters of the book, all these features will be discussed at length.

1.3.1 Integration Platform-as-a-Service

SAP Cloud Platform Integration is designed as an integration PaaS. What does that mean?

First, SAP Cloud Platform Integration helps you integrate multiple independent applications with each other in the context of a business process. This is the *integration platform* or *integration bus* aspect.

SAP Cloud Platform Integration allows you to integrate processes based on the exchange of messages. Similar to SAP Process Integration—the integration bus part of SAP Process Orchestration—SAP Cloud Platform Integration acts as an integration bus that is interconnected between the components that are connected in the context of a business process. All processes that manage data transfer and message routing run on the integration platform. SAP Cloud Platform Integration supports various methods of processing messages and offers many connectivity options that allow several software systems and applications to communicate with each other. These capabilities are described in more detail in [Section 1.3.2](#) and [Section 1.3.3](#).

Secondly, in contrast to SAP Process Integration, SAP Cloud Platform Integration provides integration services in the cloud. This is the *integration-as-a-service* (IaaS) aspect. The resources of the integration platform can be used on demand. Furthermore, you can flexibly adapt resource consumption according to changed business requirements. The latter capability is also referred to as *horizontal scaling*; that is, whenever you require more processing capacity, additional resources can be allocated quickly.

Customers don't need to care for the maintenance and upgrade of the integration software. SAP provides monthly updates of the software without any need for customers to schedule any downtime of their business processes that are based on SAP Cloud Platform Integration.

A key characteristic of the cloud-based integration platform is *multitenancy*: although different components and organizations (participants) connected to SAP Cloud Platform Integration share the same physical resources, these resources are

strictly isolated per participant. This means that data owned by a participant is strictly separated from data owned by other participants.

1.3.2 Message Processing Step Types (Integration Capabilities)

SAP Cloud Platform Integration supports various integration patterns, or means of integrating applications with each other. Each pattern requires a specific set of processing steps. At the time of this book's publication, SAP Cloud Platform Integration supports the following types of message processing steps:

- **Participants**

Contains objects that represent participants of an integration scenario, for example, a sender and receiver system.

- **Mapping**

Contains mapping steps that transforms the data structure or format used by the sender into a structure or format the receiver can consume.

- **Message Transformers**

Transforms the content of a message. Such transformation steps include content modifier, convertor, encoder, and many others.

- **Persistence**

Stores the message content in the database. Storage steps may be required when the message content is needed for later processing steps or if the message content should be analyzed in a later step.

- **Security elements**

Includes steps that enable you to digitally encrypt or sign (or both) the content of a message to ensure maximum protection of the exchanged data. It's also possible to decrypt signed messages. SAP Cloud Platform Integration supports a number of security standards, as will be described in detail in [Chapter 10](#).

- **Message routing**

Forwards a message to multiple receivers. Routing can also be defined to depend on the content of the message (content-based routing).

- **Events**

Contains different types of events, including end event, end message, error end event, and many others. Refer to [Chapter 2](#) for the complete list.

- **Process**

Enables you to define subprocesses and exception processes.

- **Call**

Enables you to perform calls to local or external services.

- **Message Validator**

At the time this book was published, this section contains the XML validator, which enables you to validate an XML message based on a specific XML schema.

We'll discuss the complete list of integration patterns and involved processing steps in [Chapter 2](#) and [Chapter 5](#).

1.3.3 Connectivity Options

The variety of integration scenarios that can be designed and operated with SAP Cloud Platform Integration depends on the kinds of systems that can be technically connected to it. These *connectivity options* are implemented by SAP Cloud Platform Integration adapters.

[Figure 1.6](#) illustrates various connectivity options.

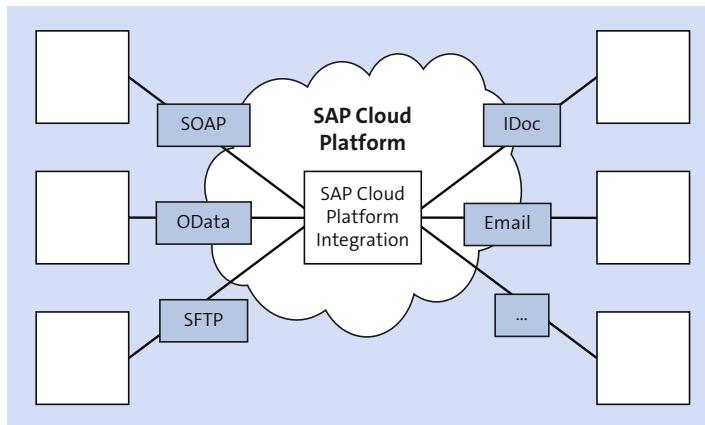


Figure 1.6 Connectivity Options That Allow You to Integrate Systems with Different Technical Characteristics with Each Other

At the time of this book's publication, the adapter types in [Table 1.1](#) are available for use.

Adapter Type	Description
SOAP adapter	<p>Allows you to connect SAP Cloud Platform Integration to a system that communicates based on the Simple Object Access Protocol (SOAP).</p> <p>There are currently two flavors available:</p> <ul style="list-style-type: none"> ■ SOAP (SAP RM) adapter: Connects SAP Cloud Platform Integration to systems based on SAP Reliable Messaging (SAP RM), which is a simplified communication protocol for one-way asynchronous communication. ■ SOAP 1.x adapter: Connects SAP Cloud Platform Integration to systems based on SOAP 1.1 or SOAP 1.2.
IDoc (SOAP) adapter	Allows systems to exchange an IDoc through SAP Cloud Platform Integration.
HTTP adapter	Allows you to connect SAP Cloud Platform Integration to receiver systems using HTTP.
Mail adapter	Allows you to connect SAP Cloud Platform Integration to mail servers and send out encrypted emails.
SFTP adapter	Allows you to connect SAP Cloud Platform Integration to a system through the Secure Shell File Transfer Protocol (SFTP).
SuccessFactors adapter	Allows you to connect SAP Cloud Platform Integration to an SAP SuccessFactors system through the SOAP, Representational State Transfer (REST), or Open Data Protocol (OData) message protocol.
Ariba adapter	Connects SAP Cloud Platform Integration to the Ariba Network.
OData adapter	Allows you to connect SAP Cloud Platform Integration to OData service providers.
Twitter adapter	Allows you to connect SAP Cloud Platform Integration to Twitter and to extract Twitter content.
Facebook adapter	Allows you to connect SAP Cloud Platform Integration to Facebook and to extract Facebook content.

Table 1.1 SAP Cloud Platform Integration Adapter Types

Adapter Type	Description
ODC	<p>Enables you to connect SAP Cloud Platform Integration to a SAP Gateway OData Channel. Note that it uses the HTTPS transport protocol.</p> <p>The following operations are currently supported: create (POST), read (GET), update (PUT), delete (DELETE), merge (MERGE), and query (GET).</p>
AS2	<p>Allows you to exchange business-specific data securely and reliably with a partner through the Applicability Statement 2 (AS2) protocol. Security is achieved using digital certificates and encryption.</p>
JMS	<p>Allows you to exchange asynchronous messaging using message queues with the Java Message Service (JMS). Note that a license for SAP Cloud Platform Integration, Enterprise Edition, is required to use this feature. Refer to Chapter 5 for more details.</p>
RFC	<p>Enables you to connect SAP Cloud Platform Integration to a SAP ABAP system using an RFC destination.</p>
LDAP	<p>Connects SAP Cloud Platform Integration to a Lightweight Directory Access Protocol (LDAP) directory service (through TCP/IP).</p>
XI	<p>Enables connectivity to handle communication using the SAP Exchange Infrastructure (XI) protocol.</p>

Table 1.1 SAP Cloud Platform Integration Adapter Types (Cont.)

It's important to mention that there are additional adapters for SAP Cloud Platform Integration provided by different SAP partners. An up-to-date list of these adapters can be found via the **Discover** section in the Web UI of SAP Cloud Platform Integration. Note that some adapters are only available for one side of the communication, either to connect to a sender system or to a receiver system. For the actual available capabilities, refer to the SAP Cloud Platform Integration product documentation (see [Appendix B](#) for resources).

Several adapter types (e.g., the HTTP adapter and the OData adapter) support SAP Cloud Platform Connectivity, which can be used as a link between SAP Cloud Platform components and on-premise landscapes. SAP Cloud Platform Connectivity

supports the simple configuration of on-premise systems that are to be connected with the SAP Cloud Platform.

You can also build your own adapters using the Adapter Development Kit (ADK), as described in detail in [Chapter 6](#).

A concrete integration scenario is specified by combining a set of processing steps and adapters from [Table 1.1](#) in this section. For more information, read on to [Chapter 2](#).

1.3.4 Prepackaged Integration Content

SAP provides predefined integration content that allows you to implement a number of integration scenarios out of the box.

Integration Content and Integration Packages

The term *integration content* summarizes integration flows, interfaces, mapping programs, and other objects that define how messages are exchanged through SAP Cloud Platform Integration in the context of an integration scenario. A collection of such objects that addresses the integration challenges of a specific business case is referred to as an *integration package*.

For a definition of what an integration flow is, see [Chapter 2](#).

In the public Integration Content Catalog, you can select integration packages that facilitate the integration of applications with SAP SuccessFactors applications, SAP Cloud for Customer, and the Ariba Network.

The Integration Content Catalog is available at <https://api.sap.com/shell/integration>.

For more information on the available integration content in the SAP Integrated Content Catalog, see [Chapter 3](#).

1.3.5 Security Features

SAP Cloud Platform Integration is designed in a way that ensures maximum protection of customer data during the operation of an integration scenario. Physical protection of relevant data is ensured by the fact that the SAP Cloud Platform Integration is hosted in SAP data centers located in different countries, which provide a maximum level of protection, for example, through the usage of surveillance cameras at the data center location.

Although different participants connected to SAP Cloud Platform Integration share the same physical resources, data that is processed and stored on the platform is strictly separated per participant (multitenancy).

Each customer can define dedicated permissions for different people working on the customer-related account and tenant.

The connections between SAP Cloud Platform Integration and other components can be secured using standard transport-level security options, such as HTTPS and Secure Shell (SSH). On top of that, SAP Cloud Platform Integration provides many options to protect the exchanged messages with digital signatures and encryption.

These are only a few key security features. More details on the topic can be found in [Chapter 10](#), which thoroughly describes the security features of SAP Cloud Platform Integration.

1.3.6 High Availability

SAP Cloud Platform Integration is available for customers to process messages at any time, and productive business processes that use SAP Cloud Platform Integration as their integration bus will continue operation even if one component of the integration platform fails.

SAP Cloud Platform Integration's architecture supports horizontal scaling in the sense that additional runtime components can be flexibly added when required (e.g., in the case of component failure). If a node fails, another node starts immediately.

Regular health checks are also performed to make sure that exceptions are detected quickly and that the responsible administrator at SAP can take immediate measures to resolve the issue.

In addition, the SAP Cloud Platform Integration platform is hosted redundantly in different SAP data centers at different locations, and SAP provides monthly software updates without any downtime of productive scenarios.

All of these measures ensure that SAP Cloud Platform Integration customers aren't left in the lurch should component failure ever occur.

1.3.7 Integration Design and Monitoring Tools

Customers using SAP Cloud Platform Integration have the option to design or enhance integration content by themselves and to monitor message exchange at runtime. The following tools are available:

- A browser-based web application called the *Web UI* is available for these tasks, and it's the central tool that you'll deal with throughout this book.
- Various Eclipse add-ons are available for adapter developments.
- The SAP Cloud Platform cockpit can be used by customers to perform user management tasks on their account.

1.4 Editions

SAP Cloud Platform Integration comes in different editions that are tailored to different customer requirements. You can find the different editions and their descriptions listed in [Table 1.2](#).

Edition	Description
SAP Cloud Platform Integration for SAP cloud applications	<p>Can be used as a dedicated integration solution in combination with at least one specific SAP cloud application, such as SAP SuccessFactors.</p> <p>When you have an integration use case with at least one connection to an SAP cloud application, this edition is the solution of choice.</p>
SAP Cloud Platform Integration, DI edition	Geared toward data integration. It therefore includes data integration capabilities but excludes other components such as SAP Process Integration.
SAP Cloud Platform Integration, PI edition	<p>Can be used as standalone, general-purpose integration platform with no restriction on the types of connected systems.</p> <p>This edition allows any-to-any system integration.</p> <p>A determined number of connections can be used, and a maximum bandwidth (in terms of transferred data volume per time) is available. Note that this edition doesn't include data integration. It's possible to extend this edition.</p>

Table 1.2 SAP Cloud Platform Integration Editions

Edition	Description
SAP Cloud Platform Integration, Enterprise Edition	<p>This is a fully featured version of SAP Cloud Platform Integration that includes the following features:</p> <ul style="list-style-type: none"> ■ Data integration ■ SAP Process Integration ■ Unlimited connections ■ API management ■ OData Provisioning ■ SAP Enterprise Messaging
Partner Edition	<p>The edition is specially tailored for SAP partners to develop their own custom integration scenarios. This includes, for instance, integration content or adapter development.</p> <p>The edition includes a determined number of connections and a maximum bandwidth. Additional connections and bandwidth can be purchased according to customer requirements. Note that this edition isn't intended for productive use.</p>

Table 1.2 SAP Cloud Platform Integration Editions (Cont.)

For more details on the number of connections, bandwidth, and included components in the editions mentioned in [Table 1.2](#), go to https://cloudplatform.sap.com/support/service-description.html#section_11.

For each edition, SAP remains fully in charge of providing, maintaining, and upgrading SAP Cloud Platform Integration on a regular basis. SAP sets up the account and tenant for the customer and operates the customer-specific integration runtime.

In general, however, the customer is responsible for integration content design and deployment on the tenant, as well as for monitoring the message exchange at runtime.

1.5 Summary

In this chapter, we introduced SAP Cloud Platform Integration as *the* integration solution that supports companies on their journey of digital transformation. We've shown that SAP Cloud Platform Integration's use cases cover both cloud-to-cloud and cloud-to-on-premise integration, and that SAP Cloud Platform Integration can also

be used together with SAP's on-premise integration solution SAP Process Orchestration. Finally, we provided you with an overview of the capabilities of the product and introduced the available editions of SAP Cloud Platform Integration.

In the next chapter, we'll set our focus even more on SAP Cloud Platform Integration, introduce you to its main concepts, and disclose the architecture and setup of components. Furthermore, we'll show how you can quickly set up and run your first integration scenario.

Chapter 2

Getting Started

This chapter provides an introduction to the main concepts and the architecture of SAP Cloud Platform Integration. In addition, a tutorial shows you how to set up and run a simple integration scenario.

In this chapter, we'll answer the following main questions: How does SAP Cloud Platform Integration work, what are the building blocks it's composed of, and how do you get started using it?

Section 2.1 allows you to glance behind the scenes of SAP Cloud Platform Integration with a detailed look at the solution's architecture. It also explains the main components of the integration platform and how they interact with each other. We then discuss the basic mechanisms that are in action when a cloud-based integration scenario is executed. Section 2.2 outlines the typical sequence and the main phases of an integration project and points out how the tools provided by SAP Cloud Platform Integration fit into the sequence. Finally, Section 2.3 describes, step-by-step, how to set up and run your first simple integration scenario.

By the end of this chapter, you'll have the foundational knowledge needed to start working with SAP Cloud Platform Integration.

2.1 Architecture Overview

In this section, we show you how SAP Cloud Platform Integration's architectural design makes it a cloud-based integration platform that can be shared by many participants, allows for flexible resource allocation for different participants, and ensures that resources belonging to different participants are strictly separated from each other.

We begin with the following questions:

- What are the runtime components in charge of message processing during the operation of an integration scenario?
- Which components are in action when messages are passed through SAP Cloud Platform Integration?

We'll then see how the integration platform is structured in more detail and, finally, provide a complete bird's eye view of the solution's architecture.

2.1.1 Virtual and Clustered Integration Platform

As we discussed in [Chapter 1](#), SAP Cloud Platform Integration is a cloud-based integration platform that allows you to integrate processes by enabling them to exchange messages with each other. It's built on the SAP Cloud Platform.

SAP Cloud Platform

SAP Cloud Platform is SAP's platform-as-a-service (PaaS) offering that provides a development environment in the cloud which enables customers to develop, deploy, and manage applications.

SAP Cloud Platform Integration is designed as a virtualized and clustered integration platform in the cloud. In this context, *virtualized* means that the fundamental entities of the integration platform are virtual machines, rather than physical ones, and *clustered* means that the platform is modularized and composed of smaller, more or less independent entities that, as a whole, make up a cluster.

To illustrate this, consider a simple landscape with only a few IT systems participating in an integration scenario and SAP Cloud Platform Integration interconnected between them, as shown in [Figure 2.1](#). Let's further assume that, in a dedicated process step within the integration scenario, data is sent from system **A** to system **C**. Another process step triggers system **B** to send data to system **D** and other systems not specified here. It's not important now if system **A** and **B** belong to the same organization or to different ones, or if the two process steps are part of the same integration scenario or different ones.

[Figure 2.1](#) illustrates the following aspects of SAP Cloud Platform Integration: first, all systems (or participants) connected to SAP Cloud Platform Integration share the same physical resources of SAP Cloud Platform. You can assume that the required

hardware that takes care of processing the messages is located on one, or a few, of SAP's data centers. Secondly, each participant can access and use only a part of the commonly shared resources. Furthermore, the resources allocated to individual participants are strictly separated from each other.

In [Figure 2.1](#), a platform resource assigned to participant **A** processes messages received by **A** and forwards them to participant **C**, whereas a resource assigned to another participant processes messages from that participant.

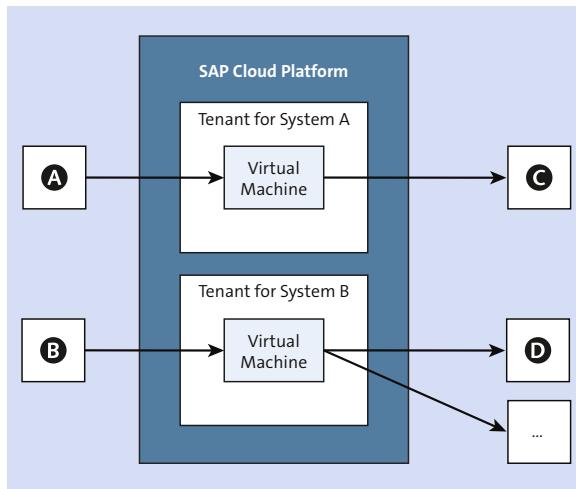


Figure 2.1 Partitioning of the SAP Cloud Platform Integration Runtime into Individual Virtual Environments

To explicate the aspect of resource isolation, let's first discuss the terms *account* and *tenant*.

Account

An account is the basic entry point to SAP Cloud Platform. It represents a hosted environment provided by SAP to a customer and defines a set of authorizations and platform resources allocated to the customer. There are two types of accounts on SAP Cloud Platform: global accounts and subaccounts.

A global account represents the functional scope of the platform you're entitled to use based on your license. It can contain one or more subaccounts.

A subaccount is associated with a certain physical location where applications, data, or services are hosted.

Tenant

A tenant is a logical entity that represents the physical resources of SAP Cloud Platform allocated to a specific participant and within an application context. SAP Cloud Platform Integration is multitenant-capable, meaning that although all customers using SAP Cloud Platform Integration share the same physical resources, their tenants can't interfere with each other. They are isolated from each other with regard to memory, data storage, and CPU. Tenant isolation means that physical resources allocated by tenant **A** can't be accessed by tenant **B**.

Although different tenants of SAP Cloud Platform Integration might physically share one common database, different database schemas per tenant are used. This ensures that data is strictly separated per tenant.

Participants or customers subscribing to SAP Cloud Platform Integration receive exactly one tenant per subaccount. The tenant is the foundational entity of SAP Cloud Platform Integration and is important topic throughout this book.

For each tenant, a separate virtual integration runtime operates. Messages sent from one participant to SAP Cloud Platform Integration are processed on a virtual machine—to be precise, on a Java virtual machine (JVM)—that is assigned to the tenant.

Basic Constituents of the Virtual Integration Platform

For each tenant, one or more virtual machines are operated. The term *virtual machine* (VM) is used in the sense that a VM is a software implementation that executes a program in the same way as a physical computer.

To be more precise, the SAP Cloud Platform Integration runtime environment is based on *Java virtual machine* (JVM), specified by the Java Platform, Standard Edition (Java SE). A JVM is a virtual environment used to execute Java applications.

The actual processing of messages at runtime is accomplished on a Java instance that is referred to as JVM instance.

An expert—such as an integration developer—can specify the way messages are to be processed on a tenant in an intuitive manner using a graphical editor. The corresponding models are referred to as *integration flows*. We'll explain this term and the related concepts in more detail in [Section 2.1.4](#). Here, we simply want to point out that an integration flow is deployed on a JVM of the SAP Cloud Platform Integration platform, enabling the JVM to process messages exactly as specified by the

integration flow model. As soon as a JVM instance (the runtime process) has been started, and an integration flow has successfully been deployed on it, the JVM instance is ready to process incoming messages in the intended way.

JVM instances constitute the basic *parts* into which the SAP Cloud Platform Integration runtime is split. As you'll see in [Section 2.1.2](#), the platform is implemented as a cluster of such virtual processes, which are referred to as *nodes* of the cluster.

Note that each node doesn't live only during the time a message is being processed by it. It's ready to process incoming messages from the moment it has been started by an administrator of SAP Cloud Platform Integration, and it lives until it's stopped either by failure or manually by the administrator.

Note that the resources provided by one VM aren't restricted to one integration flow. An integration developer can deploy multiple integration flows on the same JVM.

To recap, the SAP Cloud Platform Integration runtime is designed as a clustered virtual platform. Messages received from different connected participants are processed on separate JVM instances on the platform. The organizing entity is the tenant. All virtual machines belonging to different tenants are isolated from each other with regard to memory, data storage, and CPU.

[Figure 2.2](#) presents all entities that shape the SAP Cloud Platform Integration platform and shows how they relate to each other.

On the left side of [Figure 2.2](#), you can see that each customer subscribes to one or more subaccounts of SAP Cloud Platform. For customers who subscribed to SAP Cloud Platform Integration, exactly one tenant is assigned to each subaccount.

As we'll explain in [Section 2.1.2](#), [Figure 2.4](#), the virtual runtime environment of a tenant in general is composed of different nodes, and there are two different node types, each responsible for a different kind of task:

- Tenant management node
- Runtime node

For each tenant, one or more tenant management nodes, as well as one or more runtime nodes, can be started. Each node (independent of whether it's a tenant management node or a runtime node) is an instance of a virtual process running on SAP Cloud Platform (a SAP Cloud Platform process). Technically, each such process is a virtual machine. As the SAP Cloud Platform Integration runtime environment is based on a JVM, each VM running on the SAP Cloud Platform is realized as a JVM and can execute applications written in Java. We'll explain this in more detail in [Section 2.1.4](#).

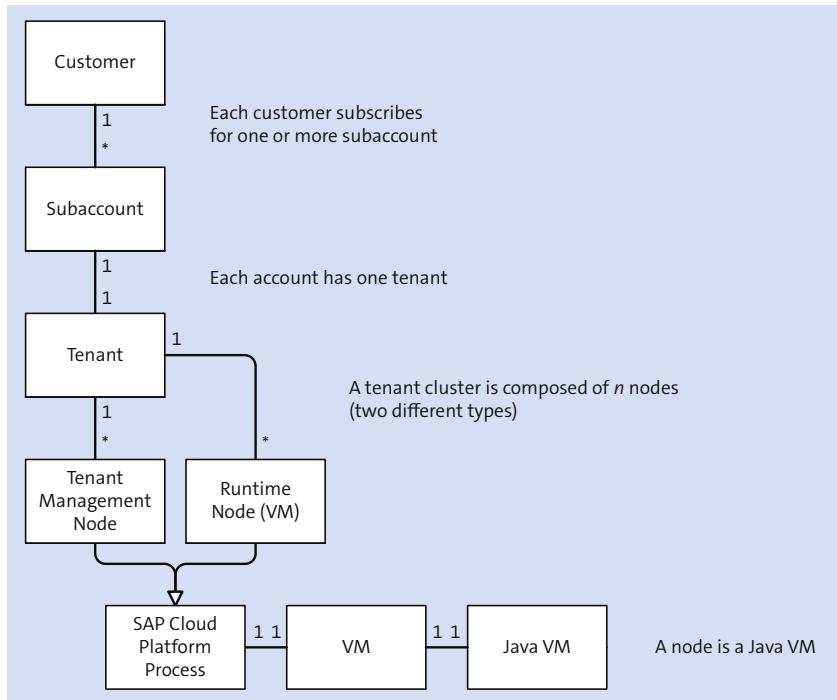


Figure 2.2 Entity Diagram of the Virtual SAP Cloud Platform Integration Runtime

HTTPS is often used as the transport protocol for remote connections, which implies Transport Layer Security (TLS) mechanisms. We'll go into more detail about the security aspects of SAP Cloud Platform Integration in [Chapter 10](#). At this stage, we only want to point out that the load balancer in the case of [Figure 2.3](#) terminates the inbound TLS connection and establishes a new connection to call the tenant.

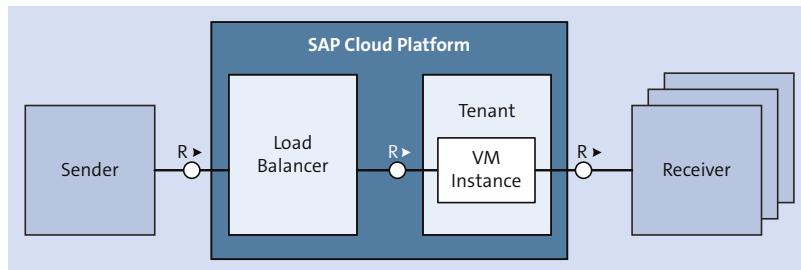


Figure 2.3 Incoming Messages Processed by the Tenant and Forwarded to One or More Receivers

Figure 2.3 shows how a tenant is related to the sender and receiver systems exchanging messages through that tenant (when the sender connects to SAP Cloud Platform Integration through HTTPS).

The processing of a sender's message is performed on a VM instance (which is the runtime node) on the *tenant*. Note, however, that the sender isn't directly connected to the tenant. A *load balancer* is interconnected that accepts all inbound requests and dispatches them to the right tenants.

The actual processing of messages is performed on the runtime node of the tenant. If routing steps are specified, the message is forwarded to multiple receiver systems, as illustrated in Figure 2.3.

You may have noticed the **R>** notation in Figure 2.3. This notation will also occur in many other figures throughout this book. **R>** indicates the direction of a request. In Figure 2.3, a sender system sends a message to SAP Cloud Platform Integration, where it's processed and forwarded to a receiver system. In other words, the direction of the message flow is from the sender to the receiver. In this same example (and in many other cases), the direction of the message flow is identical to the direction of the request. For the sake of simplicity, we assume that the sender initiates the message flow by sending a request message to SAP Cloud Platform Integration. Note, however, that there are also other cases where the directions of the request and of the message flow are opposite (e.g., in cases where a sender sends a request to a server to read, or *pull*, data from it).

2.1.2 Detailed Structure of a Cluster

In the previous section, you learned that each SAP Cloud Platform Integration tenant comprises a cluster of VMs (nodes). A cluster operated for a tenant is also referred to as *tenant cluster*. In this section, we'll shed more light on how a tenant cluster is structured.

A tenant cluster is composed of nodes of different types. Figure 2.4 shows the structure of a tenant cluster. It depicts how the different node types relate to each other. Furthermore, it points out the main data storage areas and user access points.

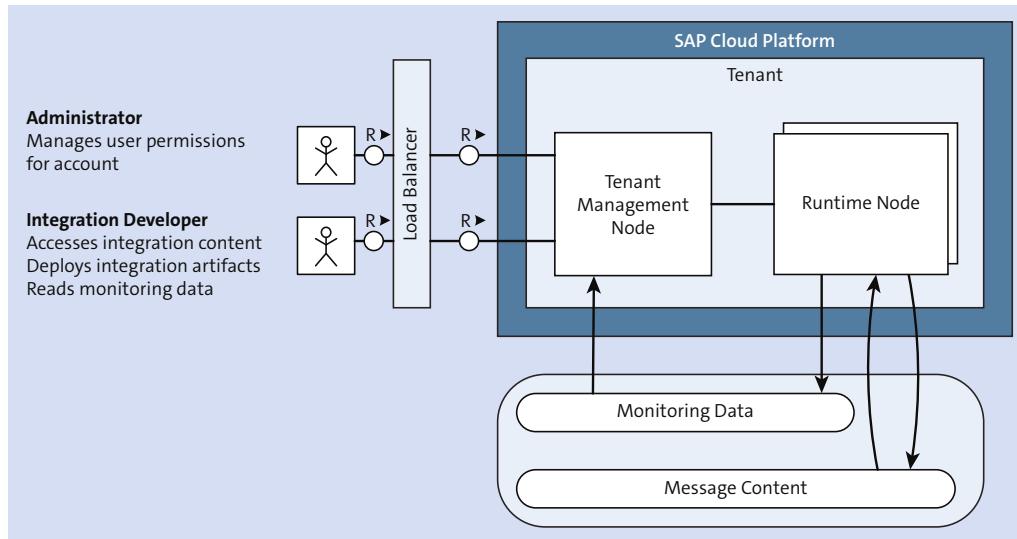


Figure 2.4 Structure of a Tenant Cluster, Indicating the Data Storage Areas and User Access Points

Table 2.1 summarizes the node types that constitute a tenant cluster.

Node Type	Tasks
Tenant management node	<p>Acts as agent between the human user and the runtime components of SAP Cloud Platform Integration.</p> <p>As illustrated by the direction of the request in Figure 2.4 (R>), the tenant management node manages the runtime nodes; in other words, it starts or stops runtime nodes.</p> <p>This node type interacts with human users (e.g., integration developers) who perform tasks such as deploying integration artifacts (Section 2.1.4 and Section 2.1.5).</p> <p>At the request of an integration developer, the tenant management node also reads message content and monitoring data from the database and makes it available for the monitoring application accessed by the user.</p>
Runtime node	<p>Processes messages that are exchanged through the tenant. Integration flows run on this node type.</p> <p>This node type interacts with the external systems and the load balancer.</p>

Table 2.1 Different Node Types Responsible for Different Kinds of Tasks

A tenant cluster is typically composed of one tenant management node with multiple runtime nodes associated to it.

High Availability

For simplicity, only one tenant management node is depicted in [Figure 2.4](#). However, SAP Cloud Platform Integration allows you to set up multiple tenant management nodes per tenant. Operating multiple nodes of the same type allows you to configure failover scenarios where, in case of a node failure, the redundant node can take over the tasks of the original one. This way, high availability is supported by the architecture of SAP Cloud Platform Integration.

At the time of this book's publication, SAP Cloud Platform Integration doesn't yet support elasticity in the sense that resources are automatically adapted to the needs of consumers of the integration platform. However, additional nodes can be easily and flexibly started in a tenant cluster. The option to flexibly add additional virtual resources is also referred to as *horizontal scaling*.

This architecture ensures that SAP Cloud Platform Integration is highly available; integration scenarios can be operated without the risk of any downtime.

Persistence

During the processing of a message (by a runtime node), there can be various steps where data is stored in a database (as indicated in [Figure 2.4](#)).

[Table 2.2](#) summarizes which kind of data can be stored at runtime.

Type of Stored Data	Description
Monitoring data	<p>Monitoring data records what happens to a message during the message processing.</p> <p>In particular, all involved processing steps are stored in a structure called a <i>message processing log</i> (MPL). This feature enables administrators to monitor message flow during the operation of an integration scenario. You can find more information on this in Chapter 8.</p> <p>As indicated in Figure 2.4, monitoring data is written to the database by the runtime node and read by the tenant management node, where it can be requested by the user.</p>

Table 2.2 Different Kinds of Data Stored at Runtime

Type of Stored Data	Description
Message content	<p>Message content contains business data (in the payload of the message) and the message header.</p> <p>Where in the processing sequence such data is to be stored can be configured per scenario.</p> <p>Message content can be stored long-term for purposes such as error analysis, audit logging, and archiving. In such a case, the storage duration is, by default, 90 days. This data is written by the runtime node to the database, but the runtime node can't read it from the database.</p> <p>Message content can also be stored temporarily to make it available for later processing steps in the same sequence. Temporarily stored message content, consequently, can be read by the runtime node during message processing.</p> <p>In all cases, the data can be stored encrypted.</p> <p>JMS Queues</p> <p>A specific, <i>temporary</i> persistence option is storing messages in Java Message Service (JMS) queues. You can use this option to asynchronously decouple inbound from outbound processing in scenarios where you like to enable the <i>integration platform</i> (rather than the sender system) to retry message processing in case an error occurs. When an integration flow has been configured to use JMS resources, it's also the runtime node that temporarily stores message content in JMS queues. This option is available with the SAP Cloud Platform Integration, Enterprise Edition.</p> <p>As indicated in Figure 2.4, message content is stored and (if temporarily persisted) accessed in the database and in JMS queues by the runtime node during message processing. For monitoring purposes, it's read by the tenant management node.</p>

Table 2.2 Different Kinds of Data Stored at Runtime (Cont.)

Software Update

The software running on the various subsystems of SAP Cloud Platform Integration is updated on a monthly basis by SAP in a process that doesn't require any downtime of productive scenarios running on SAP Cloud Platform Integration.

2.1.3 Secure Communication

A fundamental requirement for a cloud-based integration platform is that the data exchanged through it is protected from misuse. To that end, the remote systems and nodes of an integration platform should only communicate with each other through secure channels.

There are various options to set up secure connections between SAP Cloud Platform Integration and the involved remote systems. A common security measure is to use transport protocol HTTPS and to enforce mutual authentication of the communicating components based on digital certificates.

On top of transport-level security (e.g., HTTPS), SAP Cloud Platform Integration allows you to set up scenarios in a way that the exchanged messages are digitally encrypted and signed for increased security. All of these security options will be explained in detail in [Chapter 10](#).

At this point, we want to mention that each security option requires the implementation of digital keys of a certain kind, as well as the setup of storage locations for digital keys. To activate a certain security option for a tenant (and the integration scenarios deployed on it), these keys or key storages have to be deployed on the tenant as *integration artifacts*. In [Chapter 10](#), you'll learn about the complete set of security-related integration artifacts.

Integration flows, which have been introduced in [Section 2.1.1](#) (and will be described in more detail in [Section 2.1.4](#)), constitute another kind of integration artifact that can be deployed on a tenant.

2.1.4 Implementation of Message Flows

In this section, we'll discuss how a tenant cluster is enabled to process messages as defined in an integration flow.

We'll also briefly touch on additional, *specific* approaches that are supported by SAP Cloud Platform Integration, namely, first, specifying integration content for business-to-business (B2B) scenarios using the Integration Content Advisor (ICA), and, secondly, designing OData service artifacts. These options are described in detail in dedicated sections (in [Chapter 7](#) and [Chapter 4](#), respectively).

An integration flow, however, is the basic design artifact to define message flows, and it's of fundamental importance that you understand the concept of an integration flow before going into the details of more sophisticated integration design tasks.

Therefore, we focus on the basics of integration flow design in this beginner's chapter.

Integration Flows as Models to Specify Message Flows

An integration developer can intuitively define the way a message should be processed with a graphical modeling environment. The corresponding models are *integration flows*. They use a notation related to Business Process Model and Notation (BPMN).

Figure 2.5 shows a schematic sketch of an integration flow model (for the case when message routing is configured).

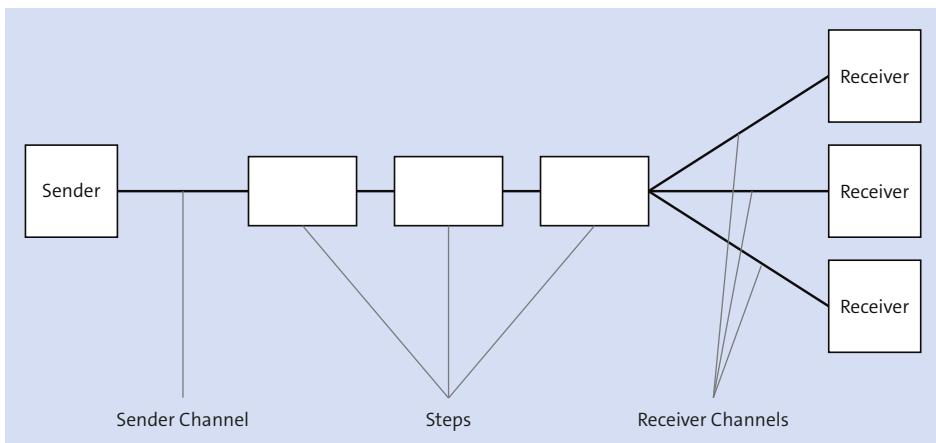


Figure 2.5 Schematic Illustration of the Integration Flow Elements

First of all, in an integration flow, you specify the sender and receiver systems of a message flow. The details of the technical connection between a sender/receiver and the tenant are configured within a channel. A channel allows you to specify an *adapter*, according to the technical protocol of the connected sender or receiver.

The way a message is processed by the integration bus and how it's transformed, routed, or enriched with additional data (to mention a few examples) are all specified within different *integration flow steps*.

Integration Flow

An integration flow specifies how a message is to be processed on a tenant. It's a BPMN-like model that can be intuitively specified by an integration developer as a

graphical model. To make an integration flow available for the runtime node (which is in charge of processing the message at runtime), it must be deployed on the corresponding tenant.

OData Services

A specific modeling option is provided by *OData service* artifacts. You can use such an artifact to expose various data sources as OData endpoints. It provides a simple and intuitive way to convert different kinds of protocols to the OData protocol. In technical terms, when an OData service is created, the system generates an integration flow out of it that contains an OData sender adapter by default. For more information, check out the information box in [Section 2.3.4](#) or [Chapter 4, Section 4.5](#).

Apache Camel as an Integration Framework

The information specified by an integration developer in an integration flow is made accessible to the runtime node, which is in charge of actually processing a message, via Apache Camel (<http://camel.apache.org>). Apache Camel is a Java-based open-source integration framework that allows developers to easily specify how a message is to be processed. [Chapter 4](#) provides a detailed introduction of Apache Camel's message model. In this section, we focus on those aspects that are required to get a first understanding of the concept behind an integration flow.

An elementary sequence of message processing steps is implemented as a *Camel route*. A Camel route specifies a message flow from one source endpoint to one or more target endpoints. The message flow can be subject to several processing steps such as data transformations and content-based routing.

As we mentioned before, as an integration expert in charge of designing integration scenarios, you don't need any Java programming skills or knowledge about Apache Camel to get the full functionality out of integration flows. You simply design the integration flow with a graphical tool and then deploy it on the tenant with a single click. With this second activity, you trigger a process that generates the corresponding Apache Camel objects out of the integration flow and makes them available for the runtime node associated with the tenant.

How the Integration Framework Relates to the Virtual Runtime

In [Section 2.1.2](#), you learned how the virtual runtime of SAP Cloud Platform Integration is structured, and what its properties are. In this section, we introduced the Apache Camel integration framework, and the associated design-time models, known as integration flows. Now let's discuss how the entities of the integration framework relate to those of the virtual runtime.

For this purpose, we've enhanced [Figure 2.2](#) by adding the entities that define how message processing is implemented on the runtime components. [Figure 2.6](#) shows the result.

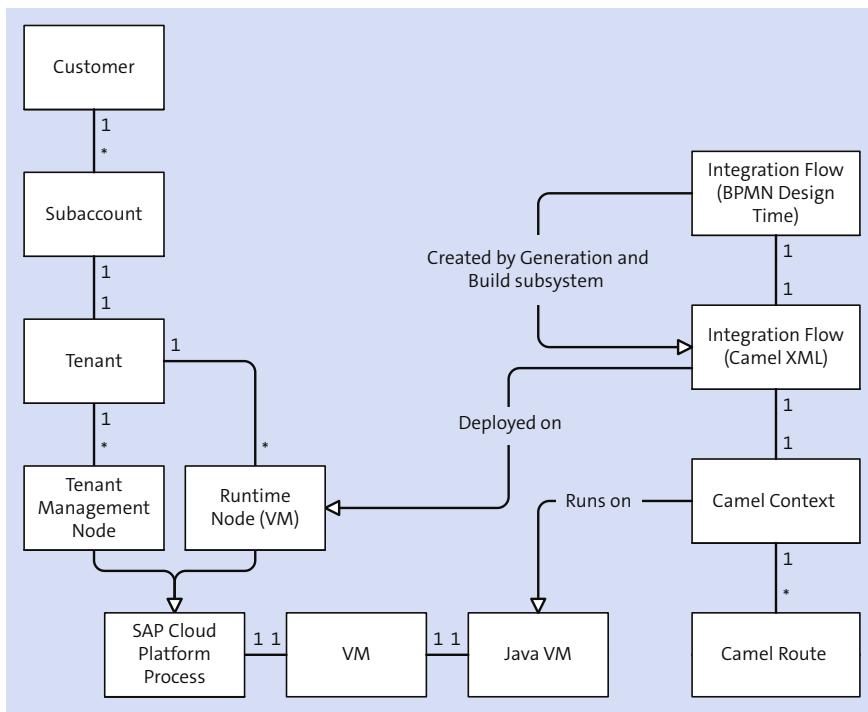


Figure 2.6 Runtime Resources and Artifacts That Define a Message Flow

The entities on the left side have already been explained in [Section 2.1.1](#) (also refer to [Figure 2.2](#)). These are the entities that make up the virtual runtime.

The right side of [Figure 2.6](#) depicts each entity that is relevant to specify message flows. The integration flow is a BPMN-like model created or edited by an integration developer with the support of a graphical tool. With an integration flow model, you

can fully specify how a message should be processed on a tenant. When the integration developer has finished modeling and deploys the integration flow on a tenant, the model is transformed into an XML structure, which is compatible with the Apache Camel specification. This XML structure is deployed on all runtime nodes assigned to the tenant. Looking into how message flows are implemented in Java, a Camel context defines the container of Camel components for a specific integration flow. Depending on the complexity of the integration flow, for each Camel context, one or more Camel routes are specified. A Camel route defines (on the level of the Java programming model) a message flow from one source endpoint to one or more target endpoints. In many cases, one integration flow finally leads to exactly one Camel route. However, complex integration flows can result in multiple Camel routes.

When you've finished the integration flow design and your integration scenario is up and running, you can monitor the runtime components with the monitoring application. [Chapter 8](#) covers these aspects in detail. Here, we would like to state that you can also monitor the status of all integration flows that have been deployed on your tenant. We refer to an integration flow as one kind of *integration artifact*. We introduced this term in [Section 2.1.3](#). With the monitoring tool, you can quickly check if the required integration flows are available for your integration runtime components.

Microservices

Before we wrap up this detailed walkthrough of the architecture of SAP Cloud Platform Integration, let's point out that the introduced cluster architecture implements the concept of microservices to a large extent.

A *microservice*, within a complex architecture, is considered a small and highly decoupled part of a software or service, which covers a specific task or a specified set of tasks. In other words, microservices allow you to modularize complex applications and help avoid monolithic systems by decomposing an architecture into smaller, independent services. They are small, easy to replace, organized around capabilities, and complete.

The presented cluster design implements this pattern: it's designed in a way that different node types are responsible for different tasks. The tasks related to message processing are accomplished by runtime nodes, whereas tasks related to the operation of a cluster and monitoring the message processing are accomplished by tenant management nodes. The nodes of a cluster are independent, in the respect that the availability of one node doesn't have any impact on that of others. Therefore, for

example, tasks like monitoring (on a tenant management node) are still possible when an associated runtime node fails.

The current design of SAP Cloud Platform Integration, however, shows one important difference from the microservices paradigm: runtime nodes and tenant management nodes can't (yet) be deployed independently.

2.1.5 Architecture Summary

So far in this section, you've learned that the SAP Cloud Platform Integration runtime is designed as a clustered virtual platform. Messages received from different connected participants are processed on separate JVM instances on the platform. The organizing entity is the tenant. All VMs belonging to different tenants are isolated from each other with regard to memory, data storage, and CPU. You also learned how message flows can be designed by a user and made available to the platform.

Now, let's pause for a moment and provide an overview of SAP Cloud Platform Integration's architecture to show how all the components described in the previous sections work together. [Figure 2.7](#) shows a high-level view of the overall architecture.

Let's start with the user access points in the left part of [Figure 2.7](#). First, user interaction takes place when a customer is provided with one or more SAP Cloud Platform subaccounts by SAP, each subaccount providing access to one SAP Cloud Platform Integration tenant.

The corresponding role involved in the first user interaction is the administrator. The administrator receives the initial information about the subaccount from SAP, accesses the subaccount through the SAP Cloud Platform cockpit, and defines authorizations for all additional users of the subaccount. Consider these additional users as representing the people who are—on the customer side—involved in tasks such as integration flow design or monitoring. In larger organizations, these tasks are typically performed by different persons who have separate permissions.

[Figure 2.8](#) shows the dialog box in the SAP Cloud Platform cockpit where the administrator can manage the authorizations of single users (arrow) or user groups (right side). For more information on the tools, [Section 2.2.1](#).

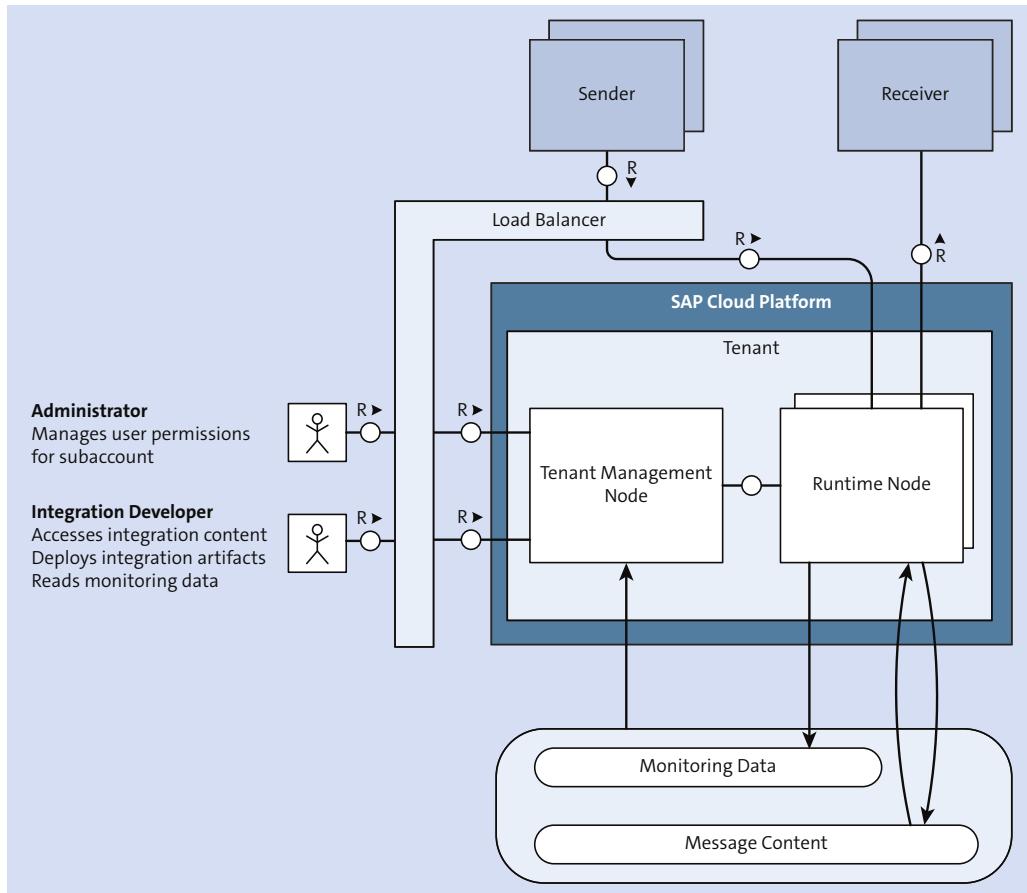


Figure 2.7 Bird's-Eye View of the SAP Cloud Platform Integration Architecture

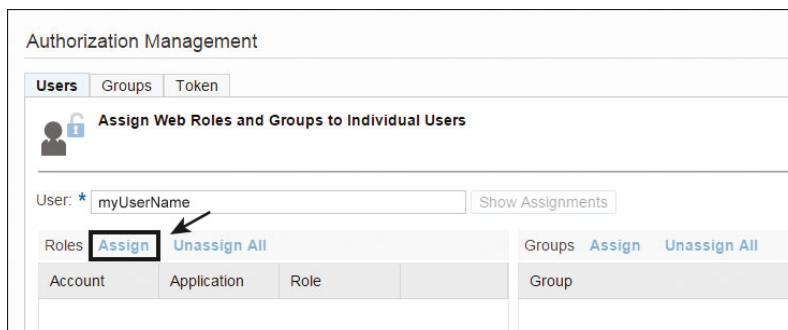


Figure 2.8 Authorization Management Page in the SAP Cloud Platform Cockpit

After the permissions have been defined, the additionally enabled users come into play and can start working with SAP Cloud Platform Integration. The role in focus is the integration developer.

The integration developer will work with a web-based frontend, called the Web UI, which we'll explain in more detail in [Section 2.2.1](#). The developer uses the Web UI for the following tasks:

- Designing integration flows or OData services with a graphical modeler
- Deploying integration artifacts (security content and integration flows) on the tenant
- Monitoring the processed messages, integration artifacts, stored data, and logs

The Web UI is connected to the tenant management node and is accessible by the user through a tenant-specific URL.

For the tenant assigned to the subaccount, the tenant cluster is started, which consists of one or more tenant management nodes, each managing one or more runtime nodes. The latter node type is actually responsible for the processing of the messages and, therefore, is connected to the external technical components (sender and receiver systems) that communicate with each other.

When an integration scenario has been set up and is in the operational phase, these technical systems can communicate with each other. As explained earlier, a tenant-specific runtime node is responsible for the processing of messages received by a sender system. As shown in [Figure 2.7](#), the sender system isn't directly connected to the tenant but is rather interconnected by a central load balancer component. All inbound traffic is managed by this component and dispatched to the appropriate tenants.

The runtime node processes the message according to the steps specified in the integration flow and forwards it to one or more receiver systems.

During the operation of the integration scenario, data is stored at various steps. This includes data that is required for monitoring purposes but also the message content itself for archiving purposes. The separation of data belonging to different customers is achieved by using different database schemas (respectively, JMS instances when JMS queues are used) per tenant (refer to [Table 2.2](#)).

Integration Content Advisor

If you have an SAP Cloud Platform Integration, Enterprise Edition, license, you additionally get access to the ICA. This application allows you to easily specify integration content for B2B scenarios in an intuitive way (see also [Section 2.2.1](#)). It comes with additional components and a database that stores the required B2B content.

In this chapter, we focus on the basic building blocks of SAP Cloud Platform Integration to introduce you to the key principles. Therefore, we didn't add the additional components related to the ICA in the architecture overview (refer to [Figure 2.7](#)).

ICA is described in detail in [Chapter 7](#).

Now that you're familiar with the architecture of SAP Cloud Platform Integration, we'll provide an overview of the tools related to SAP Cloud Platform Integration and of the processes relevant to integration developers.

2.2 Tools and Processes

This section provides an overview of the tools and main processes required to set up and run a SAP Cloud Platform Integration scenario.

2.2.1 Tools

Different tools are available for different kinds of user interactions in SAP Cloud Platform Integration.

SAP Cloud Platform Cockpit

This application provides access to SAP Cloud Platform and allows administrators to perform user management tasks for the account and tenant. The administrator can specify individual permissions for people who are supposed to work on the tenant cluster in different roles, such as integration developers (refer to [Figure 2.8](#)).

Customers with SAP Cloud Platform Integration, Enterprise Edition, also use SAP Cloud Platform cockpit to subscribe to ICA and to set up a JMS Message Broker for scenarios using JMS queues.

Public Integration Content Catalog

The Integration Content Catalog provides access to predefined integration packages provided by SAP that can be used out of the box. You can browse through the content at <https://api.sap.com/shell/integration> (see [Figure 2.9](#)).

The screenshot shows the SAP API Business Hub interface. At the top, there's a navigation bar with 'SAP API Business Hub' and 'Getting Started' on the left, and 'Classic Design' and 'Log On →' on the right. Below the header is a search bar with placeholder text 'Type here to search' and a magnifying glass icon. The main content area is titled 'SAP API Business Hub' and has a subtitle 'Discover and consume digital content packages with APIs, pre-packaged integrations, and sample apps from SAP and select partners'. On the left, there's a sidebar with a tree view of categories: SAP S/4HANA Cloud, SAP S/4HANA, SAP SuccessFactors, SAP Fieldglass, SAP Concur, SAP Ariba, SAP Cloud Platform APIs, Integration, and Digital Content. The main content area features several cards for different integration packages:

- SAP S/4HANA Cloud**: Described as 'The next generation digital core designed to help you run simple in a digital economy.' It shows 177 APIs and 74K+ users.
- Integration of SAP S/4HANA Cloud for Procurement with SAP...**: Described as 'Transform your SOAP APIs into your suppliers' intermediate documents (IDocs)'. It shows 5 Integration Content and 0.1K+ users.
- SuccessFactors Employee Central Integration with SAP S/4HANA Cloud**: Described as 'Replicate employee photo information from SAP SuccessFactors Employee Central to S/4HANA Cloud'. It shows 3 Integration Content and 1.7K+ users.

Below these cards, there are three smaller cards for SAP S/4HANA:

- SuccessFactors Employee Central to ERP**
- SAP S/4HANA with SAP Fieldglass**
- SAP S/4HANA Statutory Reporting for United**

Figure 2.9 The Integration Content Catalog

You may notice this content is published on the SAP API Business Hub (as you'll see in [Chapter 9](#)). The packages are stored in read-only mode. Customers who would like to use an integration package have to copy it into their own design workspace before they can start modifying it according to their requirements.

For more information on the Integration Content Catalog, see [Chapter 3](#).

Integration Content Advisor for SAP Cloud Platform Integration

As mentioned already in [Section 2.1.5](#), the ICA is available with SAP Cloud Platform Integration, Enterprise Edition. It facilitates the development of B2B scenarios because it allows you to easily design interfaces (referred to as *message implementation guidelines*) that best fit to your own and your partners' business requirements. Additionally, the tool allows you to create mappings between those interfaces. A key

feature of the ICA is that it assists the user in defining message implementation guidelines and mappings by using an intelligent, crowd-based machine learning approach.

When you've finished the specification of message implementation guidelines and mappings, runtime artifacts are generated out of them that can be used in integration flows.

For more information on ICA for SAP Cloud Platform Integration, read [Chapter 7](#).

Web UI for Integration Design and Monitoring

The Web UI is the most important tool used by integration developers. You can open the Web UI through a URL, although the specific URL depends on the customer tenant and is communicated to you in an email from SAP, along with the details of the tenant.

The Web UI comprises four tabs: **Discover**, **Design**, **Monitor**, and **Settings**. [Figure 2.10](#) shows how to access the tabs in the Web UI.

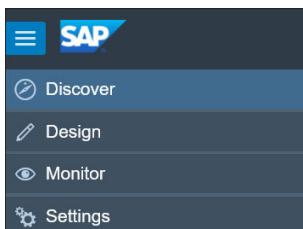


Figure 2.10 Tabs of the Web UI

[Table 2.3](#) provides an overview of the task areas covered by each tab. For more information, [Section 2.3.4](#), and the following *Basic Artifact Types* information box.

Tab	Allows You To...
Discover	Access the Integration Content Catalog. You can select already available content from this catalog and add it to your own workspace where you can further edit the content and adapt it to your own integration challenges. It also includes a list and details of third-party adapters provided by SAP partners. The Integration Content Catalog will be discussed further in Chapter 3 .

Table 2.3 Main Sections (Tabs) of the SAP Cloud Platform Integration Web UI

Tab	Allows You To...
Design	<p>Design the following kinds of artifacts:</p> <ul style="list-style-type: none"> ■ Integration Flow This is the key topic of this book. In Section 2.3, you'll learn how to create and run your first integration flow. The subsequent chapters cover the various aspects of integration content design in detail. ■ OData Service We'll introduce this design option in Chapter 4, Section 4.3. ■ Value Mapping We'll briefly touch on this topic in Chapter 4, Section 4.4. ■ Data Integration We won't cover this artifact type in this book because it isn't related to process integration. <p>For more information on these artifact types, see the <i>Basic Artifact Types</i> information box in Section 2.3.4.</p>
Monitor	<p>Monitor the message flow at runtime and check the status of deployed artifacts.</p> <p>Using this tab, you can also create and deploy security-related artifacts such as user credentials, keys, and certificates, and manage message stores and locks. Chapter 8 will cover this topic in detail.</p>
Settings	<p>Configure personal settings.</p> <p>In particular, this tab allows you to select a <i>product profile</i>. But what are product profiles good for? In this book, we only cover scenarios where the integration content designed with the Web UI is executed on your cloud-based SAP Cloud Platform Integration tenant cluster. However, you can also decide that your integration content is to be executed, for example, on an on-premise integration bus of SAP Process Integration (the integration bus that is available as part of SAP Process Orchestration). As the features that are supported by the SAP Process Integration runtime (of a specific release) slightly differ from those supported by the SAP Cloud Platform Integration runtime, the Web UI also provides a slightly different set of integration flow design features in case you select the corresponding product profile related to the target runtime SAP Process Integration. In other words, a product profile defines the target integration platform for the content designed with the Web UI.</p> <p>Chapter 6, Section 6.5.3, will touch on this topic.</p>

Table 2.3 Main Sections (Tabs) of the SAP Cloud Platform Integration Web UI (Cont.)

Tab	Allows You To...
Settings (Cont.)	Whether or not you have several profiles present on the Settings tab depends on the basic configuration of your tenant provided initially by the SAP Cloud Platform Integration operations team. Select the profile for which you intend to develop the integration flows.

Table 2.3 Main Sections (Tabs) of the SAP Cloud Platform Integration Web UI (Cont.)

Figure 2.11 shows an integration flow opened in the **Design** tab of the Web UI.

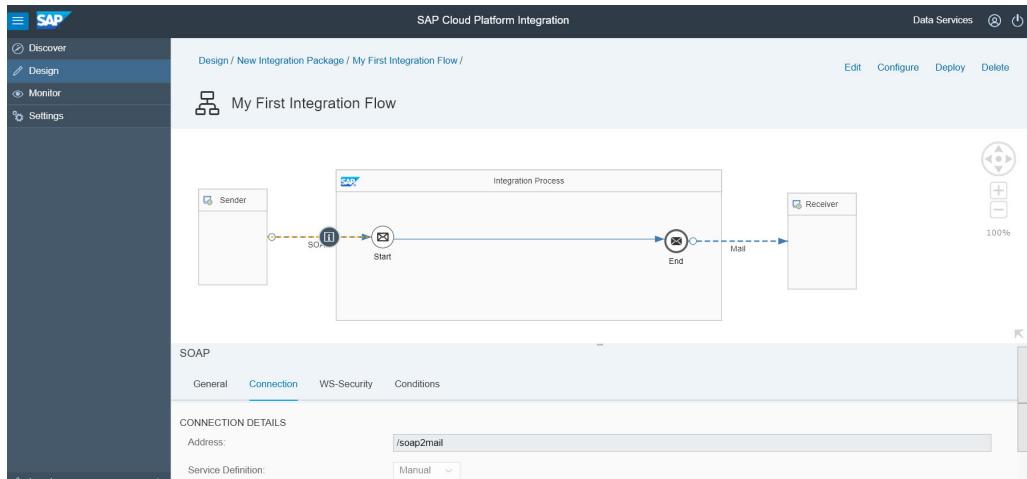


Figure 2.11 Modeling of an Integration Flow in the Design Tab of the Web UI

When designing an integration flow, you add elements (e.g., integration flow steps) to it and integrate them in a specific order to specify how messages are to be processed on the tenant. We provide an overview of the elements that can be added to an integration flow later in Table 2.4.

When designing OData services, the procedure is slightly different. We'll introduce this option in Chapter 4, Section 4.3.

Figure 2.12 shows the **Monitor** tab page from Figure 2.10.

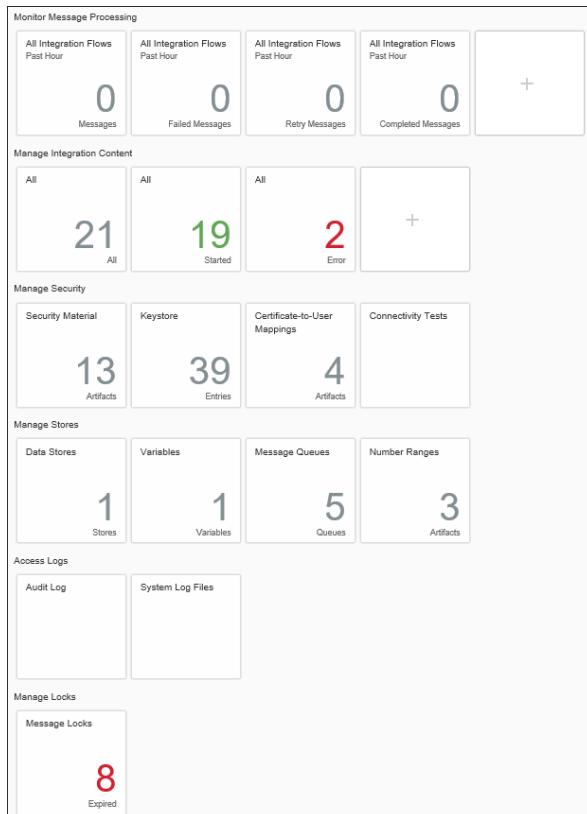


Figure 2.12 The Monitor Tab Page of the Web UI

The monitoring application is divided into the following main sections (not all are shown in the figure):

- **Monitor Message Processing**

This section allows you to monitor messages that are exchanged by your tenant based on the integration flows that are deployed.

- **Manage Integration Content**

This section allows you to monitor the status of the deployed integration content artifacts (integration flows).

- **Manage Security**

This section allows you to manage artifacts that are required when setting up secure connections between your tenant and remote systems. The **Security Material** tile allows you to manage specific security-related artifacts such as user

credentials, Pretty Good Privacy (PGP) keyrings, known hosts (required when using SFTP), or OAuth credentials. The **Keystore** tile allows you to manage keys and certificates that are required when using TLS. Using this tile, you can also manage the lifecycle of keys (renewing expired keys). The **Certificate-to-User Mappings** tile allows you to create and manage certificate-to-user mappings. You need such an artifact to configure inbound authentication based on a digital client certificate and to map this certificate to a user (for which you can then define permissions based on user roles). For more information on the concepts behind these artifacts types and how to use them, go to [Chapter 10](#). Finally, the **Connectivity Tests** tile allows you to test the outbound connection (for different protocols such as TLS, SSH, SMTP, IMAP, and POP3). We'll briefly show this tool in [Section 2.3.6](#) and explain it in detail in [Chapter 8, Section 8.3.4](#).

■ Manage Stores

This section allows you to handle storages on the tenant that are used to temporarily persist data during message processing. The **Data Stores** tile allows you to manage data stored temporarily by an integration flow (to make it available for later processing steps). You can view the content of the data store or delete entries. The **Variables** tile allows you to monitor variables used in integration flows.

When you've purchased SAP Cloud Platform Integration, Enterprise Edition, this section also shows a **Message Queues** tile that allows you to monitor queues that come into play when using the JMS or the AS2 adapter (for more information on the JMS adapter, go to [Chapter 5, Section 5.4](#)). For the different kinds of data stored during message processing, see [Table 2.2](#).

Finally, the **Number Ranges** tile allows you to get an overview of number ranges which are relevant when running B2B scenarios. In scenarios using Electronic Data Interchange (EDI) messages, a unique interchange number is added to each business document. The **Number Ranges** tile allows you to configure and monitor the interchange number (only available for SAP Cloud Platform Integration, Enterprise Edition).

■ Access Logs

This section allows you to monitor system changes, such as integration flow deployment events (**Audit Log** tile), and to get access to information related to errors that occurred during HTTP inbound processing (**System Log Files** tile). In case of inbound processing errors, the **System Log Files** tile might be the only information source available for administrators to analyze the cause of the error, as in such situations typically no MPL is created.

■ **Manage Locks**

This section allows you to manage lock entries that are created to prevent the same message from being processed more than once in parallel. In certain situations (e.g., when a runtime node shuts down unexpectedly), the system tries to reprocess a message, however, a lock entry is still in the database and blocks processing of subsequent messages. The **Manage Locks** section enables you to resolve such situations.

For more information on the **Monitor** application, see [Chapter 8](#).

In [Section 2.3](#), we'll introduce you to the Web UI and show how to create your first integration flow using this tool.

Eclipse Tools

The Web UI is the central tool that supports developers throughout the lifecycle of an integration project. However, over the course of this book, we'll also talk about an additional SAP Cloud Platform Integration-specific tool that requires an Eclipse installation. You'll need this tool to develop adapters (as explained in [Chapter 6, Section 6.7](#)).

2.2.2 Processes

In this section, we offer a brief overview of the main phases of an integration project before we walk you through your own first integration project in [Section 2.3](#).

The main phases of any SAP Cloud Platform Integration project are as follows:

1. **Browsing the Integration Content Catalog**

First-time users and decision makers can browse the Integration Content Catalog to find out which integration scenarios can be used out of the box with minimum implementation effort. We'll provide more details on this in [Chapter 3](#).

2. **Requesting a tenant and administering the account**

Customers who intend to use SAP Cloud Platform Integration need to contact SAP and request a tenant. After the customer has been provided with an account and a tenant, an administrator on the customer's side uses the SAP Cloud Platform cockpit to perform user management tasks on the account. In this phase, the administrator defines who else is allowed to work on the account and defines permissions for individual users.

3. Provisioning a JMS Message Broker

If you like to use JMS queues, you need to perform an additional step to set up a JMS Message Broker (only for users with SAP Cloud Platform Integration, Enterprise Edition). This is also done using the SAP Cloud Platform cockpit.

4. Setting up a secure connections between remote systems and the tenant

In this phase, secure communications with SAP Cloud Platform Integration are enabled for sender/receiver systems. This phase is divided into the following tasks (typically performed by different people):

- *Configuring the sender/receiver systems*

This task is usually performed by administrators of the sender/receiver system and comprises all steps to set up the technical connection between the sender/receiver system and SAP Cloud Platform Integration.

- *Configuring the tenant*

Users performing this task enable the tenant to securely communicate with the related sender and receiver systems.

In [Chapter 10](#), we'll explain how to set up a secure connection between SAP Cloud Platform Integration and remote systems.

5. Designing integration flows

In this phase, an integration developer designs integration flows to specify how messages should be processed on the tenant. When the design has been finished, the integration flows are deployed on the tenant.

To perform these tasks, the integration developer uses the **Discover** and **Design** tabs on the Web UI (refer to [Figure 2.10](#)).

Designing integration flows is the core topic of this book. After you've learned how to create, deploy, and run your first integration flow, you can find more details on integration flow design in [Chapter 4](#), [Chapter 5](#), and [Chapter 6](#). [Chapter 7](#) and [Chapter 10](#) also contain tutorials that show how to develop specific integration flows.

The process of designing an OData service is described in [Chapter 4, Section 4.3](#).

6. Operating and monitoring SAP Cloud Platform Integration

In this phase, you monitor the integration artifacts on the tenant cluster and the processed messages. To perform these tasks, the integration developer uses the **Monitor** tab on the Web UI.

For more details, consult [Chapter 8](#).

2.3 Running Your First Integration Scenario

In this section we'll show you how to set up and run a simple integration flow. By following these steps, you'll learn how to use the Web UI to create and edit an integration flow and how to deploy the integration flow on your tenant. Finally, when you've started the integration flow, you'll be able to monitor the resulting messages and the involved integration artifacts.

2.3.1 Demo Scenario and Landscape

For the sake of illustration, a simple integration flow has been chosen for our demo scenario. It will do nothing more than receive a Simple Object Access Protocol (SOAP) message (with a simple structure) from a SOAP client and forward it to an email receiver. In addition to having access to a SAP Cloud Platform Integration tenant, you only need a publicly accessible email account to set up and run this scenario.

The scenario intuitively shows one basic capability of SAP Cloud Platform Integration: exchanging messages between different systems. SAP Cloud Platform Integration supports connectivity with a large number of different systems, as shown in [Section 1.3.3](#). Such a system can be, for example, a complex SAP business system, a file system, or, as in our first example, an email server (to mention only a few examples).

This first integration flow might give you an idea of how SAP Cloud Platform Integration can be used as a message broker in heterogeneous system landscapes. Imagine that you, as an integration expert, are asked to build up an IT solution for your enterprise that accepts incoming messages (e.g., orders) and mails them automatically to the relevant departments within your organization. The first challenge to solve is to set up the message exchange between the sender system(s) and the mail server. This is where the various adapters (namely, the SOAP sender and the mail receiver adapter) come into play. For simplicity, in this first tutorial, we exclusively cover the connectivity aspect of SAP Cloud Platform Integration. The integration flow won't impose any further processing of the message on the SAP Cloud Platform Integration platform: it does nothing more with the received message than simply hand it over to an email server. [Chapter 4](#) and the subsequent chapters will then successively show you how a message can further be processed in various ways on its path between sender and receiver.

The following tutorial also provides you with a simple method of checking whether your tenant cluster works properly: You can simply check in your email account (and

with the **Monitor** application) to see if SAP Cloud Platform Integration delivered the message as expected.

Figure 2.13 visually depicts the target integration flow already in the notation that is used by the Web UI's integration flow design application.

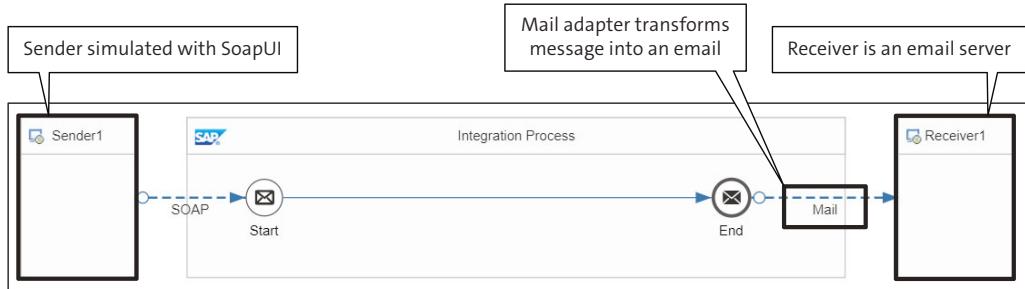


Figure 2.13 Your First Integration Flow

To simulate a SOAP Sender, you can use SoapUI (www.soapui.org). This software allows you to set up a SOAP client and send a SOAP message to an endpoint with a few clicks. The SOAP endpoint is a service on the runtime node where the integration flow is deployed.

Let's get started!

2.3.2 Prerequisites

Our first step is to make sure that the following prerequisites are met:

- Request a SAP Cloud Platform Integration tenant from SAP (www.sap.com). You'll receive an email from SAP with information on how to access the tenant and the Web UI.
- Install the SoapUI from www.soapui.org.
- Register an email account. For simplicity, we recommend using Gmail (www.google.com/gmail), and, in this tutorial, we show how to connect the tenant to a Gmail server.

2.3.3 Set Up the Landscape and the Technical Connections

To keep it simple, we assume that the SOAP sender calls the SAP Cloud Platform Integration system using basic authentication (with your user credentials). This way, you

don't need to generate and install client certificates and have them signed by a certification authority (this process will be explained in [Chapter 10](#)). Nevertheless, a keystore with specific content must be deployed on the tenant because even when inbound authentication is accomplished based on a username and password, a TLS-based authentication of the server (SAP Cloud Platform Integration in our case) is enforced (as explained in [Chapter 10](#)). SAP will provide the required keystore with the tenant, so you don't need to perform any additional steps related to this. Depending on the mail server that should be used as receiver, it might be required to import additional certificates into the keystore as explained in [Section 2.3.6](#).

Install and Configure SoapUI

Prepare the SOAP sender, and specify the format of the message that is to be sent to SAP Cloud Platform Integration. The data structure of the message is specified in the Web Services Description Language (WSDL). For your convenience, you can download the WSDL file from www.sap-press.com/4650. The name of the file is *SendOrder_Async.wsdl*.

After you've downloaded the WSDL file and installed SoapUI, perform the following steps:

1. Open SoapUI and create a new SOAP project by clicking on **File** and then selecting **New SOAP Project** in the menu.
2. Specify a **Project Name**, and click on the **Browse** button, located to the right of the **Initial WSDL** field ([Figure 2.14](#)).

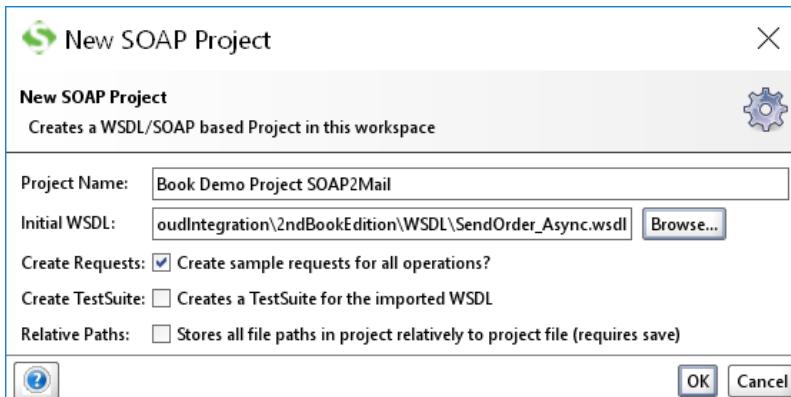


Figure 2.14 Required WSDL File Upload When Creating a Project

3. Navigate to the WSDL file on your computer, and click **Open**. The project appears in the left navigation bar.

When you open the request, you can find the structure of the message to be sent (see [Figure 2.15](#)).

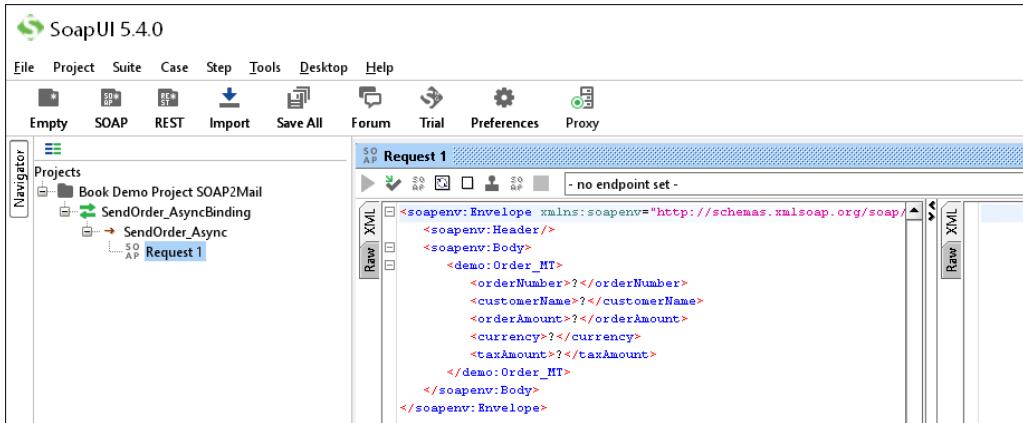


Figure 2.15 The Demo SOAP Project Consisting of One Request with a Simple Message Structure

Assign Role for Inbound Call to Runtime Node

The following scenario uses basic authentication as a simple option to authenticate the sender of the message (which is the SOAP client configured with SoapUI). The SAP Cloud Platform Integration runtime evaluates the authorization of the sending system (the SOAP client) based on permissions defined for the role associated with the sender. The permission to process messages on the SAP Cloud Platform Integration platform is defined in the role `ESBMessaging.send`. You'll encounter this role several times in the course of this book.

To authorize the user (associated with the inbound call) to process messages on the tenant, perform the following steps:

1. Access the SAP Cloud Platform cockpit through the URL provided to you by SAP with the email that contains the details of your tenant.
2. Select your subaccount.
3. Go to **Security • Authorizations**.

4. As **User**, enter the user that should be associated with the SOAP call.
5. In the next screen, for the **Application** field, select the entry from the dropdown list that ends with **iflmap** ([Figure 2.16](#)).

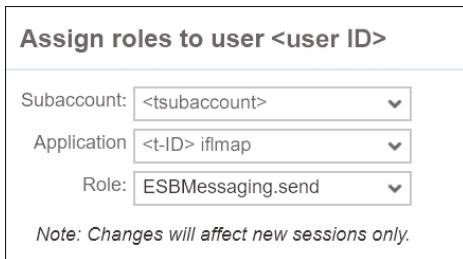


Figure 2.16 Assigning ESBMessaging.send to the Runtime Node in the SAP Cloud Platform Cockpit

The entry in the **Application** field that ends with **tmn** identifies the tenant management node. Roles with regard to permissions of human users in the integration team are assigned to the tenant management node (for more details on this, check out [Chapter 10, Section 10.3](#)).

6. For **Role**, the entry **ESBMessaging.send** should be preselected.
7. Choose **Save**.

Compare with [Figure 2.40](#) later in this chapter to check out how to specify the user credentials of the sender.

2.3.4 Develop the Integration Flow

Now it's time to develop your first integration flow. You'll begin by creating an integration flow, after which you'll design the integration flow using a number of available elements.

Create an Integration Flow

1. Open the Web UI using the **Web UI URL** provided in the email from SAP.
2. Choose the **Design** tab (refer to [Figure 2.10](#)).
3. Choose **Create** to create a new integration package (at the bottom of the left pane, see [Figure 2.17](#)).



Figure 2.17 Create Button for Integration Packages

4. In the next dialog box (shown in [Figure 2.18](#)), provide a name for the integration package and a short description.

A screenshot of the 'New Integration Package' dialog box. It has tabs at the top: HEADER (underlined), OVERVIEW, ARTIFACTS (0), DOCUMENTS (0), and TAGS. The 'ARTIFACTS (0)' tab is currently selected. Below the tabs are three input fields:

- *Name: New Integration Package
- *Technical Name: NewIntegrationPackage
- *Short Description: Integration Package for Book Examples

Figure 2.18 Specifying a Name for the New Integration Package

5. Select the **Artifacts** tab, click on **Add**, and select **Integration Flow** (see [Figure 2.19](#)). As a result, the **Add process integration flow artifact** dialog box shown in [Figure 2.20](#) opens.

This selection makes sure that, with the next steps, you can create and edit an integration flow. Before proceeding, let's briefly take a look at the other artifact types you can create in the following *Basic Artifact Types* information box.

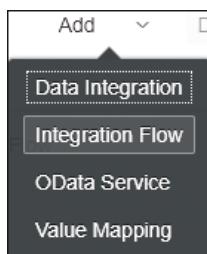


Figure 2.19 Artifact Types You Can Select When Creating Integration Content

Basic Artifact Types

Let's briefly talk about the three options that you'll encounter when you want to add an artifact to an integration package.

■ Data Integration

This option allows you to create artifacts to set up *data integration scenarios* that implement extract, transform, and load (ETL) processes. ETL processes commonly integrate data from multiple systems. This topic is beyond the scope of this book.

■ Integration Flow

This option allows you to create artifacts to implement *process integration scenarios* that integrate business processes across the boundaries of organizations or enterprises. An integration flow does exactly what is required for such scenarios: it specifies how messages are exchanged within such a process integration scenario. Integration flow design will be covered in detail in this book.

■ Value Mapping

This option allows you to define value mappings. Those of you who are already familiar with SAP Process Orchestration or SAP Process Integration know this function very well. For those not yet familiar with value mappings, the basic idea is as follows: Data to be exchanged isn't always represented in the same manner in the sender and receiver systems involved in an integration scenario. System X could be using values such as "male" and "female" to represent an employee's gender, whereas system Y could be using numeric values instead, such as "1" for male and "2" for female. Typically, when a message containing employee data needs to be exchanged between systems X and Y, these two ways of representing the same data need to be translated. The value of "male" from the source system will need to become "1" in the target system. This form of translation is called value mapping. For more information, see [Chapter 4, Section 4.4.4](#).

■ OData Service

This option allows you to develop an OData service from an existing data source (e.g., from a SOAP WSDL file). OData is an open standard that allows service providers to specify HTTP-based data access in a standardized manner. You can use an **OData Service** artifact, for example, when you like to expose data through OData to consume it in a frontend application. Note that there is also the option to create an **Integration Flow** artifact with an OData sender channel. In this case, however, you can only expose one OData entity and operation at a time, whereas the **OData Service** artifact allows you to expose multiple entities and operations

at a time. The topic of OData is covered several times within this book, for example, in the context of the SAP Cloud Platform APIs ([Chapter 9](#)) or when describing a scenario that includes an OData adapter (see [Chapter 4, Section 4.3](#)). [Chapter 4, Section 4.5](#), finally, will explicitly introduce you to the topic of OData service design.

In the dialog box shown in [Figure 2.20](#), make sure the **Create** radio button is selected, and enter a name for the integration flow in the **Name** field.

The **Upload** option (next to the **Create** option in [Figure 2.20](#)) allows you to upload an integration flow file from your computer to the Web UI.

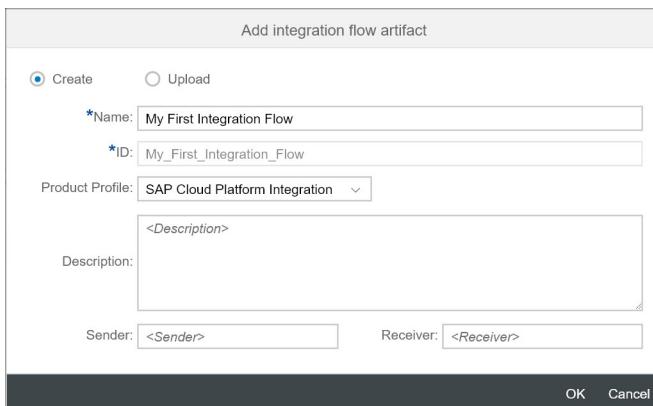


Figure 2.20 Creating an Integration Flow

6. Click **OK**. The specified integration flow is listed as a new artifact for your integration package (see [Figure 2.21](#)).

HEADER	OVERVIEW	ARTIFACTS (1)	DOCUMENTS (0)	TAGS
		Add		
<input type="checkbox"/> Name		Type		
<input checked="" type="checkbox"/> My First Integration Flow		Integration Flow		
Created				

Figure 2.21 The Newly Created Integration Flow Listed as a New Artifact

7. Click the **Save** button (located above the **Artifacts** list) to save the integration package.

8. When you click the integration flow name (in the **Artifacts** list, see [Figure 2.21](#)), the editor opens and shows a template where you can begin modeling ([Figure 2.22](#)).
9. Click the **Edit** button shown in [Figure 2.22](#) to start modeling the message flow. You'll notice that a palette becomes visible on the left side of the editing area. Behind this palette, you can select elements that can be added to an integration flow. Before we continue, let's briefly discuss the palette and the available modeling elements.

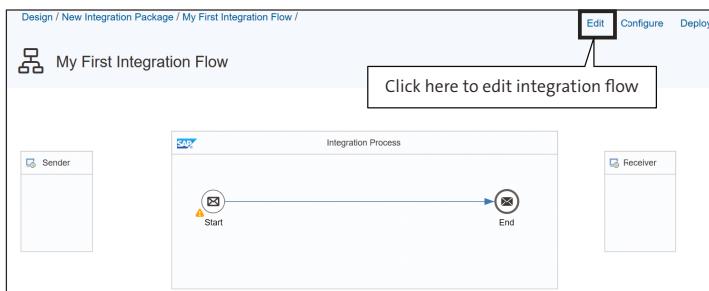


Figure 2.22 Integration Flow Template to Start Modeling

The Palette: Elements of an Integration Flow

The integration flow modeler's palette (together with the tooltips for each icon group) is shown in [Figure 2.23](#).

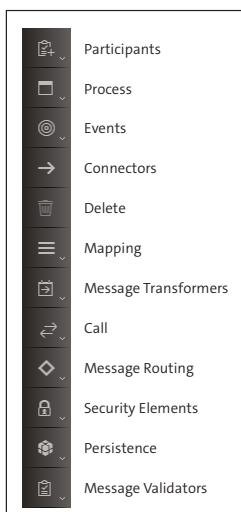


Figure 2.23 Palette of the Integration Flow Modeler

Table 2.4 summarizes the modeling elements available at the time this book published, sorted by group.

Group	Description
Participants	Contains elements that represent the connected participants of an integration scenario: Sender , Receiver .
Process	Contains elements that can be used as containers for a whole integration flow (Integration Process), a subprocess (Local Integration Process), or a subprocess that handles exceptions that occur during message processing (Exception Subprocess). You can use Local Integration Process elements to source out parts of the process logic, which are invoked from the main process, into smaller chunks. That way, you can keep larger integration flows at a reasonable size. Local integration processes and exception subprocesses are introduced in Chapter 6, Section 6.3 .
Events	Contains event elements to define the beginning or the end of message processing. You can select from the following step types: End Message , End Event , Error End Event , Escalation , Start Message , Start Event , Terminate Message , and Timer (see Chapter 6, Section 6.1).
Mapping	Allows you to insert a Mapping step to transform a source message into a target message. You can select from the following step types: Message Mapping and XSLT Mapping (see Chapter 4, Section 4.4).
Message Transformers	Contains elements to modify the message content by applying different operations, such as encoding (e.g., using Base64), conversion (e.g., from CSV to XML), or script functions. You can select from the following step types: Content Modifier (see Chapter 4, Section 4.1.4), Converter , Decoder , Encoder , Filter , Message Digest , or Script .

Table 2.4 Integration Flow Modeling Elements Offered in the Palette

Group	Description
Call	<p>Contains elements to enable the tenant to call an external source or a local integration process.</p> <ul style="list-style-type: none"> ■ External Call Enables the tenant to call an external source (e.g., to retrieve data from external sources, such as SOAP or OData, and to enrich the message with it). You can select from the following step types: Content Enricher, Request Reply (see Chapter 6, 6.2.1), or Send (see Chapter 5, Section 5.3.3). ■ Local Call Calls a local integration process either once (Process Call) or in a loop (Looping Process Call) (see Chapter 6, Section 6.3.2).
Message Routing	<p>Contains elements to forward the message to different receivers, to split larger messages, or to combine multiple messages into a larger one. You can select from the following step types: Gather (see Chapter 5, Section 5.2.1), Router (see Chapter 5, Section 5.1.1), Splitter (see Chapter 5, Section 5.2.1), Join, Multicast (see Chapter 5, Section 5.3.3), or Aggregator.</p>
Security Elements	<p>Contains elements to decrypt/verify incoming messages, or to encrypt/sign outbound messages. You can select from the following step types: Decryptor, Encryptor, Signer, or Verifier (see Chapter 10).</p>
Persistence	<p>Contains elements to store message content at specific steps within the message processing. You can select from the following step types: Data Store Operations, Persist Message, or Write Variables.</p>
Message Validators	<p>Allows you to add an XML Validator step.</p>

Table 2.4 Integration Flow Modeling Elements Offered in the Palette (Cont.)

Furthermore, it's also possible to use the **Select** and **Sort** options, which allow you to rearrange elements in an integration flow model but don't offer additional elements. The **Delete** option allows you to delete a selected element from the graphical editor.

Design the Integration Flow

Let's start designing our first integration flow using the **Sender** element. In the palette, navigate to **Participants • Sender**. After selecting and adding the sender element, follow these steps:

1. Position the cursor on the **Sender** shape. You'll notice that an information icon, a recycle bin symbol, and an arrow icon appear (as shown in [Figure 2.24](#)).



Figure 2.24 Information, Recycle Bin, and Arrow Icons for an Element

The arrow icon is used to connect elements of an integration flow (as shown in the next step). The recycle bin symbol is self-explanatory.

Information Icon

The information icon shows technical information about the selected integration flow element, namely, the ID and the version of the element. The ID is important to relate the information provided in the MPL (see [Chapter 8, Section 8.2](#)) to a certain integration flow shape (when monitoring the integration flow); you'll learn more about the version of an element in [Chapter 6, Section 6.5](#).

2. Select the arrow icon, and drag and drop it to the integration flow start event (an orange dashed line will track your path) (see [Figure 2.25](#)).
3. A dialog box opens where you can select the sender **Adapter Type** (see [Figure 2.26](#)).



Figure 2.25 Creating a Connection between Sender and Start Event

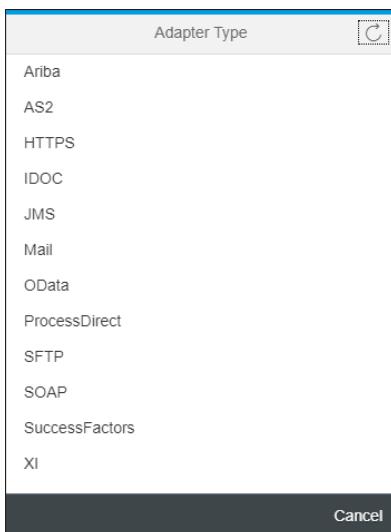


Figure 2.26 Offered Sender Adapter Types

4. Select **SOAP**. In the next dialog box that appears, select **SOAP 1.x**.
5. After this step, in the section below your model, a sheet appears where you can specify the properties of the adapter in different tabs.
6. Leave the settings under **General** as they are and choose the **Connection** tab. Here you specify settings as shown in [Figure 2.27](#). These settings include:
 - **Address:** This is the specified string (including the forward slash at the beginning) necessary to generate the endpoint URL required by SoapUI. Here, enter “/soap2mail” in the field.

- **Service Definition:** With this parameter, you specify the source of the service definition to be used for the SOAP channel. We keep the default setting as **Manual**. We use the WSDL downloaded from the book's website only on the sender (SOAP client) side to specify the structure of the message.
- You can choose the alternative option **WSDL** if the SOAP sender adapter is supposed to use information contained in a dedicated WSDL file (to be specified) to determine how the message is to be processed. In the WSDL file, you can provide more information for the integration runtime to determine how the message is to be processed, for example, if a request is to be treated as a synchronous or asynchronous call, and the runtime node can treat the message accordingly. We go into the details of this in [Chapter 5, Section 5.3.1](#).
- **Message Exchange Pattern:** With this parameter, you specify the communication type. Either keep the default setting as **Request-Reply** or set it to **One-Way**. In the first case, the adapter sends a reply message back to the sender; in the second case it doesn't. When you've selected **One-Way**, remember to keep the **Processing Settings** as **WS Standard**. Changing this setting to **Robust** would cause processing errors to be returned to the SOAP client.
- **Authorization:** This setting makes sure that for the user associated with the calling sender, the permissions are checked based on user-to-role assignments by the SAP Cloud Platform Integration framework. Here, keep the setting **User Role**. In the **User Role** field, keep the entry **ESBMessaging.send**. This role is pre-defined by SAP to authorize a sender (the SOAP client) to call your tenant. The **Select** option allows you to select a custom role if you've defined one (see [Chapter 10, Section 10.3.3](#) for more details).

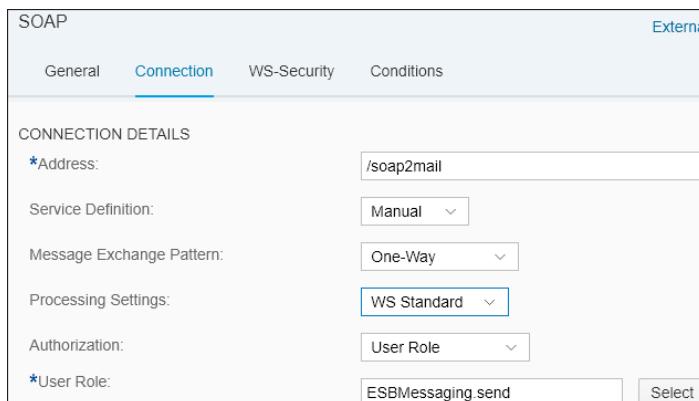


Figure 2.27 SOAP Adapter Settings

User Role Authorization

The **User Role** authorization option can be used along with *basic authentication* (among other authentication options, see the following *Inbound Authorization Options* information box). Using this authentication option, SAP Cloud Platform Integration expects credentials (username and password) to authenticate the user associated with the incoming call (compare the analog configuration of the SOAP client as shown later in [Figure 2.40](#)).

The integration flow model is again displayed. The only components missing in our integration flow model are the email receiver and the mail adapter. We explain how to configure the mail adapter in the next section.

Inbound Authorization Options

There are different options to combine inbound *authorization* with *authentication* methods. The two authorization options offered in the SOAP sender adapter (as well as in most HTTP-based adapters) can be combined with authentication in the following way:

- **User Role** authorization

The permissions of the caller are checked based on roles defined for the user associated with the inbound call. Hereby, three *authentication options* are possible. First, the user can be authenticated based on credentials (which are provided in the HTTP header of the call), which is referred to as *basic authentication*. Second, the user can be evaluated by a *certificate-to-user mapping*. In this case, the sender authenticates itself against SAP Cloud Platform Integration based on a digital client certificate (which is mapped to a user in a subsequent step). To configure this option, an additional **Certificate-to-User Mapping** artifact needs to be defined for the scenario. As a third option, authentication can be established based on *OAuth*, an option to grant access to SAP Cloud Platform Integration without explicit credentials sharing.

- **Client Certificate** authorization

Using this option, the permissions of the caller are checked by evaluating the distinguished name (DN) of the client certificate provided by the caller. Obviously, the caller is authenticated based on a client certificate.

This topic is covered in detail in [Chapter 10](#). You can also check out the documentation of SAP Cloud Platform Integration at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud by choosing **Connecting a Customer System to Cloud**

Integration • Concepts of Secure Communication • Basics • HTTPS-Based Communication • Authentication and Authorization Options (Inbound). You can find additional helpful information in the “Cloud Integration – How to Set Up Secure HTTP Inbound Connection with Client Certificates” blog (<https://blogs.sap.com/2017/06/05/cloud-integration-how-to-setup-secure-http-inbound-connection-with-client-certificates/>) published in SAP Community.

Adding and Configuring the Mail Adapter

To add and configure the mail receiver adapter, perform the following steps:

1. Click the **End** event so that the context buttons (the information, arrow, and recycle bin icons, from top to bottom) appear next to the shape, as seen in [Figure 2.28](#).



Figure 2.28 Context Buttons for the End Event

2. Connect the **End** event with the **Receiver** pool by clicking the arrow icon, dragging the cursor to the target shape (the **Receiver** pool in that case), and then releasing the mouse button (drop) (as seen in [Figure 2.29](#)).



Figure 2.29 Connecting the End Event with the Receiver Pool

3. After releasing the mouse button, the **Adapter Type** dialog box will appear, which allows you to select the connection type you desire ([Figure 2.30](#)).
4. In the **Adapter Type** dialog, click on the **Mail** entry.

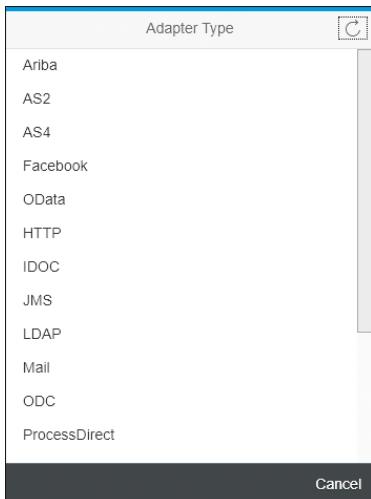


Figure 2.30 Adapter Type Dialog

5. After this step, in the section below your model, a sheet appears where you can specify the properties of the mail adapter in different tabs.

In the **General** tab, you find the basic settings for your mail receiver:

- Under **Channel Details**, you find information on the direction of the channel (directing from Cloud Integration to a receiver system) as well as the name of the connected system (**Receiver** in that case).
- Under **Adapter Details**, the adapter type (Mail) is again shown as well as the supported transport protocol. For a mail receiver adapter, only Simple Mail Transfer Protocol (SMTP) is supported.

You don't have to change anything here (Figure 2.31).

6. Click the **Connection** tab and configure the adapter according to your email account. The values shown in the screenshots (Figure 2.32) fit a Google email account. We briefly explain the parameters.

In the mail channel under **Connection Details**, specify the settings as shown in Figure 2.32.

In the **Address** field, specify the address of your mail server, in this case, Gmail. To find it in this example, check out the support information provided for Google at <https://support.google.com>. In this field, you also need to add the port, separated from the address by a colon. Which port to use follows from the other settings, as explained in the following paragraphs.

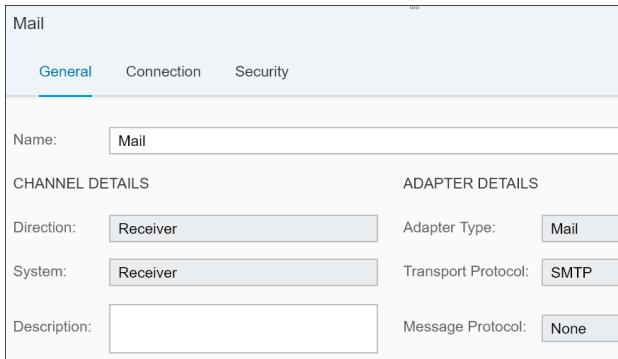


Figure 2.31 Configuring the Mail Adapter: General Tab

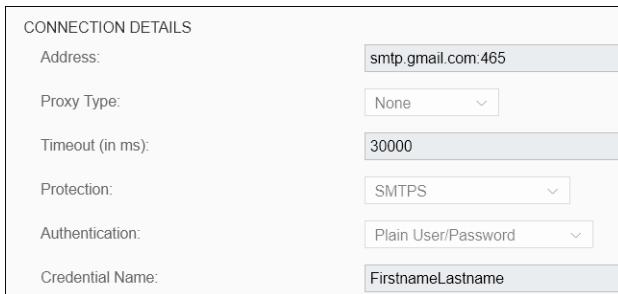


Figure 2.32 Configuring the Mail Adapter: Connection Tab/Connection Details

Allowed Ports for the Mail Receiver Adapter

Note that there are certain restrictions regarding the supported ports for the mail receiver adapter. In particular, you can use the following ports (for different **Protection** settings of the mail receiver adapter):

- 587 for SMTP+STARTTLS
- 465 for SMTPL

As we propose to choose **SMTPL** as **Protection**, port **465** must be also specified in the **Address** field, as shown in [Figure 2.32](#).

The **Credential Name** field, as shown in [Figure 2.32](#), contains a simple string that refers to a deployed credentials artifact on the SAP Cloud Platform Integration server. You can't define the username and password directly on the configuration screen of the mail adapter; instead, it's necessary to deploy the credentials

containing the username and password on the server explicitly. This step is necessary for connections with basic authentication, as is the case for this email connection. When deploying the credentials on the server, you must provide a unique name (e.g., “FirstnameLastname”, in our case) for reference purposes. This is the exact name you have to fill in the **Credential Name** field. If you have the rights for deploying (ask your tenant administrator), you can execute the steps described in [Section 2.3.5](#).

7. In the **Connection** tab under **Mail Attributes**, most settings are self-explanatory ([Figure 2.33](#)). You specify the sender mail address (**From** field) and receiver address (**To** field) mail address (optionally, you can specify the mail’s carbon copy [**CC** field] and blind carbon copy [**BCC** field] as well). Furthermore, add a text string which will then be written into the mail subject (**Subject** field).

The screenshot shows the 'MAIL ATTRIBUTES' configuration screen. It includes fields for 'From' (Fn.Ln@gmail.com), 'To' (Fn.Ln@gmail.com), 'Cc' (empty), 'Bcc' (empty), 'Subject' (Test Mail for Book Demo), 'Mail Body' (\${in.body}), 'Body Mime-Type' (Text/Plain), 'Body Encoding' (UTF-8), and attachments (Add, Delete buttons). Below these are tables for attachments and message attachments.

Name	Mime-Type	Source	Header Name
<input type="checkbox"/>			

Add Message Attachments
<input type="checkbox"/>

Figure 2.33 Configuring the Mail Adapter: Connection Tab/Mail Attributes

Note

The default definition of the **Mail Body** field (`${in.body}`) makes use of Camel’s Simple Expression Language. We’re explicitly accessing the `in` message of the exchange and taking its body, which contains nothing but the message’s payload and is exactly what we want to see in our email. The Camel data model is described in more detail in [Chapter 4, Section 4.1](#).

With the **Body Mime-Type** setting, you specify the Internet Media Type of the message body, that is, the kind of data transferred with the message. Keep the default value **Text/Plain** because no other data format other than plain text (with the XML describing the message structure) is forwarded directly from the SOAP client. **Body Encoding** allows you to specify the character encoding of the incoming data. To ensure that data is passed unmodified, keep the default value **UTF-8** (Unicode encoding).

Finally, under **Attachments**, you can specify that the received mail should contain the outbound message as attachment. We don't define any attachment settings in the first integration flow.

For more details on how to configure the mail adapter, take a look at the online documentation of SAP Cloud Platform Integration. Go to https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud, and search for "mail adapter".

The final scenario should look similar to the one depicted earlier in [Figure 2.13](#). You can easily identify the sender on the left (connected to the **Integration Process** shape by a SOAP channel), and the final receiver on the right (connected with the **Integration Process** shape by the mail channel configured just in the previous steps). The message invoked by the SOAP client (which simulates our sender system) is passed on without any further changes to the Mail adapter.

That's the advantage of a graphical environment: it clearly and intuitively describes how the message arrives at the server, how it's handled within the SAP Cloud Platform Integration server, and to which systems using which channels it's forwarded.

Save the integration flow, and click **Deploy**. You'll find the corresponding buttons for saving and deploying the integration flow on top of the integration flow editor ([Figure 2.34](#)).



Figure 2.34 Save and Deploy Buttons at the Bottom of the Editor

2.3.5 Creating and Deploying a User Credentials Artifact

To enable the tenant to connect to the email receiver using the credentials of the email account owner, you needed to add a **Credential Name** in the mail adapter of

your integration flow, which at this point is little more than a placeholder for an artifact that we'll create using the following steps:

1. Choose the **Monitor** tab of the Web UI (refer to [Figure 2.10](#)).
2. Select the **Security Material** tile under **Manage Security**.
3. Choose **Add • User Credentials** (see [Figure 2.35](#)).

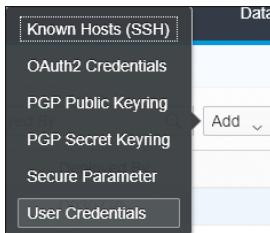


Figure 2.35 Adding a User Credentials Artifact

4. Specify the properties of the **User Credentials** artifact. For **Name**, enter the value that you entered in the **Credential Name** field in the mail adapter (refer to [Figure 2.32](#)). For **User**, enter your email box username, and as **Password/Repeat Password**, enter the associated password. Leave the **SuccessFactors** checkbox deselected. This setting is only relevant when you define credentials to be used when connecting to a SAP SuccessFactors system (see [Figure 2.36](#)).

A screenshot of the 'Add User Credentials' dialog box. The title bar says 'Add User Credentials'. The form contains the following fields:

- *Name:
- Description:
- *User:
- Password:
- Repeat Password:
- SuccessFactors

At the bottom right of the dialog are 'Deploy' and 'Cancel' buttons.

Figure 2.36 The Add User Credentials Dialog Contains the Properties of a User Credentials Artifact

5. Choose **Deploy**.

How Secure Are Your User Credentials?

You've now defined an artifact that contains the credentials used by the tenant to connect to your email account using the mail adapter. Configuring the integration scenario, it wasn't necessary to share these credentials (username and password) with anyone. In the mail adapter settings, only an alias (**Credential Name**) is specified, which the other participants of your integration team sharing the same tenant can see without any risk of a security leak.

Another artifact type, which is handled in the same way, is the **Secure Parameter** artifact required for scenarios that include social media adapters (for the Twitter adapter, see [Chapter 10, Section 10.4.5](#)) and when you use the Adapter Development Kit (ADK) (see [Chapter 6, Section 6.7](#)).

If you don't have deployment rights, ask the tenant administrator to take over the process for you.

You've almost reached the end of this first tutorial. However, there is one additional thing you have to do before you can successfully run your scenario and send the SOAP message: import certificates.

2.3.6 Import Certificate Required by the Mail Server into Keystore

The mail receiver adapter that you've configured specifies an outbound connection to an email server. To increase security, connections between an SAP Cloud Platform Integration tenant and remote systems can be protected by various methods, as you'll learn in [Chapter 10](#). In our example integration flow, the tenant (as client) authenticates itself against the email server with the credentials (user and password). You've specified the required settings in the **User Credentials** artifact in the steps before.

However, in the other direction, SAP Cloud Platform Integration also needs to establish a trust relationship to the email server. When establishing the connection, the email server needs to authenticate itself against SAP Cloud Platform Integration to prove its trustworthiness. This is done based on a digital *server* certificate. SAP Cloud Platform Integration can only confirm trustworthiness of the email server when the keystore owned by SAP Cloud Platform Integration (deployed on the tenant) contains a root certificate that is also trusted by the email server. (For more information about certificates, see [Chapter 10, Section 10.4.1](#).)

We'll now show you can get the required certificate into the tenant keystore (if not part of it already). Although you can find out which certificate needs to be in the tenant keystore from the organization that runs the email server (Google in our example at <https://pki.google.com/>), there is a smarter way (without needing to look it up online and download it): the outbound connectivity test tool. This tool is part of the **Monitor** application, which will be described in detail in [Chapter 10](#). Therefore, we keep it short here.

1. Open the **Monitor** tab on the Web UI (refer to [Figure 2.10](#)).
2. Under **Manage Security**, click the **Connectivity Test** tile.
3. On the **Overview/Test Connectivity** screen, choose the **SMTP** tab to open the test options for the (outbound) connection to the email receiver ([Figure 2.37](#)).

The screenshot shows the 'Overview / Test Connectivity' page with the 'SMTP' tab selected. The 'Request' section contains fields for Host (smtp.gmail.com), Port (465 (SMTPS)), and Protection (SMTPS). Under Authentication, 'None' is selected. Below the fields are two checkboxes: 'Validate Server Certificate' (unchecked) and 'Check Mail Addresses' (unchecked). A blue 'Send' button is located at the bottom right.

Figure 2.37 SMTP Outbound Connectivity Test

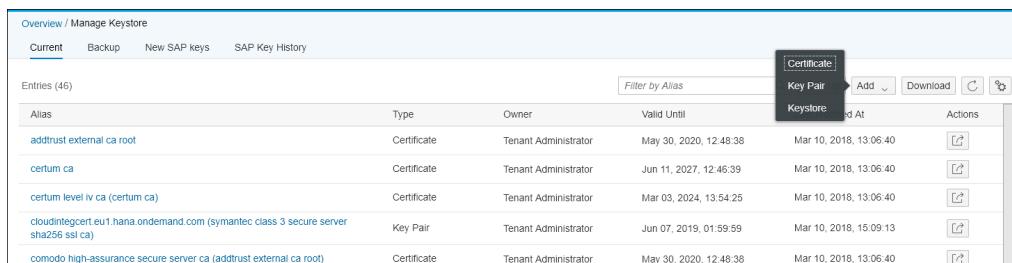
4. For **Host**, enter “smtp.gmail.com”, and, for **Port**, select **465 (SMTPS)** (as these are the settings also specified in the mail receiver adapter, compare with [Figure 2.32](#)).
5. For **Authentication**, select **None**.
6. Deselect the **Validate Server Certificate** checkbox.
7. Choose **Send**.
8. As a response, you'll receive the message: **Successfully reached host at smtp.gmail.com:465**. Details of the server certificate are also displayed.
9. Choose **Download**.
10. The certificate is downloaded as a compressed file to your computer (*certificates.zip*).
11. Extract the certificate in a directory.

Next, you need to import the certificate into the tenant keystore, which has been provided to you by SAP together with the tenant by following these steps.

Note

If you don't have permissions for the following security-related steps, let the tenant administrator do it for you.

1. In the **Monitor** tab on the Web UI under **Manage Security**, click the **Keystore** tile. All certificates that are contained already in the tenant keystore are displayed in a table (Figure 2.38).



Alias	Type	Owner	Valid Until	Keystore	Created At	Actions
addtrust external ca root	Certificate	Tenant Administrator	May 30, 2020, 12:48:38	Mar 10, 2018, 13:06:40	[Edit]	[Delete]
certum ca	Certificate	Tenant Administrator	Jun 11, 2027, 12:46:39	Mar 10, 2018, 13:06:40	[Edit]	[Delete]
certum level iv ca (certum ca)	Certificate	Tenant Administrator	Mar 03, 2024, 13:54:25	Mar 10, 2018, 13:06:40	[Edit]	[Delete]
cloudintegcert.eu1.hana.ondemand.com (symantec class 3 secure server sha256 ssl ca)	Key Pair	Tenant Administrator	Jun 07, 2019, 01:59:59	Mar 10, 2018, 15:09:13	[Edit]	[Delete]
comodo high-assurance secure server ca (addtrust external ca root)	Certificate	Tenant Administrator	May 30, 2020, 12:48:38	Mar 10, 2018, 13:06:40	[Edit]	[Delete]

Figure 2.38 The Keystore Monitor

2. Click **Add** and then select **Certificate** as shown in Figure 2.38.
3. Select the extracted certificate from your computer. The list of certificates in the keystore monitor is refreshed and shows the imported certificate.

2.3.7 Send the SOAP Message

That's it! You've designed your first integration flow and finished the additional configuration settings. Now it's time to put this integration flow to use by sending a SOAP message.

Now that SAP Cloud Platform Integration is configured and prepared to process a message, let's send a SOAP message by following these steps:

1. Open SoapUI, and double-click **Request 1** in the expanded tree on the left side of the split screen (see Figure 2.39).
2. To tell SoapUI where to send the message, you need to specify an endpoint. In the dropdown list in the header of the **Request 1** window, select [**add new endpoint...**] (see Figure 2.39).

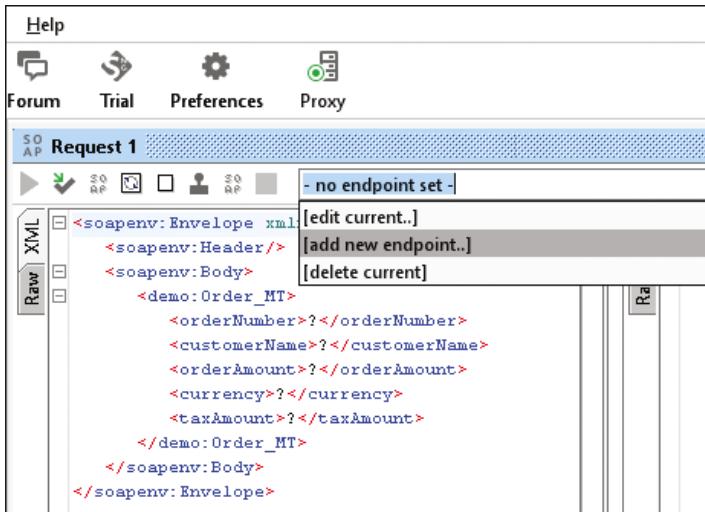


Figure 2.39 Adding a New Endpoint in SoapUI

- In the next screen, enter the endpoint URL, and choose **OK**. The endpoint URL is composed of the runtime URL provided by SAP in the mail about your tenant, the string `/cxsf`, and the service address defined in the SOAP adapter (in the demo example, `/soap2mail`) in the following way: *https://<Runtime URL provided by SAP>/cxsf/soap2mail*.

You can also find the endpoint URL by opening the **Monitor** tab of the Web UI and clicking on a tile under **Manage Integration Content**. In the list of deployed integration content, find your newly deployed integration flow, and click the corresponding entry. Copy the URL you find in the **Endpoint** field.

- Now you need to enable the SOAP client to authenticate itself against SAP Cloud Platform Integration based on your user credentials. In SoapUI, click **Auth (Basic)**, and enter your credentials in the next screen (see [Figure 2.40](#)). Refer also to [Figure 2.16](#) for the assignment of the corresponding role `ESBMessaging.send` to the user associated with the sending SOAP client.
- Close the dialog box by clicking **Auth (Basic)** again. Make any entries for the elements in the **XML structure**, which can be found in the **Request 1** window. For example, enter your name between the opening and the closing tag for **Customer Name**, as shown in [Figure 2.41](#).

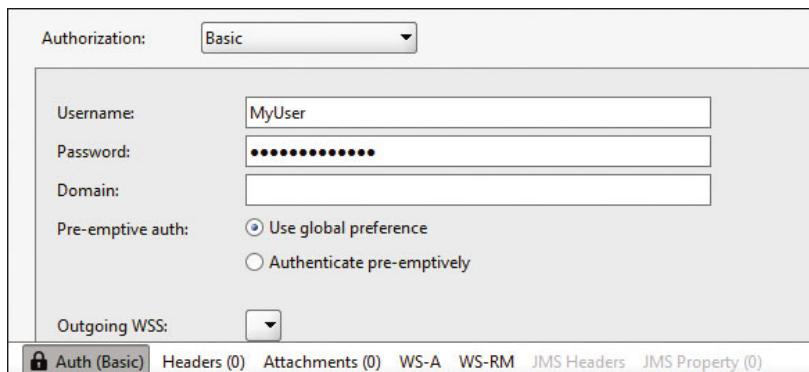


Figure 2.40 Specifying Username and Password to Enable the SOAP Client to Send Messages to SAP Cloud Platform Integration

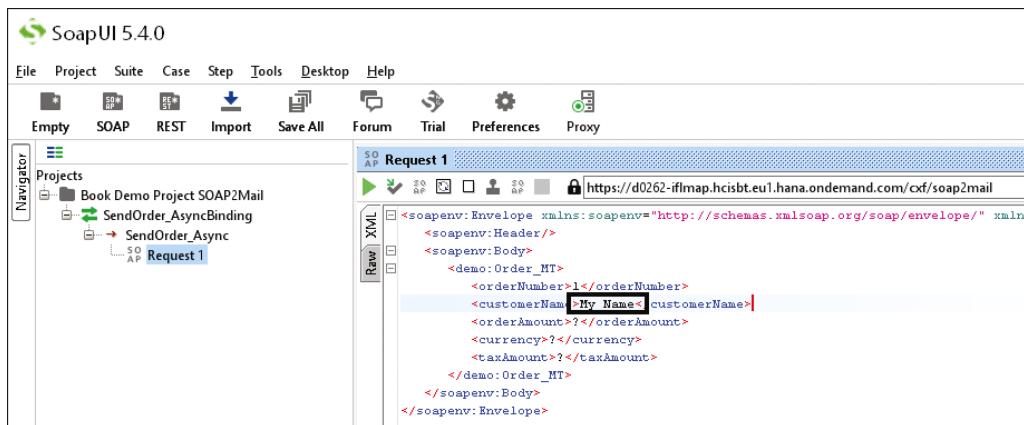


Figure 2.41 Entering Data in the XML Structure

6. To send the message, click on the green triangle icon (at top left corner of [Figure 2.42](#); tooltip **Submit request to specified endpoint URL**), or press **Alt**+**Enter**.

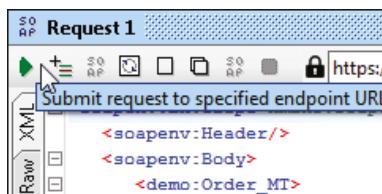


Figure 2.42 Clicking the Green Triangle Icon to Send the Message

The message will be delivered to your email account. You should receive an email like the one shown in [Figure 2.43](#).

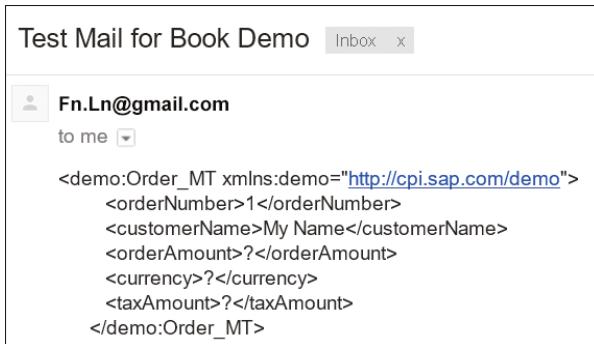


Figure 2.43 Email Content Shows Message Structure

Troubleshooting

If you're using a Google email account, as we've done in this exercise, consider the following:

- You might need to temporarily allow less secure apps to access your account. Otherwise, Google email will refuse your connection attempt via SAP Cloud Platform Integration. Note that this is just for test purposes. You should revert the settings in your Google email account after you've verified the sending of emails via SAP Cloud Platform Integration. More details can be found on the Internet. Search for “Google email allow less secure apps to access your account” or directly navigate to <https://support.google.com/accounts/answer/6010255>.
- If you receive an error message, such as **javax.net.ssl.SSLHandshakeException: unable to find valid certification path to requested target**, the reason is a missing certificate. You have to add the Google certificate to your keystore on your tenant (as explained in [Section 2.3.6](#)).

2.3.8 Monitor the Message

More information on the monitoring application, and the meaning of each section, will be provided in [Chapter 8](#). For now, we'll only show you how to quickly check the processing status of your first integration flow.

1. Open the **Monitor** tab on the Web UI (refer to [Figure 2.10](#)).
2. To check for the message that you've just sent, select a tile with suitable filter criteria under **Monitor Message Processing** (see [Figure 2.44](#)).

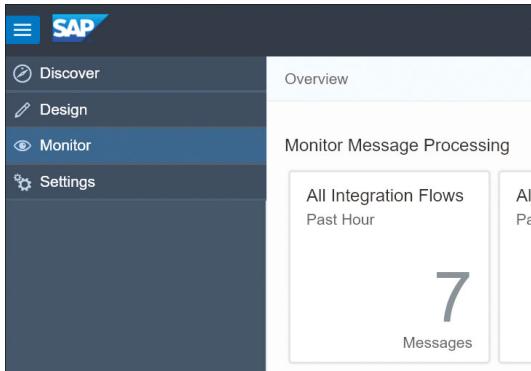


Figure 2.44 Monitor Message Processing Tile in the Monitoring Tab

A list of messages is displayed. Details about the processed message (for which the row is selected at the left side) are shown on the right side of the window ([Figure 2.45](#)).

The screenshot shows the 'Monitor Message Processing' page with the following details:

- Time:** Past Month (May 11, 2018, 10:29:51 - Jun 10, 2018, 10:29:51)
- Status:** All
- Artifact:** All Integration Flows
- Messages (18):** A list of 18 messages, with the first three rows visible:

Artifact Name	Status
My Second Integration Flow	Completed
Jun 10, 2018, 10:29:47	2 sec 197 ms
My First Integration Flow	Completed
Jun 10, 2018, 10:29:33	2 sec 400 ms
My First Integration Flow	Completed
Jun 10, 2018, 10:29:30	2 sec 314 ms
- My First Integration Flow:** Last Updated at: Jun 10, 2018, 10:29:33
 - Status:** Completed
 - Properties:** Message processing completed successfully.
 - Logs:** Processing Time: 2 sec 400 ms

Figure 2.45 Detailed List of Processed Messages and MPL

3. The **Logs** tab shows a link to the MPL, which provides information on the processing of the message. Click **Open Text View** to display the processing steps of the selected message in a structured view ([Figure 2.46](#)).

You can configure different log levels to display information about the processing of the message on different detail levels. For more information on this and on the MPL, see [Chapter 8](#).

The screenshot shows the SAP Cloud Platform Integration interface. The top navigation bar includes 'Overview / Monitor Message Processing / Message Processing Log Attachments'. Below this, artifact details are listed: 'Artifact Name: My First Integration Flow', 'Status: Completed', 'Processing Time: 2 sec 400 ms', 'Last Updated at: Jun 10, 2018, 10:29:33', and 'Log Level: Info'. A 'Log' tab is selected, displaying the 'Message Processing Log' with various parameters and their values.

Parameter	Value
StartTime	Sun Jun 10 08:29:31.457 UTC 2018
StopTime	Sun Jun 10 08:29:33.855 UTC 2018
OverallStatus	COMPLETED
MessageGuid	AFsc4WsgLGrdAM61MnbIhbtsq20M
ChildCount	0
ChildrenCounter	18
ContextName	My_First_Integration_Flow
CorrelationId	AFsc4WumbDyPuChHwu2j4Lpvq_17
IntermediateError	false
Node	vsa4268206
ProcessId	091ddcc47d972a1bb68e5712ce9fb0af8c5e21c2
SenderId	SenderId_SOAP

Figure 2.46 MPL

4. To check the status of the deployed integration flow, select a tile under **Manage Integration Content** ([Figure 2.47](#)).

The screenshot shows the SAP Cloud Platform Integration interface with the 'Monitor' option selected in the sidebar. The main area displays the 'Manage Integration Content' screen. It shows a table with three rows, each representing an integration artifact: 'My First Integration Flow' (Status: Started), 'Integration Flow' (Status: Started), and 'Advanced Flow' (Status: Started). A filter bar at the top allows searching by name or ID.

Name	Status
My First Integration Flow	Started
Integration Flow	Started
Advanced Flow	Started

Figure 2.47 Managing Integration Content Screen

The page in [Figure 2.47](#) shows all integration artifacts (integration flows, value mappings, and OData services) that have been deployed on the tenant. This is a useful page for administrators who like to check the status of the integration artifacts. We'll provide more information on this in [Chapter 8](#).

Congratulations! You've successfully processed your first message.

Mail Sender Adapter

SAP Cloud Platform Integration also offers the option to connect an email sender using a mail sender adapter. An integration flow with this adapter can read emails from a specified email account and further process them.

You can enhance your first integration flow by replacing the SOAP sender channel with a mail sender channel and trying out this feature. For simplicity, you can use the same Gmail account and email address you've specified for the mail receiver adapter. That way, you use SAP Cloud Platform Integration to send yourself an email.

You can quickly set up this scenario using the following steps and settings:

1. Remove the SOAP channel between **Sender** and **Start** event (by clicking the connection and selecting the recycle bin icon), as shown in [Figure 2.48](#).



Figure 2.48 Removing an Existing Channel

2. Create a new channel, and select adapter type **Mail • IMAP4**.
3. In the **Connection** tab of the mail sender adapter, specify the following settings:
 - **Address:** Enter “imap.gmail.com:993”. For more information on the allowed ports for the sender mail adapter, refer to the online documentation of SAP Cloud Platform Integration. For your convenience, we copy the list of allowed ports at the bottom of this box.
 - **Authentication:** Choose **Plain User/Password** to keep it simple, and use the same **Credential Name** as you did for the mail receiver adapter (i.e., “Firstname-LastName”).
4. In the **Processing** tab, we recommend using the following settings:
 - **Selection:** Choose **Only Unread** (otherwise, the adapter would pick up all mails from the specified folder).
 - **Folder:** Specify a certain folder in your email account (other than the Inbox) so that SAP Cloud Platform Integration will take unread message from the Inbox and copy them to the specified folder.
 - **Post-Processing:** Specify what should happen with a mail once it has been processed by SAP Cloud Platform Integration.

5. Be careful with the settings in the **Scheduler** tab not to spam your inbox with your own mails after having deployed the integration flow. But as you've specified that only unread mails will be processed, you might be on the safe side (for **Selection**, you've chosen **Only Unread**).

Be aware of certain security risks when using mail sender channels, as SAP Cloud Platform Integration can't authenticate the sender of an email. For more information, check out the online documentation of SAP Cloud Platform Integration in the mail adapter section.

Allowed Ports for Mail Sender Adapter

In particular, you can use the following ports (for different **Protection** settings of the mail sender adapter):

- 143 for **IMAP+STARTTLS**
- 993 for **IMAPS**
- 110 for **POP3+STARTTLS**
- 995 for **POP3S**

A Simple Smoke Test Integration Flow

The described integration flow was quite simple: it didn't do anything further with the message than simply forwarding the content received by a sender component to an email receiver. However, even for this first flow, you needed to configure a receiver, which, in the context of this exercise means you needed to configure your email account in such a way that it accepts messages from SAP Cloud Platform Integration. The "Troubleshooting" box in [Section 2.3.7](#) showed that even here, you can already run into errors.

If you like to quickly just check if your cluster works correctly, you can set up a "smoke test" integration flow without any receiver system involved. To find out how to do this, check out the online documentation of SAP Cloud Platform Integration at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud (in the **Getting Started with SAP Cloud Platform Integration (Onboarding Guide)** section, select **Performing a Smoke Test**).

2.4 Summary

In this chapter, we provided you with a detailed introduction to the architecture of SAP Cloud Platform Integration. We explained its main components and showed you how messages are processed by the virtual runtime environment. We also introduced you to the available tools and processes, and we then finished up the chapter with a brief tutorial on how to design and run your first, very simple, integration flow. The tutorial showed you how to use SAP Cloud Platform Integration to exchange messages between different systems. Using a SOAP client and an email receiver system allowed you to set up a very simple system landscape without further technical prerequisites to meet. The first integration scenario, however, didn't impose any further processing of the message. If you can't wait to find out how to start modifying a message while processing it, you can proceed with [Chapter 4](#) where step-by-step methods for processing a message will be introduced.

However, now that you're equipped with the basic knowledge required to dive further into the world of integration patterns, you might first want to find out what integration scenarios are provided by SAP out of the box. The next chapter provides you with an overview of the predefined integration content provided by SAP in the Integration Content Catalog.

Chapter 3

Integration Content Catalog

SAP provides prepackaged integration content that enables quick implementation of integration scenarios. These packages are found in the Integration Content Catalog. The chapter dives into the specifics of the Integration Content Catalog, presents its available features, and explores the prepackaged integration content currently available for customers.

In [Chapter 1](#), we discussed the role that SAP Cloud Platform Integration plays within SAP's cloud strategy. We also discussed SAP Cloud Platform Integration's positioning within the SAP landscape and presented a number of use cases.

As the adoption of cloud-based applications keeps growing, the chance of more customers needing to build the same integration scenarios between these cloud-based applications will continue to increase. Why not build these common integration scenarios in advance and reduce the implementation cost for customers? With this approach, customers simply need to reuse existing integration scenarios, rather than build their own.

That is exactly what SAP has made available through its Integration Content Catalog. This chapter introduces the catalog and takes you through the different steps required to consume its prepackaged SAP-provided contents. The chapter further explores currently provided prepackaged content and then discusses some of their use cases.

3.1 Introduction to the Integration Content Catalog

Since the introduction of cloud computing technology, there has been a shift in the level of investment in software licenses by organizations. Organizations are rapidly moving from the concept of software ownership to software rental. As a result, SAP is also growing its cloud portfolio with standardized products in human resources

(with SAP SuccessFactors), marketing, sales, service, procurement, supply chain management, and finance. We're convinced that this portfolio will continue to grow.

For most customers, the need might arise to integrate cloud-based applications with other on-premise or cloud-based applications to cover the total end-to-end business process. As explained in previous chapters, SAP Cloud Platform Integration is well positioned as the integration platform for such a use case.

Most capabilities provided by these cloud-based applications are standardized in terms of protocols, endpoints, and message structures. Therefore, integration scenarios built over SAP Cloud Platform Integration for these cloud-based applications have a good chance of being implemented and reused by many other customers and partners. That is exactly what SAP provides with its prepackaged integration content for the most frequently used SAP cloud-based applications. This integration content is available in the Integration Content Catalog.

The catalog covers templates with prebuilt integration flows, value mappings, and other integration artifacts that customers can reuse, enabling customers to significantly reduce implementation time, cost, and risk. The catalog presents and categorizes content in a simple manner, allowing customers to browse and discover content that might be relevant for their scenarios. The content of the Integration Content Catalog is bundled in packages. Each package contains artifacts and objects that logically belong together and support a particular integration scenario. The artifacts and objects bundled in a package relate to one of these categories: data integration, integration flow, OData service, or value mapping (see in the note box in [Chapter 2, Section 2.3.4](#)).

When dealing with the Integration Content Catalog, it's important to understand the different roles involved in consuming and publishing content. We generally distinguish the following roles:

- **Integration developer**

A member of the partner or customer organization responsible for consuming the prepackaged content available in the Integration Content Catalog.

- **Content publisher**

The person responsible for building and making the integration package available in the Integration Content Catalog.

- **Content reviewer**

The person responsible for reviewing and ensuring the quality of the content delivered in the Integration Content Catalog by the publisher.

Separating roles during the publication process helps improve the correctness of the content published in the catalog. It's important to stress that at the time of this book's release, only SAP can publish content packages. Consequently, the content publisher and reviewer roles aren't yet relevant to customers. This chapter will therefore primarily focus on the *integration developer*, which is the role that you, as a reader of this book, will most probably play.

Accessing the Integration Content Catalog

You can access the Integration Content Catalog in two different ways:

- Via a publicly accessible URL
- Via your own tenant

We'll now explore both of these approaches.

Via a Publicly Accessible URL

The publicly (and freely) available Integration Content Catalog is a web-based application that you can access at <https://api.sap.com/shell/integration>. An impression of the page is shown in [Figure 3.1](#).

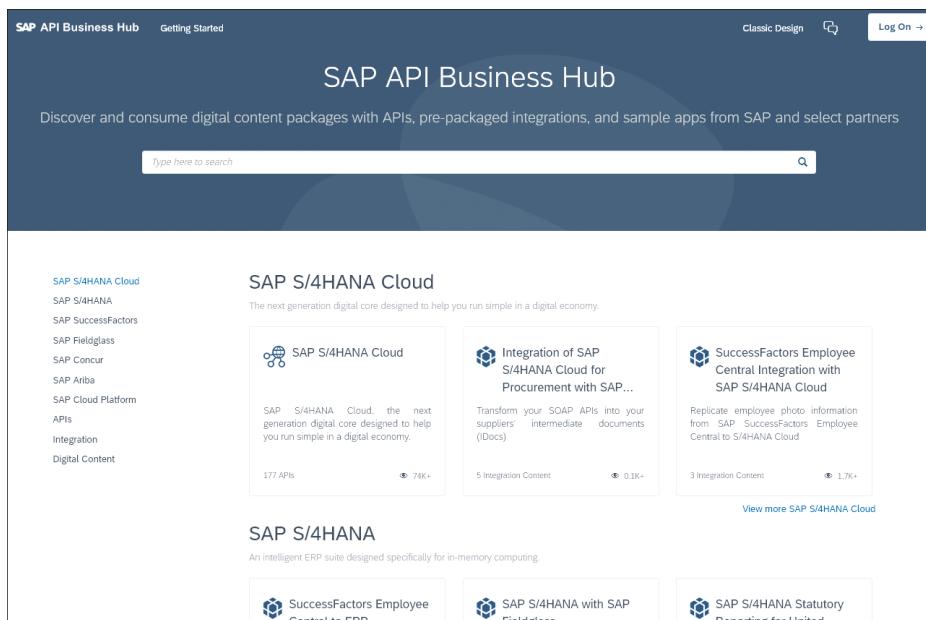


Figure 3.1 Publicly Accessible Integration Content Catalog

Note that the Integration Content Catalog is published on the SAP API Business Hub. You don't need an SAP Cloud Platform Integration tenant to use this web application. From this publicly available URL, only read access is available. If you need to reuse this content or have access to other features, you have to access a SAP Cloud Platform Integration tenant.

Via Your Own Tenant

You can access the Integration Content Catalog via the customer's tenant at `http://<server>:<port>/itspaces`.

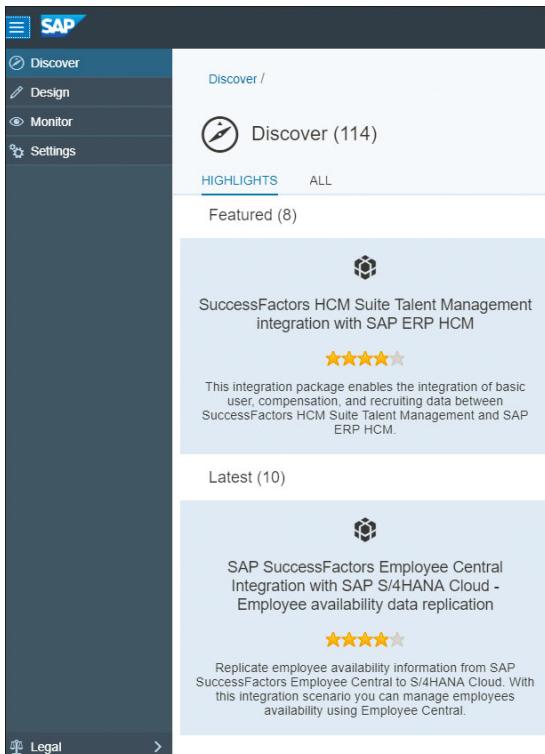


Figure 3.2 Landing Page of the Web UI

As shown on the left panel of [Figure 3.2](#), the SAP Cloud Platform Integration Web UI is made of the following four main sections:

- **Discover**
- **Design**
- **Monitor**
- **Settings**

Refer to [Chapter 2, Section 2.2.1](#), for more details about each section.

The **Discover** section is used to browse through the Integration Content Catalog. Let's dive deeper into the subject of consuming the available content, as shown in [Figure 3.2](#).

3.2 Terms and Conditions of Using Prepackaged Integration Content

Terms and conditions, within the context of SAP Cloud Platform Integration, refer to usage restrictions that affect prepackaged content. The publisher of the content can decide on what and how the content in their package can be used. The terms and conditions mainly affect three aspects (at the time of this writing):

- **Quick configure versus content edit**
Conditions that the publisher applies and that influence how the content can be consumed.
- **Notify about update (manual update)**
Related to updating the prepackaged content included in the Integration Content Catalog.
- **Automatic update**
Related to consumed prepackaged content automatically being updated.

We'll discuss these three aspects in the next sections.

3.2.1 Quick Configure versus Content Edit

One of the conditions that a publisher can apply to prepackaged content is the *quick configure versus content edit* condition. Quick configure or content edit conditions are used during the publication process to apply a usage restriction on the content. Let's explore these two conditions:

- **Quick configure (also called configure-only)**
The user of the package can only configure the options already made available by the package artifacts. Depending on the specifics of the concerned package, this usually only includes configuring the different adapters used in the integration flow. The configure-only option also means that the integration content itself (e.g., the steps in an integration flow) can't be changed and are therefore use-only. This

option might be seen as restrictive because its consumers must stick to the provided content. However, one of the advantages of this approach is that it's easier for the content's publisher to manage the versions of the content and be certain that the content is being used the way it was intended. For example, you would be allowed to adjust the adapter's specific settings, including connection parameters and username/password, but you wouldn't be allowed to change the adapter type from, for instance, Simple Object Access Protocol (SOAP) to Java Message Service (JMS). The impact of new versions is, therefore, controllable and predictable. If your integration content has quick configure terms and conditions, the configure-only approach of configuration will apply, which will be discussed in [Section 3.3.3](#).

■ Content edit

Consumers are free to modify the content as it suits them. This approach provides a lot of flexibility. However, with great power comes great responsibility. With content editing open to the user, it's possible for the integration developer to make changes to the content and fully deviate from the original intention of the content. For example, an integration developer might decide to add new steps to an integration flow or change the type of adapter used to communicate with the sender or receiver system. The changes brought to the content by its consumer might make future updates to the content more difficult. The resulting conflicts need to be manually resolved. At the time of this book's publishing, no automatic conflict resolution solution is available. Therefore, as the consumer, you must consider the impact of the changes you make to consumed content in relation to future updates of the content package. If your integration content uses the content edit terms and conditions, the content edit approach of configuration will apply, as will be discussed in [Section 3.3.3](#).

Note that these options are only available to SAP at this moment because only SAP may publish content to the Integration Content Catalog.

3.2.2 Notify about Update (Manual Update)

During the publishing process, the **Notify about Update**, checkbox is available. If this option is selected, you'll be notified of any updates made to the integration package. Notifications are sent automatically to consumers using the integration content. As shown in [Figure 3.3](#), a green **Update Available** link accompanies all updatable artifacts in the prepackaged content. In this way, the consumer is made aware that his content has an update available and can decide whether to perform the update.

Note

This type of notification is also called manual update. With manual update, the customer has the option to implement the update whenever it suits him. It's also possible to decide not to implement the changes. Not updating the content package doesn't create the danger that the deployed artifact will stop working. But the customer will obviously not benefit from the newly added features. Furthermore, it's also possible to revert back to an older version if necessary.



Figure 3.3 Notification of Updated Content in the Catalog

As a consumer, you can update your entire content package using the **Update package** button in the top-right corner of the screen ([Figure 3.4](#)). It's also possible to update single objects of the package instead of the entire package, by following these steps:

1. Select the content package whose object you would like to update.
2. Within the **Artifacts** tab of the package, select the artifacts that you want to update by checking the respective checkboxes. This tab lists all artifacts contained in the package and indicates which of them can be updated with the **Update Available** link, as shown in [Figure 3.4](#).
3. Click on the **Update package** button.

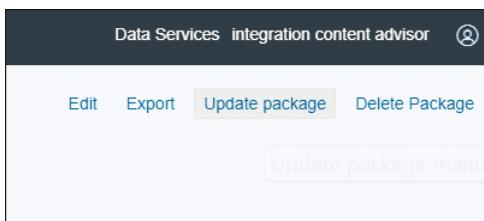


Figure 3.4 Updating Selected Items of the Content Package

Note that it's also possible to update a single integration flow by selecting the **Update** button in the **Actions** menu, as shown in [Figure 3.5](#).

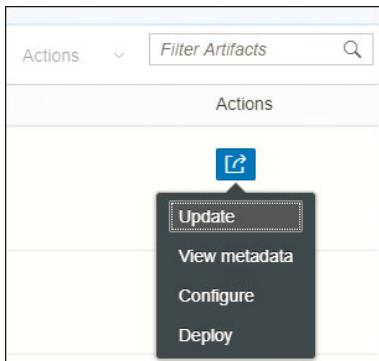


Figure 3.5 Updating a Particular Item of the Content Package

Note

In cases where the consumer modified integration content in the “modifiable” mode, a notification isn't sent when the content is updated by the publisher. The notification is only sent in the following cases:

- The consumed prepackaged content is in configure-only mode.
- The consumed prepackaged content is in modifiable mode but has only been configured and not modified. In other words, it has been used as if it was a configure-only package.

3.2.3 Automatic Update

As opposed to being notified of an update and being free to choose if or when to perform the actual update (as discussed in [Section 3.2.2](#)), there are cases when the publisher of the prepackaged content chooses automatic updating.

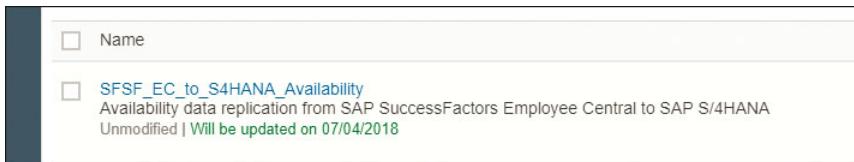
The main difference is that for manual update, the user explicitly has to perform the update operation, whereas no user interaction is required for automatic update. The updated content is automatically pushed to the tenant, and the deployed content is updated in one of two ways:

- **Immediate**

If it's an immediate update, the customers have a window of up to 12 hours before the update is automatically applied to the corresponding deployed artifacts.

- **Scheduled**

If it's a scheduled automatic update, the corresponding artifacts are marked with the date when the automatic update will be applied; for example, an artifact is marked with the message **Will be updated on . . .**, as shown in [Figure 3.6](#).



[Figure 3.6 Artifacts Marked by the Scheduled Automatic Update](#)

Note

Irrespective of whether the automatic update is immediate or scheduled, the customer has the option to apply the update before it's automatically done.

Furthermore, an automatic update will always be applied, leaving the consumer without control.

Up to now, we've learned the steps involved in finding and consuming any prepackaged content. Let's now explore some of the content packages available in the Integration Content Catalog at the time of this book's publishing.

3.3 Consuming Prepackaged Content

We touched on the role of an integration developer in [Section 3.1](#). In most cases, the integration developer is a member of the partner or customer organization and has the task of developing well-defined integration scenarios. [Figure 3.7](#) shows the steps involved in consuming prepackaged content from the catalog.

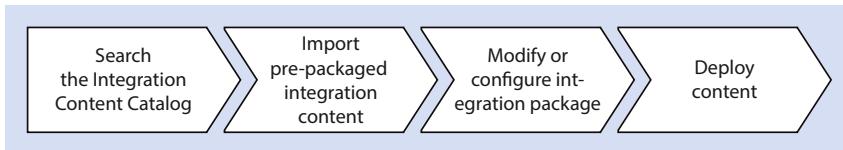


Figure 3.7 Process of Consuming Prepackaged Integration Content

We'll discuss these tasks in more detail in this section, with a hands-on, step-by-step guide.

3.3.1 Search in the Integration Content Catalog

It's a good practice to search through the Integration Content Catalog for existing content before attempting to develop your own content from scratch. To search the catalog, proceed as follows:

1. Navigate to your SAP Cloud Platform Integration tenant using the link provided by SAP on your browser. The link follows the format: `http://<server>:<port>/itspaces`.
2. Select the **Discover** menu item on the left side of the page to access the Integration Content Catalog, as shown in [Figure 3.8](#).

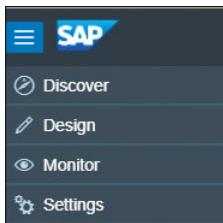


Figure 3.8 Accessing the Integration Content Catalog Main Page

3. In the Integration Content Catalog, you'll see a list of integration packages to choose from. By default, only the latest packages are listed. To view all existing packages, click on the **ALL** link on the top left of the page (see [Figure 3.9](#)). You then get a page similar to the one shown in [Figure 3.10](#).

The screenshot shows the SAP Cloud Platform Integration Discover landing page. At the top, it says "SAP Cloud Platform Integration". Below that, there's a breadcrumb navigation "Discover /". A search bar contains the text "Discover (114)". Underneath, there are two tabs: "HIGHLIGHTS" (which is underlined in blue) and "ALL". A section titled "Featured (8)" displays two cards:

- SuccessFactors HCM Suite Talent Management integration with SAP ERP HCM**: This card features a yellow star icon with four filled stars and one empty star. The description states: "This integration package enables the integration of basic user, compensation, and recruiting data between SuccessFactors HCM Suite Talent Management and SAP ERP HCM."
- SAP Hybris Marketing Cloud - Twitter Integration**: This card features a yellow star icon with four filled stars and one empty star. The description states: "This package provides you SAP Cloud integration content to load Twitter posts to your SAP Hybris Marketing Cloud system."

Below this, a section titled "Latest (10)" displays two more cards:

- SAP SuccessFactors Employee Central Integration with SAP S/4HANA Cloud - Employee availability data replication**: This card features a yellow star icon with four filled stars and one empty star. The description states: "Replicate employee availability information from SAP SuccessFactors Employee Central to S/4HANA Cloud. With this integration scenario you can manage employees availability using Employee Central."
- SAP Hybris Cloud for Customer for Utilities Solution - Integration with SAP ERP**: This card features a yellow star icon with four filled stars and one empty star. The description states: "The integration scope includes replication of technical, functional master data and transactional data online from IS-Utilities to execute utilities specific processes."

Figure 3.9 Discover Landing Page

4. A new screen appears with a variety of filtering categories. You have the option to filter using categories such as **Supported Platforms, Vendor, Countries, Industries, Line of Business, Products, and Keywords** ([Figure 3.10](#)). Note that you can also perform a keyword search on this screen.

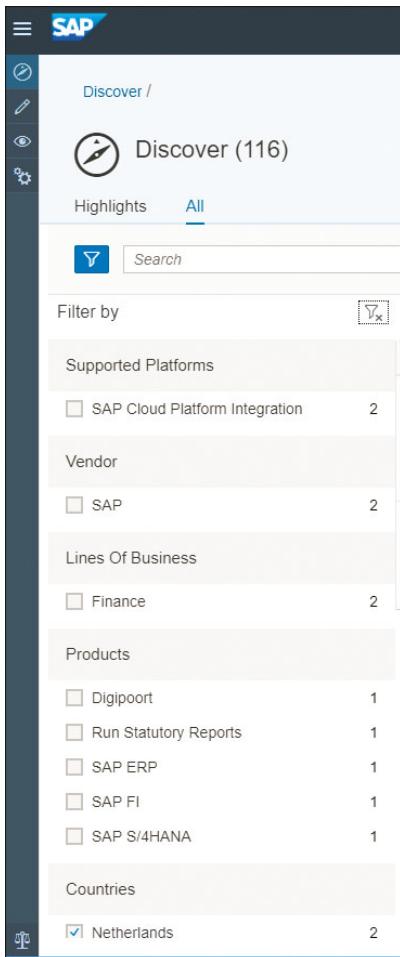


Figure 3.10 Filtering or Searching the Integration Content Catalog

The resulting list provides the name, high-level description of the package, published date, vendor, and version. Additionally, a user rating of the integration content is also available.

To view details of a particular package, select it from the list. As a result, you'll see a page similar to the one in [Figure 3.11](#).

The screenshot shows a SAP Cloud Platform Integration interface. At the top, there's a navigation bar with icons for search, refresh, and user profile, followed by the SAP logo and the text 'SAP Cloud Platform Integration'. Below the header, a breadcrumb trail reads 'Discover / NL Electronic VAT Return Statement and EC Sales List /'. The main content area features a title 'NL Electronic VAT Return Statement and EC Sales List' with a gear icon. A brief description follows: 'From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel (Digipoort). This integration package introduces submission of VAT return...'. To the right of the description are metadata fields: 'Vendor: SAP', 'Version: 1.0', 'Mode: Editable', and 'Published: 06 Dec 2016'. Below the description is a rating section with a yellow star icon and the text 'Average User Rating: 4.0 out of 5 (1 Rating)'. At the bottom of the main content area are tabs: 'Overview' (which is selected), 'Artifacts (4)', 'Documents (1)', 'Tags', and 'Ratings'. The 'Overview' tab contains detailed information: 'Description: From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel ([Digipoort](#)). This integration package introduces submission of VAT return statement and EC sales list via SAP HANA Cloud Integration platform. The package consists of two integration flows, *Aanleveringservice* and *Statusinformatieservice*, each of which communicates with a different web service (provided by [Digipoort](#)) in preproduction landscape and two more integration flows which are set to work in production landscape. Please note that necessary prerequisite for this solution to work is implementation of SAP note [1943145](#) in your ERP system.' Below this text are supported platform and category information: 'Supported Platform: SAP Cloud Platform Integration', 'Category: Integration', and 'Created: 25 Aug 2016'.

Figure 3.11 Viewing Details of an Integration Package

This Integration content detail page contains the following tabs:

- **Overview**

Contains the description of the package and scenarios it covers.

- **Artifacts**

Includes a list of integration flows, data integration flows, and other integration artifacts that make up the bundle.

- **Documents**

Includes guides and links to provide more documentation and information about the integration content to assist the user further. It's common that integration guides are included among the documents. An integration guide provides step-by-step guidelines on how to set up and configure the integration scenario. Note that the corresponding contacts or components are mentioned in the release notes of the content. This is useful if there are issues with the artifacts, and you want to report the issue to SAP.

- **Tags**

Provides different metadata to help classify content. The list of metadata includes industry, line of businesses, keywords, supported platforms, and so on.

■ Ratings

Contains details about consumer ratings as well as the logged-in user's own ratings.

Any of the items listed under the **Artifacts** tab, as shown in [Figure 3.12](#), can be clicked to view. For instance, you can click on an integration flow's name to display it (see [Figure 3.13](#)).

The screenshot shows the SAP Cloud Platform Integration interface. At the top, there are navigation icons (three horizontal lines, SAP logo, magnifying glass, pencil, eye, gear) and the title "SAP Cloud Platform Integration". To the right are links for "Data Services" and "Integration content advisor" along with a copy icon. Below the title, the path "Discover / NL Electronic VAT Return Statement and EC Sales List /" is shown, with a "Copy" button to its right. The main content area has a dark header with the package name "NL Electronic VAT Return Statement and EC Sales List". Below the header is a brief description: "From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel (Digipoort). This integration package introduces submission of VAT return...". To the right of the description are details: "Vendor: SAP", "Version: 1.0", "Mode: Editable", and "Published: 06 Dec 2016". Below this, there are tabs for "Overview", "Artifacts (4)" (which is selected), "Documents (1)", "Tags", and "Ratings". A search bar "Filter Artifacts" with a magnifying glass icon is located at the top right of the artifact list. The artifact list itself has columns for "Name", "Type", "Version", and "Actions". There are five entries:

Name	Type	Version	Actions
Preproduction AanleverService This integration flow provides connection to the Digipoort VAT statement delivery service in preproduction landscape.	Integration Flow	1.0.3	
Preproduction Statusinformatieservice This integration flow provides connection to the VAT statement status information service in preproduction landscape.	Integration Flow	1.0.3	
Production AanleverService This integration flow provides connection to the VAT statement delivery service in production landscape.	Integration Flow	1.0.3	
Production Statusinformatieservice This integration flow provides connection to the VAT statement status information service in production landscape.	Integration Flow	1.0.3	

Figure 3.12 List of an Integration Package's Artifacts

After browsing around and finding the integration package that fits your needs, you're now ready to further modify and configure it according to your requirements. Note that the possibility to configure or modify depends on the mode of the prepackaged content, as we'll discuss in [Section 3.3.3](#).

The next section will explore how to consume the content by copying it into your customer workspace.

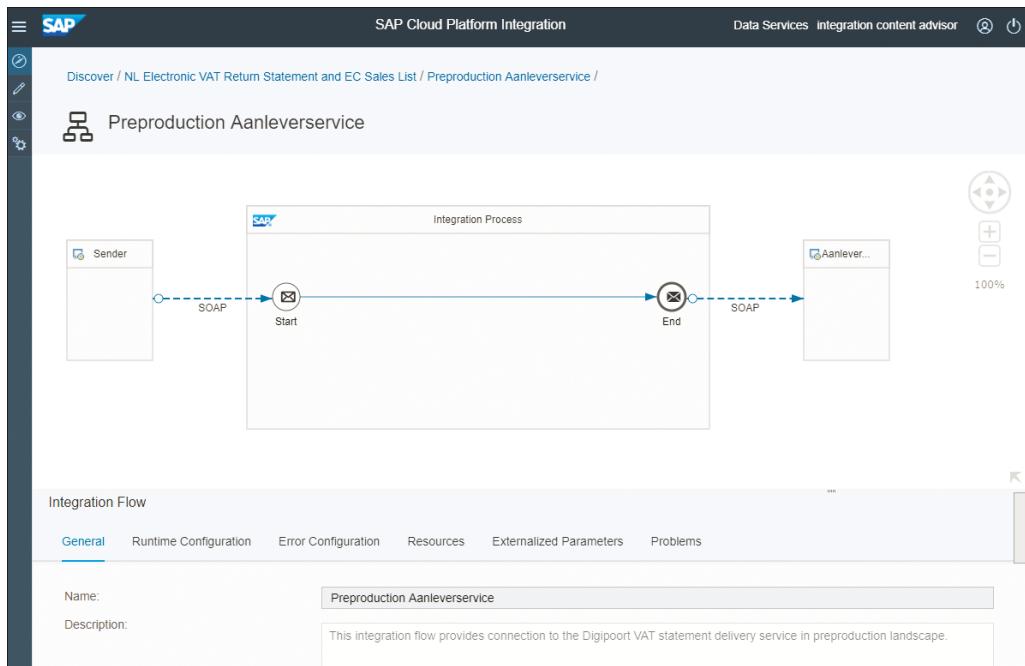


Figure 3.13 Details of the Integration Flow: Preproduction Delivery Service

3.3.2 Import Prepackaged Integration Content

You can copy the content available in the Integration Content Catalog into your own design workspace for further customer-specific configuration and enhancements. You can choose to use the template contained in the package as the basis upon which to make changes to suit your business requirements. To copy an integration package, perform the following steps:

1. After selecting the package that you want to copy (in our example, the **NL Electronic VAT Return Statement and EC Sales List** integration package), a **Copy** link appears in the top-right corner, as shown in [Figure 3.14](#). This **Copy** link enables you to copy the integration package to your own customer workspace.

3 Integration Content Catalog

The screenshot shows the SAP Cloud Platform Integration interface in the 'Discover' view. The package 'NL Electronic VAT Return Statement and EC Sales List' is selected. The top right corner has a 'Copy' button. Below the package name, there's a brief description: 'From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel (Digipoort). This integration package introduces submission of VAT return...'. To the right, there are details: Vendor: SAP, Version: 1.0, Mode: Editable, Published: 06 Dec 2016, and an average user rating of 4.0 out of 5 (1 Rating). Below this, a table lists four artifacts: 'Preproduction AanleverService', 'Preproduction Statusinformatieservice', 'Production AanleverService', and 'Production Statusinformatieservice', each with its type (Integration Flow), version (1.0.3), and a copy icon.

Figure 3.14 Copy Templates to Your Own Workspace

- After performing the copy action, the copied package and its artifacts are displayed in your own design workspace. Click on the **Design** section (refer to [Figure 3.8](#)) to further enhance the copied content. [Figure 3.15](#) shows that the copied package is now available in the customer's workspace, on the left.

The screenshot shows the SAP Cloud Platform Integration interface in the 'Design' view. On the left, there's a sidebar with icons for 'Design' and 'Playground'. The main area lists several packages: 'MQTT Webinar Demo 2016-07-25', 'NL Electronic VAT Return Statement and EC Sales List', 'PizzaSession', and 'Playground'. The 'NL Electronic VAT Return Statement and EC Sales List' package is visible, showing its details: Version 1.0, published on Sat, 24 Jun 2017 at 09:03:36 GMT, and a description about VAT return statements and EC sales lists in the Netherlands.

Figure 3.15 Design Component with Copied Templates

Note

As soon as you've copied a package from the **Discover** view to the **Design** view, a subscription is created in the background for each artifact contained in the package.

Having a subscription means that SAP Cloud Platform Integration knows that you're interested in any changes or planned updates on the concerned prepackaged integration content.

When anything changes on the package, you're informed via a tag next to the concerned artifacts. This subject is further discussed in Sections [Section 3.2.2](#) and [Section 3.2.3](#).

3.3.3 Modify or Configure the Integration Package

The content copied in the previous step is now ready for configuration. Such configuration steps might include adapter-specific endpoints. Depending on the customer-specific requirements, it's also possible to remodel and completely change the content. As stated previously, there are two approaches to configuring your integration package: content edit and configure-only. In the next sections, we'll explore each approach.

Content Edit Approach

This approach allows you to perform configuration steps to remodel and change the package content.

Notes

Being able to completely remodel and change the content of a copied package can be restricted by the terms and conditions of the integration package. When the package is restricted, use the configure-only approach, which we'll discuss further in the next section. The subject of terms and conditions was also discussed in [Section 3.2](#).

The integration package copied to your tenant can be modified and configured to your own needs by following these steps:

1. Click on the desired package's name on the list of packages presented in the screen (refer to [Figure 3.15](#)).
2. A new page similar to the one presented earlier in [Figure 3.12](#) loads. The page should display the full list of objects contained in the integration package. Note that the artifacts can be a mixture of integration flows, data integration, OData services, and value mappings. Furthermore, the **Documents** tab can contain files and URLs.

3. To display an integration flow, click on its name. You then get a detailed view of the flow as shown in [Figure 3.16](#).

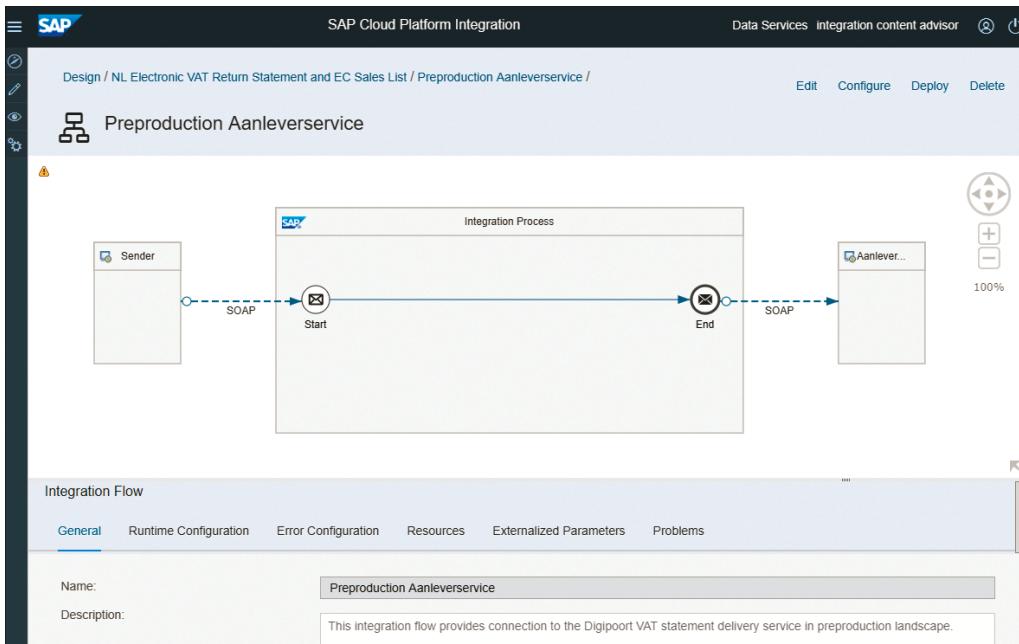


Figure 3.16 Details of the Integration Flow

4. To change an integration flow, click the **Edit** button in the upper-right corner of the screen. Note that, when in edit mode, the integration package editor locks the object and prevents any other user from changing it.
5. For most integration flows, connectivity details on the sender and receiver side need to be changed. [Figure 3.17](#) for instance, shows that the SOAP receiver connection details can be filled in by selecting the respective connector and specifying the properties in the **Connection** tab.
6. After you've made the desired changes, you can click on the **Save** button on the top-right corner of the page (see [Figure 3.17](#)). Alternatively, click **Save as version** to save a new version of the integration flow. You're then asked to provide a comment for the new version. Note that the version number is also automatically incremented.

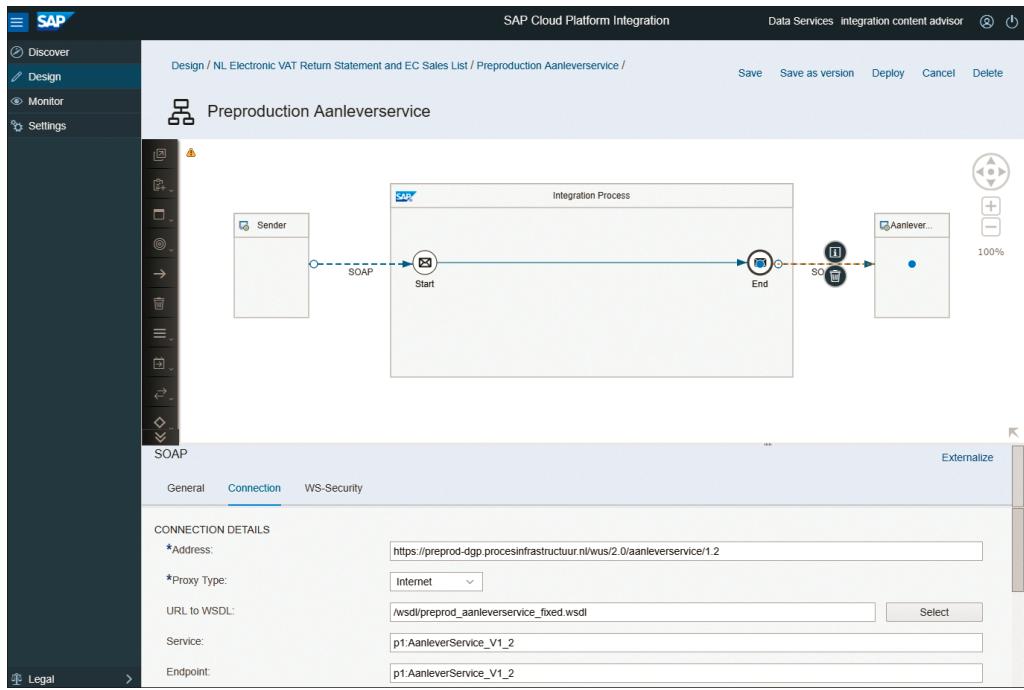


Figure 3.17 Adding Different Items to an Existing Integration Flow Template

- After saving your work in the previous step, you can now choose to deploy the integration flow on the tenant by using the **Deploy** option (see [Figure 3.17](#)). If you attempt to deploy without saving, a popup message will warn you that there are unsaved changes. You'll also be asked if you want to save and deploy at the same time.

The integration flow can then be used at runtime to process actual messages.

[Figure 3.18](#) shows that it's possible to download the content of an integration flow.

The content is downloaded to your local machine in a form of an archive file (e.g., ZIP file) containing the entire integration flow project. This integration flow's project ZIP file can then be imported into another package or deployed to run on a SAP Process Orchestration server as described in [Chapter 17](#) of *SAP Process Orchestration: The Comprehensive Guide* (SAP PRESS, 2017).

From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel (Digipoort). This integration package introduces submission of VAT return...

Vendor: SAP
Version: 1.0
Mode: Editable

Name	Type	Version	Actions
Preproduction AanleverService	Integration Flow	1.0.4	
Preproduction Statusinformatieservice	Integration Flow	1.0.4	
Production AanleverService	Integration Flow	1.0.5	
Production Statusinformatieservice	Integration Flow	1.0.4	

Figure 3.18 Overview of Updated Package

As shown in [Figure 3.18](#), every integration flow contains a version number. After consuming a particular version of the integration flow, you have the option to revert to an older version by performing the following steps:

1. In the **Artifacts** tab (refer to [Figure 3.18](#)), click on the **Version** number of the integration flow. Note that this works in design mode only.
2. The next screen displays the history of the different versions of the integration flow (see [Figure 3.19](#)).

Version history

Current version	1.0.4	Date Created: 6/24/17	Created By: S0011540061
	More ▾		
	1.0.3	Date Created: 6/24/17	Created By: S0011540061
		3bcde044a7134000bcd3f2c805526b7	
		Less ^	

Click To Revert

Figure 3.19 Reverting to a Different Version in the Version History

3. Hover over the version that you want to revert to, and click on the clock-like icon on the left side of the **Version history** screen, as shown in [Figure 3.19](#).
4. A new screen pops up, from which you need to confirm your action by clicking the **Ok** button.

After the consumed prepackaged content is configured to suit your needs, it's time to deploy it and make it available in your tenant's runtime. We'll discuss this in [Section 3.4](#).

Configure-Only Approach

The configure-only option provides an easy-to-use method of quickly adapting an integration flow to your requirements. It enables the user to perform only configuration activities, such as adding adapter-specific endpoints and assigning values to externalized parameters.

When you want to modify the content of the integration package, such as to add an extra step to the integration flow, you should use the content edit approach discussed earlier.

Following are the steps involved in using the configure-only approach:

1. Navigate to your design workspace by choosing the **Design** tab, as shown earlier in [Figure 3.8](#).
2. Click on the package's name as shown earlier in [Figure 3.15](#). The next screen displays a list of artifacts contained in the **Artifacts** tab.
3. Select the **Action** button on the row corresponding to the integration flow that you want to configure (see [Figure 3.20](#)), and then choose **Configure** from the dropdown menu.

If the integration flow doesn't have configurable attributes, a warning pops up, as shown in [Figure 3.21](#).

The screenshot shows the SAP Cloud Platform Integration interface. At the top, it displays 'SAP Cloud Platform Integration' and 'Data Services integration content advisor'. Below the header, there's a breadcrumb trail 'Design / NL Electronic VAT Return Statement and EC Sales List /'. On the right, there are 'Edit', 'Export', and 'Delete Package' buttons. The main content area is titled 'NL Electronic VAT Return Statement and EC Sales List'. It includes a brief description: 'From 2014 onwards VAT return statement (BTW aangifte) and EC sales list (ICP aangifte) in Netherlands can be reported to the Tax Office electronically via the SBR channel (Digipoort). This integration package introduces submission of VAT return...'. To the right of the description are details: 'Vendor: SAP', 'Version: 1.0', and 'Mode: Editable'. Below the description, there are tabs for 'Overview', 'Artifacts (4)', 'Documents (1)', 'Tags', and 'Comments'. The 'Artifacts (4)' tab is selected. A search bar 'Filter Artifacts' is present. The list of artifacts includes:

- Name: Type: Integration Flow, Version: 1.0.4, Actions: View metadata, Download, Configure, Deploy.
- Preproduction Aanlevering: This integration flow provides connection to the Digipoort VAT statement delivery service in preproduction landscape. Modified.
- Preproduction Statusinformatieservice: This integration flow provides connection to the VAT statement status information service in preproduction landscape. Modified.
- Production Aanlevering: This integration flow provides connection to the VAT statement delivery service in production landscape. Modified.
- Production Statusinformatieservice: This integration flow provides connection to the VAT statement status information service in production landscape. Modified.

Figure 3.20 Accessing the Configure-Only Option

This screenshot shows a configuration dialog box. The title bar says 'Information'. The message inside the box is: 'No attributes available for configuration in the selected artifact.' In the bottom right corner of the dialog box is a button labeled 'OK'.

Figure 3.21 Warning for Missing Attributes in the Configure-Only Approach

- If the integration flow does have configurable attributes, configure the details of each tab shown in [Figure 3.22](#). Notice that in this screen, the **Time**, **Receiver**, and **More** tabs are displayed, but there are even more tabs available. A complete list of possible configuration tabs and their explanations are available in [Table 3.1](#).

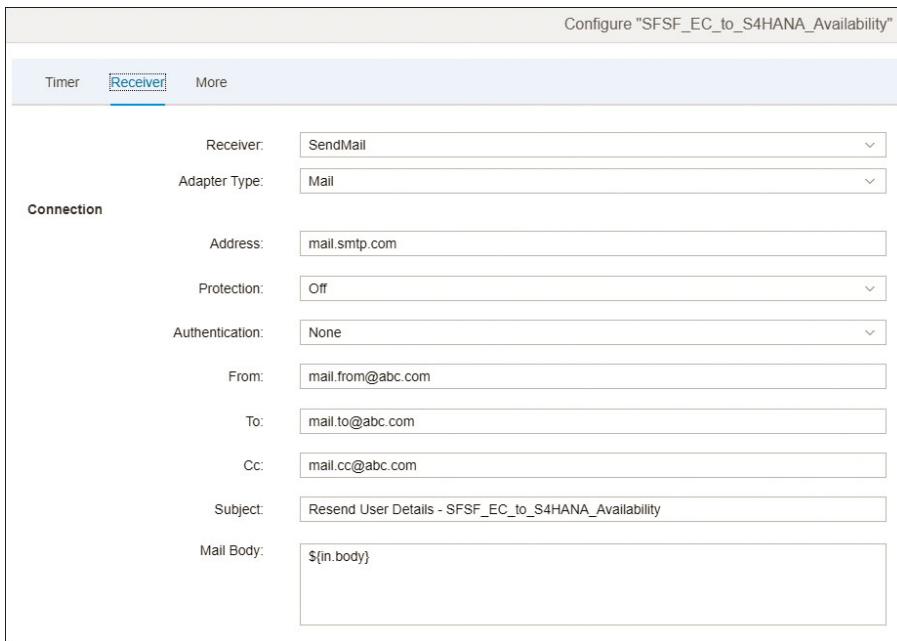


Figure 3.22 Configure the Receiver Connection Details of an Integration Flow

Configuration Tab	Description
Timer	If the integration flow uses a scheduler (a timer start event), its settings can be configured here. Possible options include Run once , Schedule on Day , or Schedule to Recur . We'll cover the topic of using a timer start event in an integration flow in Chapter 6, Section 6.1.2 .
Sender	Configure the connectivity details of the sender adapter.
Receiver	Configure the connectivity details of the receiver adapter. See the example in Figure 3.22 .
More	Provide a configuration feature for externalized parameters. Note that at the time of this book's publishing, all string fields can be externalized in all flow steps. Externalized parameters allow you to define variables and use them in an integration flow. The values of these variables can be assigned later in the configuration process. Parameters and externalization are further discussed in Chapter 4, Section 4.2 .

Table 3.1 Available Configuration Tabs for the Configure-Only Approach

5. After you've made the desired changes in the previous step, click on the **Save** button, as shown earlier in [Figure 3.22](#).
6. Deploy the integration flow on the tenant by using the **Deploy** option (refer to [Figure 3.22](#).)

Note

Note that the four tabs listed in [Table 3.1](#) are available after you've clicked on the **Configure** option. However, the tabs are only populated with configurable properties under the following conditions:

- The presence of a sender adapter in the integration flow (**Sender** tab)
- The presence of a receiver adapter in the integration flow (**Receiver** tab)
- The presence of a timer start event (**Timer** tab)
- When you've externalized any string fields in any flow step (**More** tab)

For instance, when you have an integration flow beginning with a timer start event (i.e., no sender system is involved), the **Sender** tab won't be populated. If nothing has been externalized in the entire flow, you'll receive an error message stating, **No attributes available for quick configuration in the selected artifact.**

We'll explore the subject of externalizing parameters further in [Chapter 4](#), Section 2.

3.3.4 Deploy Content

Your integration flow is now configured and ready to be deployed. You can deploy the integration flow by following these steps:

1. From within your customer workspace (**Design** tab), select the integration flow that you configured in [Section 3.3.3](#).
2. Click on the **Deploy** button on the top right of [Figure 3.23](#).

The deployed integration flow is now ready to reliably connect systems with each other through message exchange. Congratulations, you just learned to consume pre-packaged integration content! Imagine how much time it would have taken to figure out the mapping requirements and build this entire integration flow from scratch.

Now it's time to explore the terms and conditions that might affect and restrict the way you consume prepackaged content.

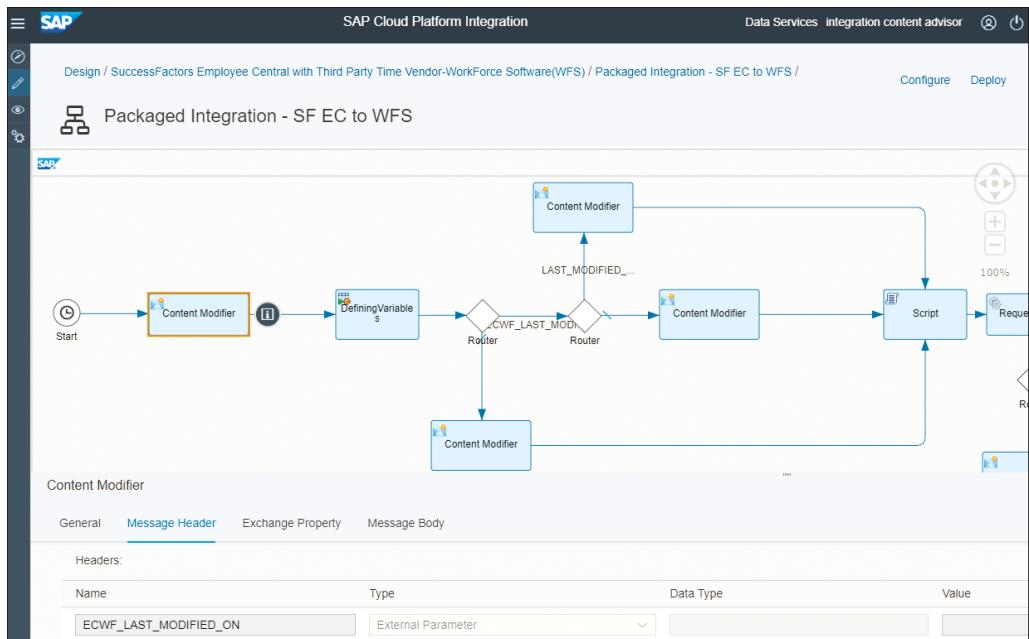


Figure 3.23 Deploying an Integration Flow

3.4 Prepackaged Content Provided by SAP

Let's showcase the Integration Content Catalog's content, which includes integration scenarios of SAP's most commonly used cloud-based applications:

- SAP SuccessFactors
- SAP Cloud for Customer
- SAP Ariba
- SAP C/4HANA
- Content for globalization scenarios

For each one of these categories, we'll explore their content and specify the use case under which the provided content can be leveraged.

3.4.1 Content for SAP SuccessFactors

SAP SuccessFactors is a cloud-based human capital management (HCM) solution that integrates onboarding, social business and collaboration tools, a learning management system (LMS), recruiting software, performance management, succession planning, applicant tracking software, talent management, and HR analytics to deliver business strategy alignment, team execution, and maximum people performance. SAP has another HR product called SAP ERP Human Capital Management (SAP ERP HCM), which provides an integrated set of modules to help an organization manage its people. It's effectively an on-premise HCM product.

Some customers have opted to use a hybrid approach, where SAP ERP HCM (on premise) and SAP SuccessFactors (cloud) work in tandem. In these cases, customers have to decide which part of an end-to-end HCM process runs on which system. One popular approach is to use SAP ERP HCM for core HR processes and use SAP SuccessFactors for one or more talent management processes. It's understood that with such a division of responsibilities between these two HCM systems, integration plays a critical part in linking and synchronizing them.

For illustration purposes, [Figure 3.24](#) depicts a common onboarding process between SAP ERP HCM and SAP SuccessFactors. This process shows a requirement to export an employee's prehire data (information about candidates before they become employees) from SAP ERP HCM to SAP SuccessFactors Onboarding. Moreover, when the process is completed in SAP SuccessFactors Onboarding, you can export the employee data from SAP SuccessFactors Onboarding back to SAP ERP HCM and create employee master data. The complete use case is presented in [Figure 3.24](#).

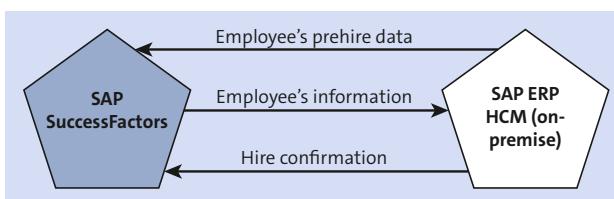
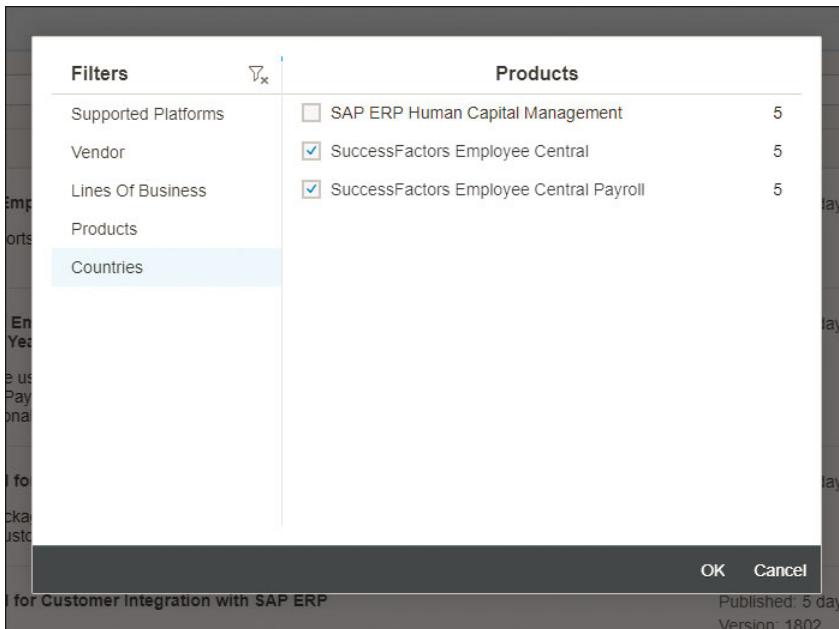


Figure 3.24 Use Case to Integrate SAP SuccessFactors Onboarding and SAP ERP HCM

By integrating and combining the functionalities of these two platforms, the customer can achieve a better end-to-end process result. The Integration Content Catalog provides a variety of packages to cover different integration scenarios, including for the use case presented in [Figure 3.24](#). Refer to the Integration Content Catalog for the most up-to-date list of SAP SuccessFactors-related packages.

To discover all packages that relate to SAP SuccessFactors, you need to apply a filter on the main page of the Integration Content Catalog by selecting any entry with the word **SuccessFactors** as the value of the **Products** dropdown list. This filtering exercise is depicted in [Figure 3.25](#).



[Figure 3.25](#) Filtering SAP SuccessFactors Packages

Most packages are self-explanatory based on their names. In addition, you can further explore and consume their content as already discussed in [Section 3.2](#).

SAP SuccessFactors Adapter

Most of the integration flows included in this package make use of the SAP SuccessFactors adapter for SAP Cloud Platform Integration. This is a special adapter that has been developed to connect solely to SAP SuccessFactors applications. Refer to [Chapter 1, Section 1.3.3](#), to read more about the SAP Cloud Platform Integration connectivity options.

3.4.2 Content for SAP Cloud for Customer

SAP Cloud for Customer is SAP's cloud customer relationship management (CRM) solution, which brings marketing, sales, commerce, and customer service together.

As a cloud-based CRM system, SAP Cloud for Customer needs to interact with a number of other systems to ensure that information such as accounts, materials, price conditions, and other master data are in sync.

Currently, the Integration Content Catalog provides three main content packages related to SAP Cloud for Customer. These packages include content that supports the following use cases:

- SAP Cloud for Customer integration with SAP ERP
- SAP Cloud for Customer integration with SAP CRM
- SAP Cloud for Customer integration with SAP Marketing Cloud

To further illustrate how the integration packages for SAP Cloud for Customer can be used, let's dig a bit deeper into the use case of integrating SAP Cloud for Customer with SAP ERP.

SAP Cloud for Customer Integration with SAP ERP

As stated in the previous section, SAP Cloud for Customer needs to exchange master and transactional data with SAP ERP. In terms of master data, in most use cases, SAP ERP acts as the master system. This means that master data is synchronized one-way, from SAP ERP to SAP Cloud for Customer. In addition, transactional data, such as opportunity, pricing, and quotes, is also exchanged between these platforms. An overview of the data exchanged between these two systems is depicted in [Figure 3.26](#).

Through the Integration Content Catalog, SAP provides the needed integration flows to synchronize your on-premise SAP ERP and SAP Cloud for Customer systems. The integration flows cover the scope of business objects presented in [Figure 3.26](#).

SOAP Adapter

From a technical perspective, the SOAP adapter is used to integrate SAP Cloud Platform Integration and SAP Cloud for Customer. The consumer of the integration package needs to perform configuration tasks in the **Adapter Specific** tab of the concerned integration flow, as explained in [Section 3.3.3](#). Refer to [Chapter 1, Section 1.3.3](#), to read more about SAP Cloud Platform Integration connectivity options.

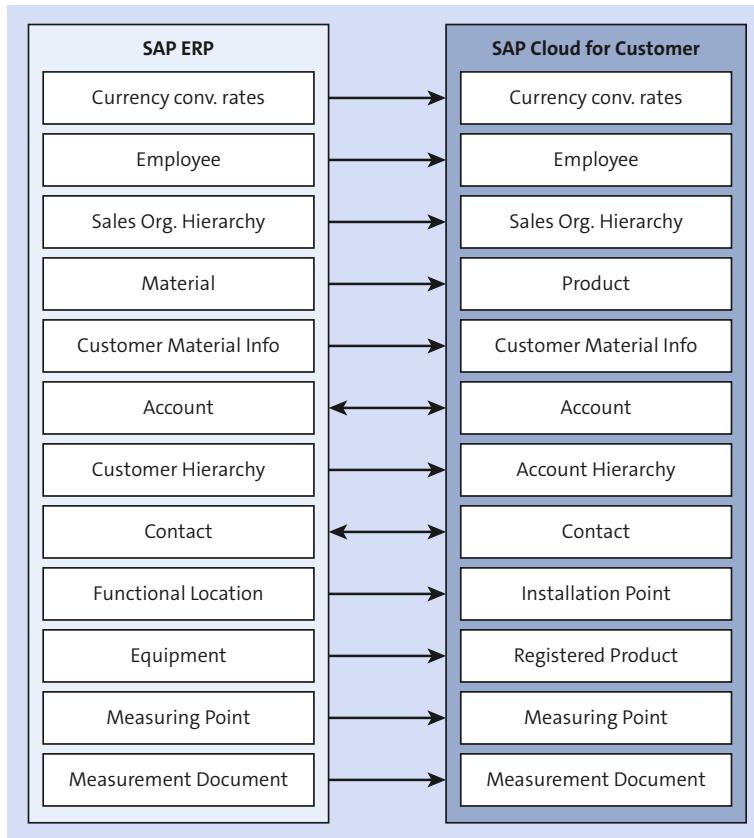


Figure 3.26 Master Data Synchronization between SAP Cloud for Customer and SAP ERP

To discover SAP Cloud for Customer-related integration packages, you need to apply a filter on the main page of the Integration Content Catalog by selecting the **SAP Hybris Cloud for Customer** entry from the **Product** dropdown list.

3.4.3 Content for Integrating with SAP C/4HANA

SAP C/4HANA is a family of cloud-based solutions that includes the following different products:

- SAP Commerce Cloud
- SAP Customer Data Cloud

- SAP Sales Cloud
- SAP Service Cloud
- SAP Marketing Cloud

In the preceding list of products, SAP Sales Cloud and SAP Service Cloud are included under the code name SAP Cloud for Customer. The integration content for SAP Cloud for Customer was already covered in [Section 3.4.2](#). In the coming sections, we'll explore the content provided for SAP Commerce Cloud, SAP Marketing Cloud, SAP Subscription Billing, and SAP Billing and Revenue Innovation Management (formerly SAP Hybris Revenue and SAP Hybris Billing).

Note

For the most updated information of the current integration packages for SAP Commerce Cloud, SAP Subscription Billing or SAP Billing and Revenue Innovation Management, and SAP Marketing Cloud, refer to the **Documents** tab within each package. It generally contains integration guides and various informative documents. You can also refer to the SAP Community for more information. Please note that while the billing and revenue functionality was part of the old SAP Hybris suite, it is not part of the new SAP C/4HANA suite.

SAP Commerce Cloud

The integration content for SAP Commerce Cloud/SAP Cloud for Customer provides the possibility to do the following:

- Synchronize customer data from SAP Commerce Cloud to SAP Cloud for Customer.
- Synchronize customer service ticket data between the two systems to more efficiently connect customers with service or sales agents.

After SAP Commerce Cloud and SAP Cloud for Customer are connected, agents can provide service through the ticket or directly over the phone, all while accessing the same singular storefront as the customer, through the Assisted Service Module for SAP Commerce Cloud for exceptional service and sales assistance on the spot.

At the time of publishing this book, the integration package “SAP Customer Engagement Center Integration with SAP Commerce” is an example package used to synchronize data between SAP Commerce Cloud and SAP Customer Engagement Center integration. To illustrate common use cases, here are the business objects exchanged:

- Customer address replication from SAP Commerce Cloud to SAP Customer Engagement Center
- Customer replication from SAP Commerce Cloud to SAP Customer Engagement Center
- Basic sales order details replication from SAP Commerce Cloud to SAP Customer Engagement Center for indexing

SAP Subscription Billing and SAP Billing and Revenue Innovation Management

At the time of publishing this book, the integration package “Integration with SAP Subscription Billing” is an example package used to provide the capabilities to process bills originating from the SAP Subscription Billing in SAP S/4HANA for billing and revenue innovation management. Common use cases are listed here:

- Extract billing documents from SAP Subscription Billing to SAP S/4HANA.
- Send back Customer IDs from SAP S/4HANA to SAP Subscription Billing when customers are created or updated.
- Extract customers from SAP Subscription Billing and then replicate to SAP S/4HANA.
- Replicate customers from SAP S/4HANA to SAP Subscription Billing.

SAP Marketing Cloud

At the time of publishing this book, the available integration content covers many integration scenarios for SAP Marketing Cloud:

- **Master data and basic replication**
The accounts, contacts, individual customers, leads activities, and opportunities are replicated from SAP Cloud for Customer to SAP Marketing Cloud.
- **Call center scenario**
This scenario creates a call center campaign in SAP Marketing Cloud and executes the campaign in SAP Cloud for Customer.
- **Lead management scenario**
Any lead that is converted into an opportunity further creates an opportunity interaction in SAP Marketing Cloud. These leads and opportunities in SAP Cloud for Customer also contain information about the corresponding campaigns from SAP Marketing Cloud.

Furthermore, several integration packages cover a number of use cases with SAP Marketing:

- SAP S/4HANA Enterprise Management on-premise – SAP Marketing Cloud integration
- SAP Marketing Cloud – SAP Customer Relationship Management (SAP CRM) integration
- SAP Cloud for Customer – SAP Marketing integration
- SAP Marketing Cloud – SAP Customer Attribution integration
- SAP Marketing Cloud – SAP ERP order and business partner integration
- SAP Marketing Cloud – Twitter integration
- SAP Marketing Cloud – Content Management System integration
- SAP Marketing – Google AdWords Paid Search integration
- SAP Marketing Cloud – Twitter integration admin
- SAP Marketing Cloud – Facebook integration admin
- SAP Marketing Cloud – Facebook integration
- SAP Marketing – Google Analytics integration

3.4.4 Content for Integrating with the Ariba Network

The Ariba Network is a cloud-based procurement solution that allows you to locate new suppliers, streamline transaction processes, and save costs. It's one of the largest trading partner communities that provides connectivity and online services to organizations engaged in business-to-business e-commerce.

With the Ariba Network, buyers and suppliers can do business with each other over the Internet and access each other's services. It contains additional functionality such as directory services, reporting tools, supplier tools, payment, and sourcing.

To illustrate a common use case, customers can create purchase orders, goods receipts, invoices, and so on, from the Ariba Network and have them synchronized back to their own (on-premise) SAP Business Suite operational purchasing or supplier-side processes. An overview of these common use cases between SAP ERP Materials Management (SAP MM) and SAP Supplier Relationship Management (SAP SRM) is depicted in [Figure 3.27](#).

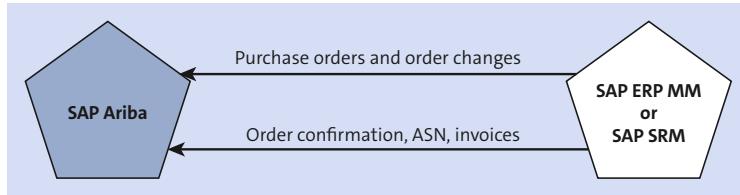


Figure 3.27 Integration Use Cases between SAP Ariba and SAP ERP or SAP SRM

Currently, one package is available to cover the need for integrating the Ariba Network with your existing SAP ERP using SAP Cloud Platform Integration. This content package integrates and automates your SAP Business Suite operational purchasing processes or supplier-side processes with the Ariba Network.

For buyers, the scope of this mediated connection based on the Ariba Network integration for SAP Business Suite Add-On 1.0 includes the following:

- Purchase order and invoice automation for MM and SAP SRM classic
- Discount management integration (optional)

The scope supports selected aspects of the following procure-to-pay end-to-end business scenarios:

- Self-service and indirect procurement
- Direct procurement
- Service procurement
- Invoice management
- Collaborative Supply Chain 1.0 with schedule agreement release order processing

For suppliers, the scope of this mediated connection, based on the Ariba Network integration with SAP Business Suite Add-On 1.0, includes the following:

- Sales order and billing integration with the Ariba Network for SAP ERP Sales and Distribution (SAP SD)

Notes

From a technical perspective, the SOAP adapter is used to integrate SAP Cloud Platform Integration with SAP Ariba. The consumer of the integration package needs to perform configuration tasks in the **Adapter Specific** tab of the concerned integration flow, as already explained in [Section 3.3.3](#). Please refer to [Chapter 1](#), [Section 1.3.3](#), to read more about the SAP Cloud Platform Integration connectivity options.

To discover SAP Ariba-related integration packages, you need to apply a filter on the main page of the Integration Content Catalog by selecting the **Ariba Network with SAP Business Suite** entry from the **Product** dropdown list.

3.4.5 Content for Globalization Scenarios

Recently, many government agencies have been moving from traditional paper-based documents to digital or electronic documents. As such, businesses dealing with these government agencies (e.g., the tax office) must quickly adapt their internal processes to comply.

For multinational companies having, for instance, to report their taxes in many countries, this can be a challenge because documents need to be sent to the tax offices of different countries with different formats and standards from the same source application (SAP ERP system). Another aspect to consider is that, in most cases, the transmission of these government-related documents requires a high level of security.

The mapping of the source application's messages to the required government formats and the transmission thereof is a classic integration scenario. The need to transfer electronic documents to government agencies around the world is a use case SAP Cloud Platform Integration covers with its integration content for globalization. This content is also known as the eDocument Framework.

The eDocument Framework provides a generic approach to create, process, and manage electronic documents. The framework supports handling country-specific requirements for electronic documents with regard to the format of messages, security requirements, and processing steps in end-to-end integration scenarios. The source data for the documents may originate in any application that implements the web services available in the integration flows. At the time of this book's publication, a number of integration packages are provided in SAP Cloud Platform Integration to support eDocument for Chile, Italy, Spain, Peru, The Netherlands, Great Britain, Colombia, Germany, India, Mexico, United States, and Hungary.

With these packages, customers don't need to spend time trying to figure out how to perform mappings or find out which security levels are required to comply with these government electronic document formats. The customer can simply reuse the provided integration content and reduce implementation time and costs.

To discover a particular eDocument package for a specific country, you need to apply a filter on the main page of the Integration Content Catalog by selecting a country (e.g., **Netherlands**) from the **Country** dropdown list.

In this section, you learned about the prepackaged content delivered by SAP in the Integration Content Catalog. In the next section, we'll cover how to create your own content.

3.5 Creating Your Own Content Package

There might be cases when existing prepackaged content doesn't meet your business needs, and you would like to create a package that can be used and reused specifically by team members within your organization. In such a case, you may consider creating your own content package.

The first step of publishing your own content is to create the desired integration content. The process of creating your own content has been discussed when creating your first integration flow in [Chapter 2](#). Furthermore, [Chapter 4](#) will walk you through the process of creating different integration content artifacts. Therefore, we won't go into the development details here.

To create an integration package, follow these steps:

1. Open the **Design** menu item (refer to [Figure 3.8](#)).
2. Select the **Create** option to create an integration package, (see [Figure 3.28](#)).

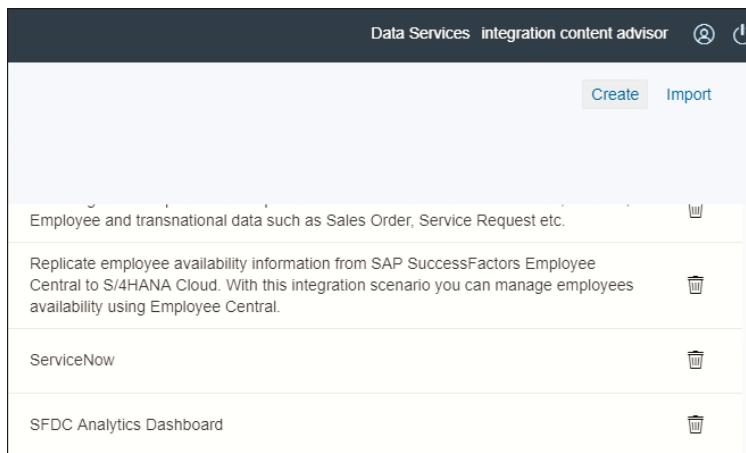


Figure 3.28 Creating an Integration Package

3. On the next screen, specify the name of the new integration package. Additional metadata details, such as the version, owner, mode, description, tags, products, industries, line of businesses, country, and keywords, can be provided. Some of this metadata is shown in [Figure 3.29](#). You can maintain of these metadata details in tabs such as **Overview** and **Tags**. Bear in mind that this metadata is important to better classify content and allow consumers to find it easily. It's therefore important to provide as much detail as possible.

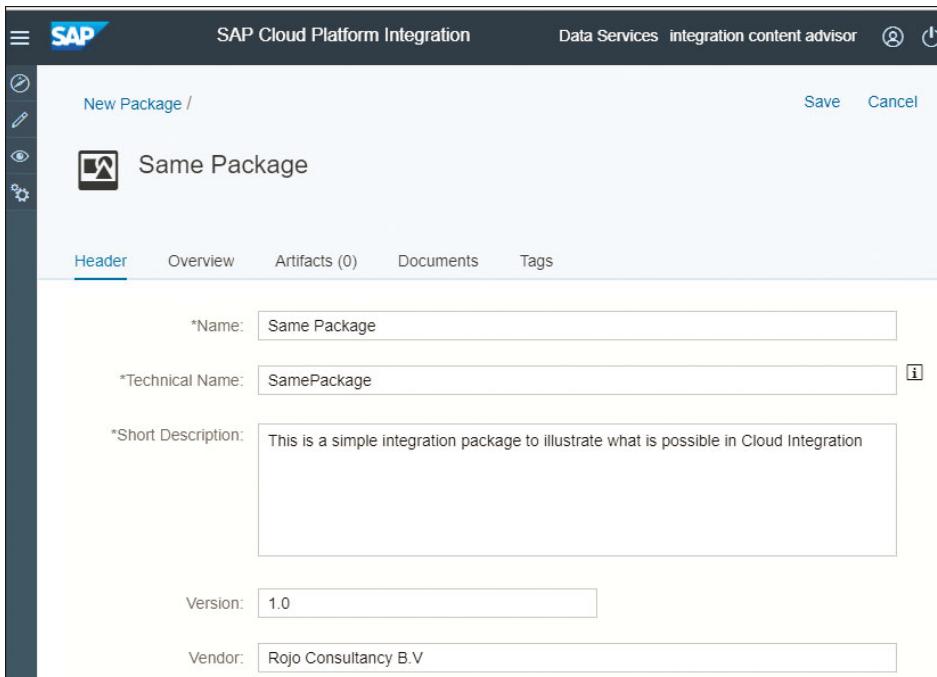


Figure 3.29 Specifying Details of the New Integration Package

4. Save the package using the **Save** button, as shown in [Figure 3.29](#).
5. On the **Artifacts** tab, click **Add**, and choose an artifact of type **Data Integration**, **Process Integration**, **OData Service**, or **Value Mapping** to add it to the integration package ([Figure 3.30](#)).
6. Artifacts can be added to the newly created package by creating them from scratch if they don't already exist. Alternatively, the artifacts can also be imported if they already exist. These two options are made possible using the **Create** or **Upload** radio buttons, as shown in [Figure 3.31](#).

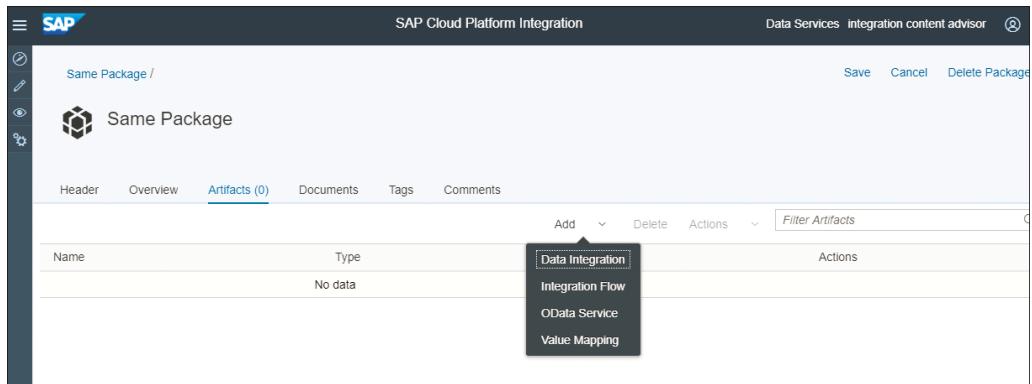


Figure 3.30 Adding Artifacts to an Integration Package

This is a dialog box titled 'Add integration flow artifact'. It has two radio button options: 'Create' (selected) and 'Upload'. Below these are fields for 'Name' (containing '<Name>') and 'ID'. A dropdown menu for 'Product Profile' is set to 'SAP Cloud Platform Integration'. There is a large text area for 'Description' with placeholder text '<Description>'. At the bottom, there are fields for 'Sender' (<Sender>) and 'Receiver' (<Receiver>). At the very bottom are 'OK' and 'Cancel' buttons.

Figure 3.31 Creating or Importing a Process Integration Artifact

7. To provide documentation for the integration package, click on **Add** in the **Documents** tab, as shown in [Figure 3.32](#). Documents of type **File** or **URL** can be added to this tab.

After packaging and saving the integration content, it becomes available in your own design space and can therefore be used by team members of your organization.

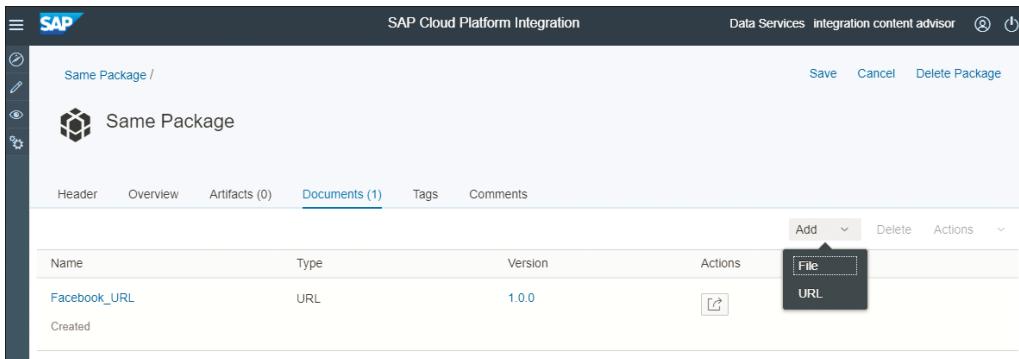


Figure 3.32 Adding a File or URL to a Content Package

Notes

A content package is automatically locked against modification from other users when someone is in the process of editing it. The package lock is only released for modification after it has been saved. If the session times out, or the browser closes while you're still working on your integration package, it remains locked until it's saved, canceled, or deleted by the lock owner.

Currently, only SAP is allowed to publish data to the Integration Content Catalog, so packages created by customers or partners are for their use only and can't be directly leveraged by other customers. Different SAP Partners have listed their prepackaged integration content on the Integration Content Catalog, but you'll need to contact the SAP Partners to purchase and obtain an archive of the package.

3.6 Summary

This chapter introduced you to the capabilities and features of the Integration Content Catalog in SAP Cloud Platform Integration. A systematic guide was used to demonstrate how to consume the contents of the catalog. You've also learned about the terms and conditions that can affect and restrict the way prepackaged content is consumed.

The chapter then explored the prepackaged content that SAP delivers in the Integration Content Catalog to speed up implementation time and save costs related to performing integration with the most-used SAP cloud-based applications. The Integration Content Catalog is expected to grow and see new packages added to it on a regular basis. It's therefore recommended that you first check the catalog before developing your integration scenarios from scratch.

In the next chapter, we'll further explore how SAP Cloud Platform Integration empowers developers to create their own basic integrations when prepackaged content won't suffice.

Chapter 4

Basic Integration Scenarios

If prepackaged integration content won't suffice, developers can use SAP Cloud Platform Integration to create their own integrations. This chapter explores these capabilities by taking a closer look at concrete integration challenges and how to solve them.

In the previous chapter, you learned how to configure prepackaged integration content. However, in many cases, you have individual integration needs that simply can't be fulfilled by an already existing package or that require too much effort to adjust a delivered scenario to the required functionality. Fortunately, SAP Cloud Platform Integration comes with a web-based development environment that allows you to model unique scenarios from scratch. We'll explore the options you have at your disposal in this chapter, followed by three more chapters drilling into topics that are more specialized. In this chapter, we'll take a close look at SAP Cloud Platform Integration's data model, how content enrichment works by invoking an OData service, and how to add mappings to an integration flow.

4.1 Working with SAP Cloud Platform Integration's Data Model

Before we dive into the details of a concrete modeling exercise, let's briefly discuss some terms and fundamental concepts of SAP Cloud Platform Integration's development and runtime environment. [Figure 4.1](#) depicts a basic integration flow in SAP Cloud Platform Integration's web environment, showing the transfer of a message from a sender to a receiver via SAP Cloud Platform Integration.

The overall representation of integration flows is based on Business Process Model and Notation (BPMN), a widely adopted Object Management Group (OMG) standard for graphically depicting processes. Although the roots of BPMN are in the business process domain (as the name indicates), it's expressive enough to be useful for integration processes, as well. Hence, SAP decided to rely on BPMN as the lingua franca for all kinds of processes. BPMN is broadly used in SAP systems—for example, in SAP

Solution Manager, SAP Process Integration, and SAP Business Process Management (SAP BPM), to name a few.

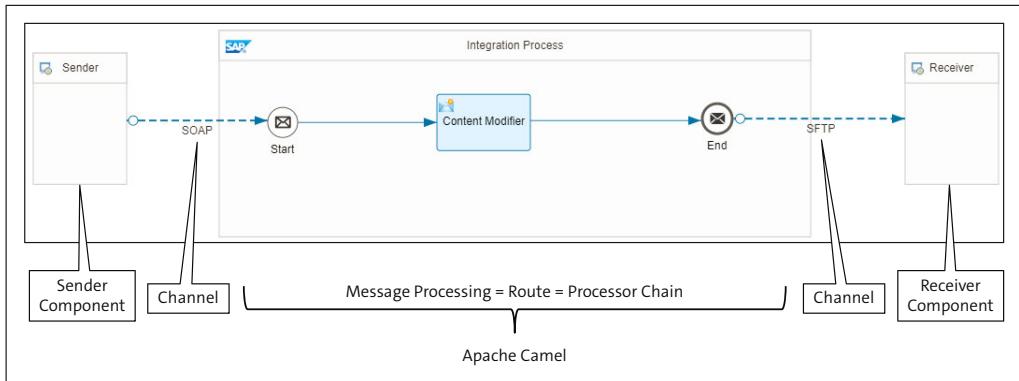


Figure 4.1 A Basic Integration Flow

On the left side of [Figure 4.1](#), you can identify the sender component representing the sender of a message. The receiver is depicted on the far right. Both sender and receiver are represented by pools (the rectangles headlined with **Sender** and **Receiver**, respectively). A *pool* is a term taken from the BPMN standard and typically stands for a participant in collaborative scenarios in which several processes work together to fulfill a certain goal. A third pool, entitled **Integration Process**, shown in the middle of [Figure 4.1](#), represents the SAP Cloud Platform Integration server where message processing actually happens.

The **Sender/Receiver** pools are connected via dashed arrows, or message flows, representing the technical connectivity between the respective participants. A dedicated label details how the two are connected: the sender and the integration flow via the Simple Object Access Protocol (SOAP) and the integration flow with the receiver via SSH File Transfer Protocol (SFTP). Dedicated channels take over the responsibility of implementing and speaking the respective protocols.

What's left is the integration flow in the middle, which contains details about the message processing steps. The circle on the left of the flow containing a white envelope is a *start event* (again, BPMN nomenclature). It expresses the start of the message processing sequence. The envelope indicates the start of the process caused by the reception of a message. After the flow has received the message, it continues with the content modifier step because they are connected with a directed solid arrow, also known as a *sequence flow*. The arrow indicates in which sequence steps are executed.

The rounded rectangle step is called a *task*, and indicates an activity that is executed during runtime. In our example, the content modifier changes the message content, as you'll see later in this chapter. After the task has finished its modifications, the process continues downstream. Again, just follow the sequence flow to the circle on the right, which contains a black envelope and represents an *end event* (circles in general stand for events in BPMN).

Black symbols and white symbols in events have a special meaning: white means waiting for the event to happen, and black means that the process itself initiates the event specified by the symbol inside the circle and immediately continues. In this example, the process waits for an incoming message at the beginning (white envelope) and sends a message after the process has reached the end of the flow (black envelope). The sent message is then transported via SFTP to the receiver. This is expressed by the message flow from the end event to the **Receiver** pool in [Figure 4.1](#).

4.1.1 Message Processing: The Apache Camel Framework

Let's concentrate on the message processing part of SAP Cloud Platform Integration for a moment. As you now know, from a modeling perspective, the flow is based on BPMN. But how is the integration flow interpreted and executed during runtime? For this purpose, SAP Cloud Platform Integration relies on an open-source integration framework called Apache Camel (or Camel for short). To understand the inner workings of SAP Cloud Platform Integration, you should be familiar with the inner workings of Camel. That's why you find additional names for message processing in [Figure 4.1](#), because, in Camel, the respective terms for the handling of messages are *route* or *processor chain*. It doesn't make sense to discuss the complete Camel framework in a chapter like this. We only concentrate on Camel's functions and features that are relevant for the respective scenario we're going to discuss. However, Claus Ibsen and Jonathan Anstey have written an excellent book about Camel called *Camel in Action* (Manning Publications Co., 2010). This book is a helpful reference for all detailed Camel questions you might have. Another valuable source of information for the Camel framework is the online documentation, found at <http://camel.apache.org/documentation.html>.

Note

Be aware that you can't just use any Camel feature, property, or header in SAP Cloud Platform Integration. You should only use the features, properties, and headers that

are explicitly supported by SAP Cloud Platform Integration. Please refer to the SAP documentation found at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud.

So, what exactly is Camel? It's a message routing and mediation engine. Interestingly enough, Camel is payload-agnostic, which means you can feed the engine with any data format, and Camel forwards it to the respective receivers depending on the modeled route. As long as there is no need to access the message's content (e.g., for routing purposes), Camel can handle any message format. However, some basic structure must also be available in Camel, as depicted in [Figure 4.2](#).

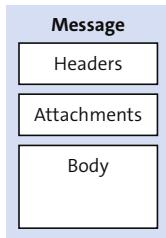


Figure 4.2 Camel's Message Model

Camel messages consist of headers, a body containing the raw data (the payload), and (optional) attachments. Messages are uniquely identified by an identifier of the type `java.lang.String` (not shown in [Figure 4.2](#)). The headers are additional values associated with the message, such as sender identifier, hints about content encoding, and authentication information. This information is added as headers in the form of name-value pairs. The name is a unique, case-insensitive string, whereas the value is of the type `java.lang.Object`. This is quite interesting, as almost anything can be added as an object to the header. The same is applicable for the body, which is also of the type `java.lang.Object`. Attachments are typically used for web service and email components and can transport additional data as separated items, if necessary.

During message processing, Camel requires a dedicated container for the message. The container is called an *exchange*, and it holds additional data besides the message. The exchange is passed along, step by step, in the processor chain, and every step has access to all the information the exchange carries. It can be seen as a global storage for the route as long as the message is being processed. The structure of the exchange is shown in [Figure 4.3](#).

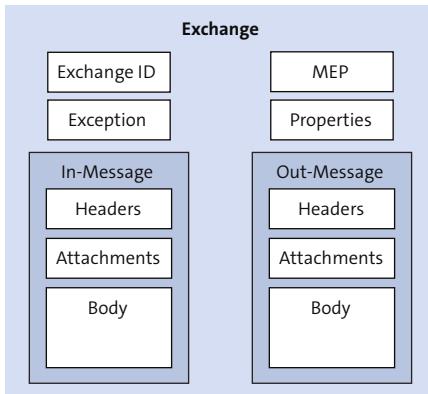


Figure 4.3 An Exchange

Let's briefly go over the parts that make up an exchange:

- **Exchange ID**

A unique ID that identifies the exchange.

- **MEP**

Short for message exchange pattern, field can contain two possible values: **InOnly** and **InOut**.

- **InOnly**: The route handles a one-way message, where the sender doesn't wait for a reply from the receiver. Hence, the exchange carries an *in* message *only*. A scenario where a message travels in one direction only and where no response message is expected during the communication is also known as *asynchronous message handling*.

- **InOut**: The route handles a request-response message. The sender expects a reply from the route, which will be stored as an out-message in the exchange. This behavior is also known as *synchronous message handling*. We'll revisit asynchronous and synchronous message handling in [Chapter 5, Section 5.3](#).

- **Exception**

If an error occurs during message processing, the reason for the error is stored in the **Exception** field of the exchange.

- **Properties**

A form of temporary storage where process steps can store data in addition to the header area in the message. Properties can contain global-level information. Developers can store and retrieve properties at any point during the lifetime of an exchange.

Difference between Headers and Properties

Note that headers are part of a message and are propagated or transferred to a receiver. On the other side, properties last for the entire duration of an exchange but aren't transferred to a receiver.

A big difference regarding message handling within SAP Cloud Platform Integration, as compared to SAP Process Integration, is the flexible pipeline concept that stands behind Camel. In SAP Process Integration, you basically have three fundamental steps:

1. Receiver determination
2. Interface determination
3. Mapping

In addition, the sequence of these three steps is fixed. It's not possible to have, for example, a mapping step before an interface determination step. The result is a rather static message-processing environment. With SAP Cloud Platform Integration, this changes significantly. You have many more steps at your disposal, and you can use them in (almost) any sequence that your scenario requires. We'll demonstrate the benefits of this flexibility in later chapters when we address more complex integration scenarios.

Notice that we've also shown already in [Chapter 2, Section 2.1](#), how the Camel framework fits into the overall architecture of SAP Cloud Platform Integration (also see Figure 2.6 in [Chapter 2, Section 2.1.4](#)).

4.1.2 Exercise: Working with Camel's Message Model

Now that you have a basic understanding of the data and message models that are used within Camel and SAP Cloud Platform Integration, let's see how you can benefit from this knowledge during message processing. To demonstrate how to access properties and headers when building a response message and to illustrate the consequences of storing data in the two locations, we'll use a simple scenario. In this example, we take an XML input message and store parts of the message in the properties/header area of the message model. To build the reply, you access the previously stored data from the properties/header areas and construct the response message. Let's get started!

1. We'll first create a new package for our exercise. After you connect to your SAP Cloud Platform Integration tenant with your Internet browser, switch to the **Design** view ([Figure 4.4](#)).
2. You'll see a list of available packages. Create a new package by clicking the **Create** button located beneath the package list. The **New Integration Package** dialog box opens (see [Figure 4.5](#)).

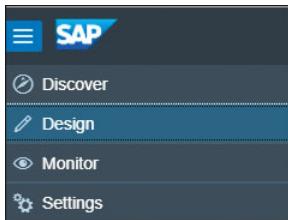


Figure 4.4 Switching to Design View

A screenshot of the 'New Integration Package' dialog box. The title bar says 'Design / New Package /'. Below it is a logo for 'Cloud Integration Book Package'. The top navigation bar includes tabs for 'Header' (which is selected and underlined in blue), 'Overview', 'Artifacts (0)', 'Documents', and 'Tags'. The main form contains several input fields:

- *Name: Cloud Integration Book Package
- *Technical Name: CloudIntegrationBookPackage
- *Short Description: You first Cloud Integration Package
- Version: 1.0
- Vendor: sap.com

Figure 4.5 Creating a New Integration Package

3. Provide some basic information about the package, such as its name, version, and creator. Save it using the **Save** button.
4. From the resulting page, navigate to the **Artifacts** tab and add an integration flow to your package by choosing **Integration flow** from the **Add** dropdown list (see [Figure 4.6](#)).

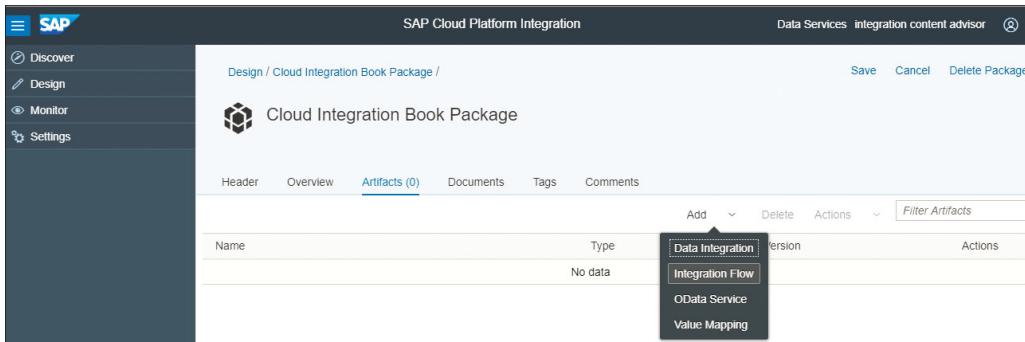


Figure 4.6 Adding an Integration Flow to the Newly Created Package

Another dialog box opens to provide some basic information about the integration flow ([Figure 4.7](#)).

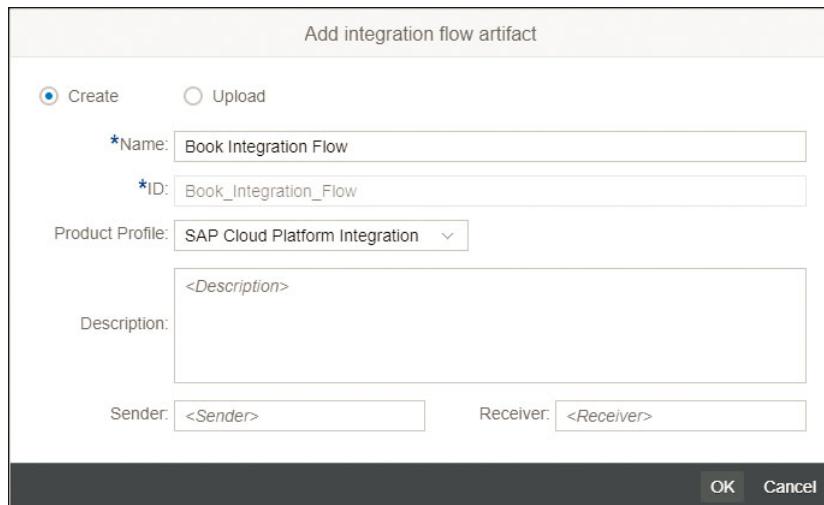


Figure 4.7 Creating an Integration Flow

5. Confirm the new integration flow by clicking the **Ok** button, and then store the new package by clicking on **Save**.
6. You've now created a package and an associated integration flow. Open the integration flow, which should look like the one depicted in [Figure 4.8](#).

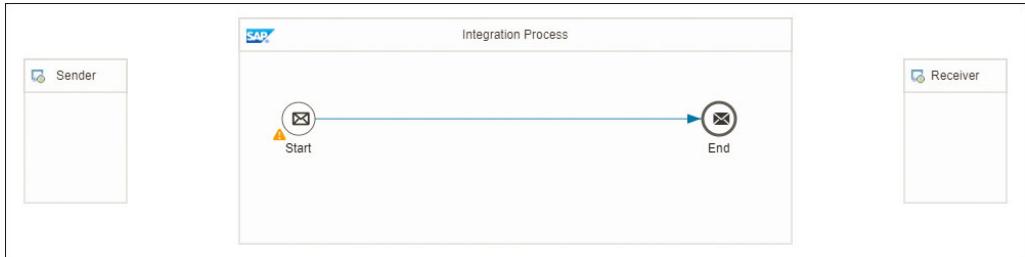


Figure 4.8 The Empty Integration Flow as a Starting Point for the Exercise

4.1.3 Connecting and Configuring a Sender with an Integration Flow

Switch to edit mode (the **Edit** button is in the top-right corner of the integration flow screen), and delete the **Receiver** pool, as we don't need the pool for this exercise. Start modeling the connection between the **Sender** pool on the left and the **Start** event by dragging a connection from the pool to the **Start** event via context buttons. Every time you select a shape in the model, context buttons appear on the right side of the shape to indicate what can be done with the shape at this moment. So, if you select the **Sender** pool, the context buttons shown in [Figure 4.9](#) appear.



Figure 4.9 Context Buttons Next to the Selected Shape

Next click the connector icon (the "connect" icon), drag the cursor to the target shape (the **Start** event in that case), and then release the mouse button (drop) (see [Figure 4.10](#)).

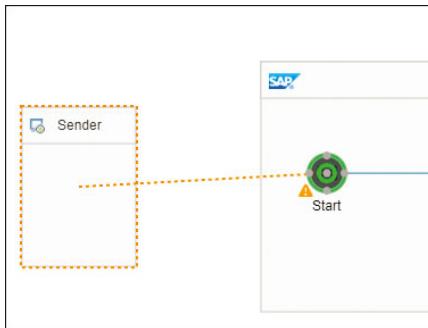


Figure 4.10 Connecting the Sender Pool with the Start Event

After dropping the mouse button on the **Start** event, a dialog box opens automatically that allows you to pick the connection type you want between the **Sender** and the SAP Cloud Platform Integration server. In this case, the connection should be a SOAP adapter and the message protocol SOAP 1.x. Simply pick the respective entries from the dialog boxes.

Finally, you have to maintain one parameter of the SOAP adapter: the address under which the service will be accessible. For this, select the message flow between the **Sender** component and the **Start** event so that it appears in orange (see the dotted line in [Figure 4.11](#)).

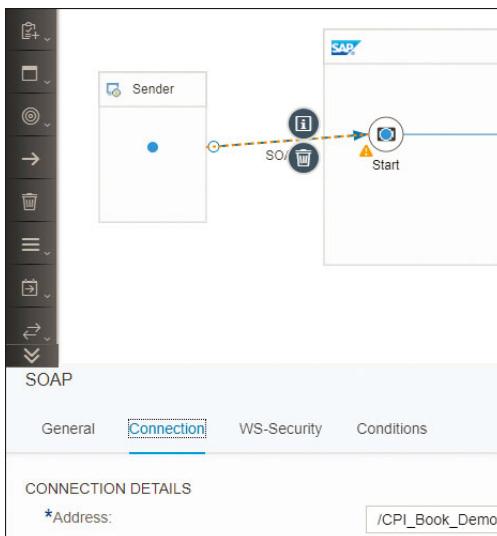


Figure 4.11 Configuration of the SOAP Channel

Beneath the process model, you should see the properties of the connection (in this case, the details of the channel). It's typical for the SAP Cloud Platform Integration modeling environment to select shapes in the graphical model and adjust properties in the area beneath the process model. The properties change depending on the selected component in the main area. In our example, click the **Connection** link and add “/CPI_Book_Demo” as the address in the **Address** field. That's all you have to do to configure the channel.

You also have to define the authentication type for the communication between the sender and the SAP Cloud Platform Integration server. We chose the SOAP adapter for our connection, so the most convenient authentication type is basic authentication, which requires the user's username and password. In SAP Cloud Platform Integration, the User Role authorization can be used either for username and password credentials or with a client certificate. To tell SAP Cloud Platform Integration that this type of authorization should be used, select the message flow between the **Sender** component and the **Start** event, and under the **Connection** tab, choose **User Role** from the **Authorization** dropdown list (see [Figure 4.12](#)).

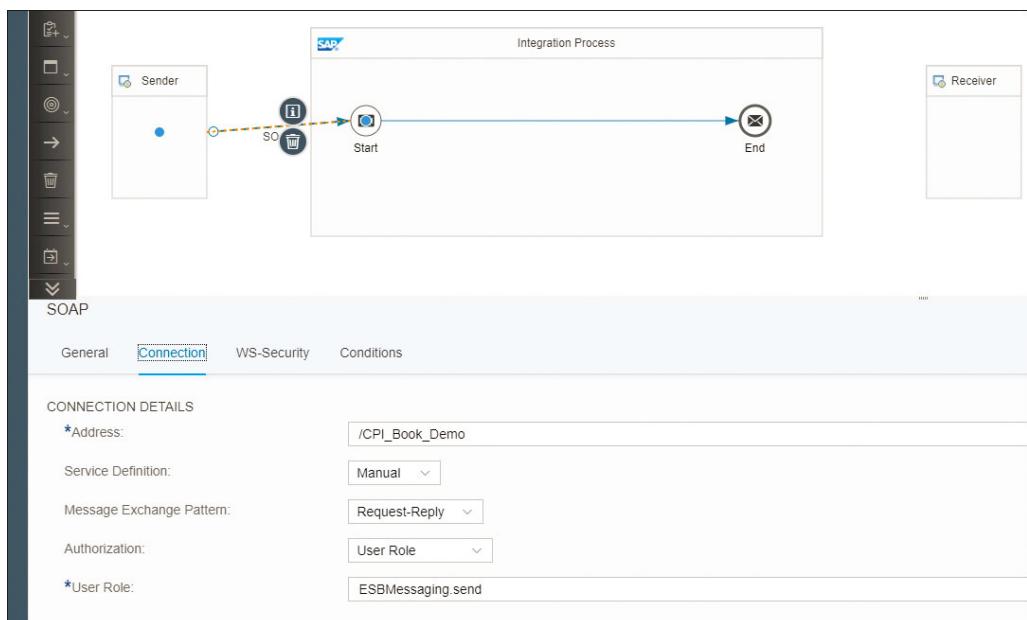


Figure 4.12 Setting the Authentication Type to Basic Authentication

As shown in [Figure 4.12](#), the default value for the user role is **ESBMessaging.send**. This role authorizes a sender system to process messages on a tenant. For now, we'll leave the **Service Definition** and **Message Exchange Pattern** field to their default values.

4.1.4 Adding and Configuring Steps in the Integration Flow

Now, let's add and configure steps in our integration flow. We'll add two **Content Modifier** steps in the model. You can find them in the palette on the left as shown in [Figure 4.13](#).

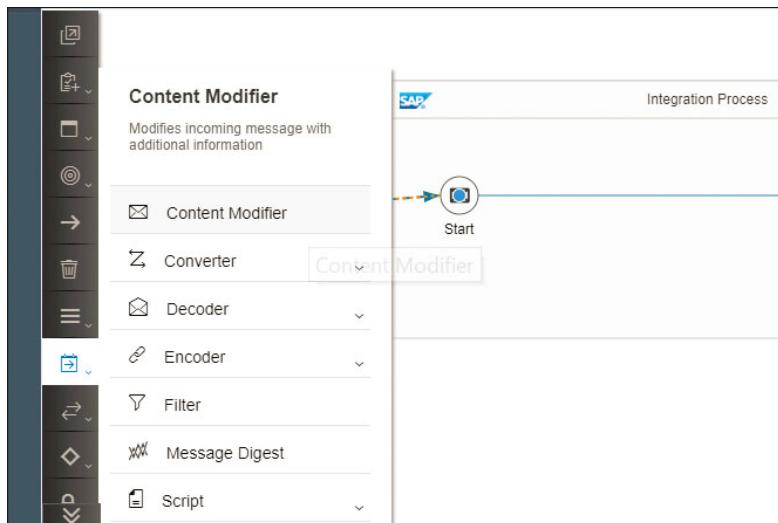


Figure 4.13 Selecting Content Modifier Steps from the Palette

Click the transformer icon  in the palette. A submenu opens. Click the **Content Modifier** shape, which is the black envelope , and move the mouse pointer to the **Integration Process** pool. Click again to position the shape in the pool. Repeat the previous steps so that two **Content Modifier** shapes are added to the process model (see [Figure 4.14](#)).

Let's configure the first **Content Modifier**. Our goal is to write data with the first **Content Modifier** into the header of the message and into the properties area of the exchange. The second **Content Modifier** step retrieves the previously stored data from the respective locations and creates a new result message. By doing this, you prove that data can be stored in different locations and be available during future steps within the route.

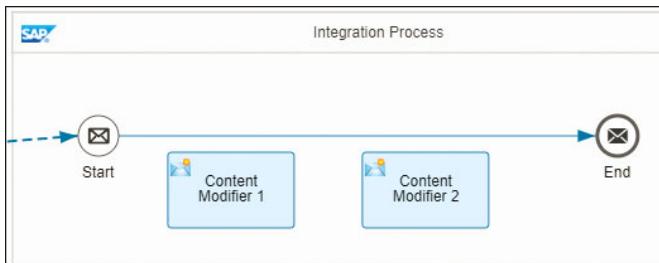


Figure 4.14 Process Model after Adding Two Content Modifier Steps

To retrieve data from the message, it must be clear what the incoming message looks like. We've defined the message structure used in this example for you, using a simple Web Services Description Language (WSDL) document called *GetOrderShipDetails_Sync.wsdl*. You can download the WSDL file from www.sap-press.com/4650. An example message following the WSDL's structure is depicted in [Figure 4.15](#).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4.15 Example Message

Our first **Content Modifier** should retrieve the order number from the message and store it in the header of the message. In addition, it should take the complete message's body (the part between the Body tags in [Figure 4.15](#)) and store it in the properties area. By doing this, we demonstrate two important features:

- Single fields can be accessed, selectively retrieved, and stored in the header area.
- Complete complex structures, such as complete messages, can be retrieved and stored as properties.

Select the left **Content Modifier** by clicking it once so that its properties can be configured. Note that the configuration parameters of the **Content Modifier** distinguish between three locations: **Message Header**, **Message Body**, and **Exchange Property** ([Figure 4.16](#)).

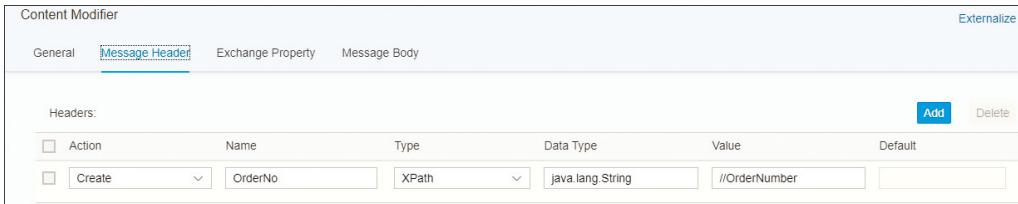


Figure 4.16 Writing Data into the Message’s Header Area

The first **Content Modifier** uses the **Message Header** and the **Exchange Property** areas to write the data. The second **Content Modifier** uses the **Message Body** to create the response message. By selecting the appropriate link shown at the top of [Figure 4.16](#), you always select the location where data is being stored. [Figure 4.16](#) shows the correct configuration of the **Content Modifier** for writing data into the message’s header.

Select the **Message Header** link in the top row. Continue by clicking the **Add** button to create a new row in the table underneath. Next, define the name under which the data should be stored in the message header. In our example, the name is `(OrderNo)`. You can later access the value by using this exact name.

You now have to tell the integration engine how to access the value in the message and the type of data to retrieve. We chose **XPath** from the dropdown list, as we want to navigate inside an XML document, and **XPath** is the way to go to retrieve data from an XML document. The XPath expression is simply `//orderNumber`, as this is the information we’re interested in, and the retrieved value is of type `String`. Here, a valid Java data type needs to be assigned—hence, the entry `java.lang.String`. That is all you need to do to store the order number in the message’s header.

[Figure 4.17](#) shows the configuration of the first **Content Modifier** for writing the complete body of the message into the property area of the exchange. As you can see, it’s possible to have two (or more) write operations configured in a single **Content Modifier** step.

To achieve the result shown in [Figure 4.17](#), click the **Exchange Property** link in the top row. Add a new row in the **Properties** table. The message will be stored under the name `(msg)` in the exchange’s property area. This time, we want to store the complete message. Therefore, we aren’t accessing a single field but the whole body. For this purpose, Camel provides the Simple Expression Language, which includes pre-defined variables as shortcuts that allow convenient access to certain parts of a message.

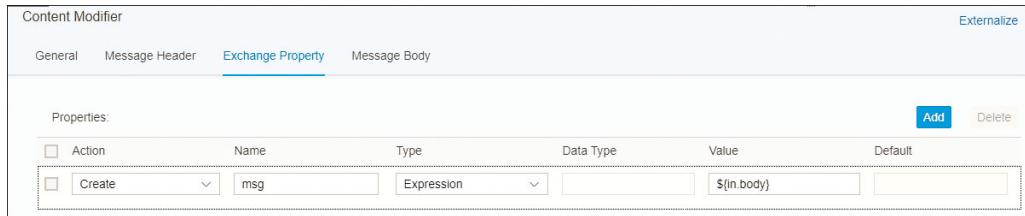


Figure 4.17 Writing Data into the Exchange's Property Area

In our case, we want to access the body of the incoming original message. The Camel variable to do so is `in.body` (as shown in [Figure 4.17](#)). The syntax requires the dollar sign and the curly brackets to actually access the variable's content. In the **Content Modifier** screen, choose **Expression** from the **Type** dropdown list, and add “ `${in.body}`” to the **Value** field.

With that, you've completed the configuration of the first **Content Modifier**. If you want to learn more about Camel's expression language, we recommend checking out the Simple Expression Language documentation at <http://camel.apache.org/simple.html>.

We'll continue with the configuration of the second **Content Modifier** responsible for setting the body of the result message. The correct configuration is shown in [Figure 4.18](#).



Figure 4.18 Defining the Content of the Reply Message as the New Message Body

Select the **Message Body** link in the top row of the **Content Modifier** screen. Next, fill the **Body** input field as shown in [Figure 4.18](#). We defined a new opening and closing tag named `result`. In between those tags, we placed the contents of the two variables defined during the configuration of the first **Content Modifier**. You can see how to access the different storage areas: the properties in the exchange are accessible via

the predefined Camel variable property, and the data is stored in the message's header via the predefined Camel variable header. In both cases, the actual data is accessed by adding the custom declared variable's name, separated by a dot: `property.msg` for the complete message and `header.OrderNo` for the order number.

Now that you've configured both **Content Modifier** shapes, you can add them to your model by dragging and dropping them on the sequence flow. As soon as the sequence flow changes its color to orange (Figure 4.19), you can release the mouse button, and the shape is inserted correctly. After adding both shapes, the result should look like the one shown in Figure 4.20.

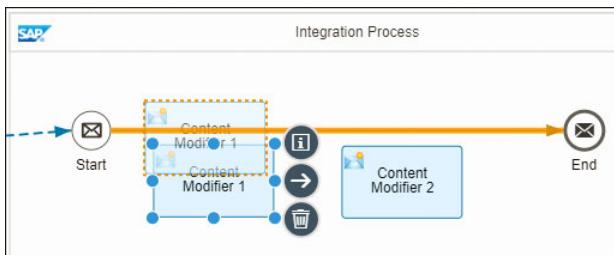


Figure 4.19 Adding the Content Modifier to Your Model

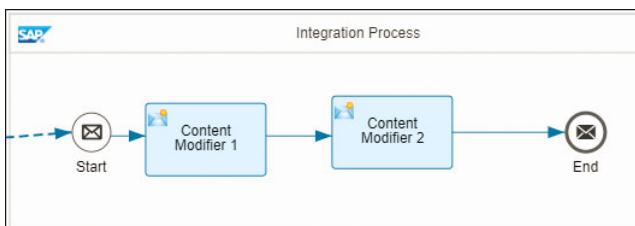


Figure 4.20 The Final Result of the Created Integration Flow

Finally, save your changes, and deploy the integration flow on your tenant.

4.1.5 Checking Configuration Using the Problems View

SAP Cloud Platform Integration supports the development process by performing validation checks each time you open, save, and deploy an integration flow. If these checks fail, warnings or problems are detected in the integration flow, and the concerned problematic step is visually marked with a red **✗** icon (for an error) or a yellow icon **⚠** (for a warning). Figure 4.21 shows that the integration flow model didn't

pass the validation check and that there is a problem with the sender connector—as indicated by the red icon ✖ on the integration flow.

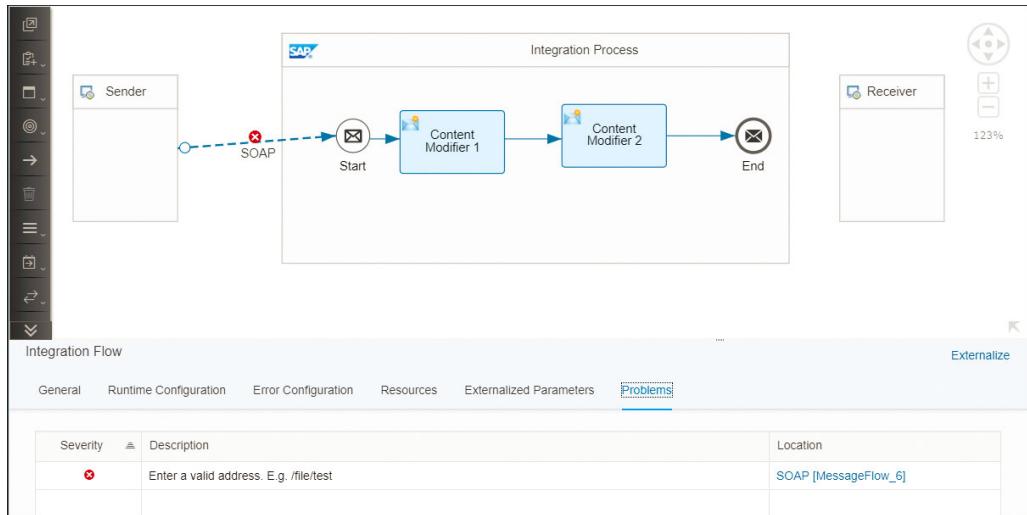


Figure 4.21 Overview of the Problems View Tab

As a developer of the integration flow, you can rely on two options to determine the exact problem causing the error:

- Hover your mouse over the error or warning icon in the integration flow to get a tooltip that describes the issue.
- Use the **Problems** view tab to obtain details about the error or warning.

Note

When a validations check is performed on the integration flow both warnings and errors can be detected and visually shown in the integration flow model and the Problems view tab. Errors are flagged by a red icon ✖, whereas warnings are flagged with a yellow icon ⚠.

The **Problems** view tab helps you manage issues and problems in your integration flow by giving you specific details of the problem's root cause. It displays all design-time issues related to integration components and resources.

Table 4.1 provides a description of the columns available in the **Problems** view tab.

Column Name	Description
Severity	The severity status of an issue. This column is populated by icons that represent an error or a warning.
Description	A description of the problem. This column also often presents a tip or example of how to solve the issue.
Location	The location of the integration component or resource with the issue. The location is the name of the concerned integration component or resource, followed by the ID displayed in brackets. It's also possible to click the location ID to be redirected to the concerned step in the integration flow or to the affected resource.

Table 4.1 Attributes Available in the Problems View Tab

The visual representation of the warnings and errors can get quite cluttered if you have a lot of issues in the integration flow. Furthermore, having to hover over each problem icon before knowing the issues isn't always handy. Compared to the visual representation, the **Problems** view tab has a better overview and a consolidated list of issues in one glance. It's also possible to filter or sort the problems based on the different columns listed in [Table 4.1](#).

4.1.6 Running the Integration Flow

After a successful deployment, you need the endpoint of your integration flow to actually invoke the integration flow via a SOAP client, such as SoapUI (<http://soapui.org>). You can find the endpoint's address using SAP Cloud Platform Integration's monitoring environment. Choose **Monitor** from the three-bar navigation icon  in the upper-left corner of the screen ([Figure 4.22](#)).

**Figure 4.22** Switch to the Monitoring Environment of SAP Cloud Platform Integration

A screen with several tiles should appear. Click the **Started** tile, which is beneath the **Manage Integration Content** heading (Figure 4.23). This allows you to navigate to all of the already-started integration flows for your tenant.

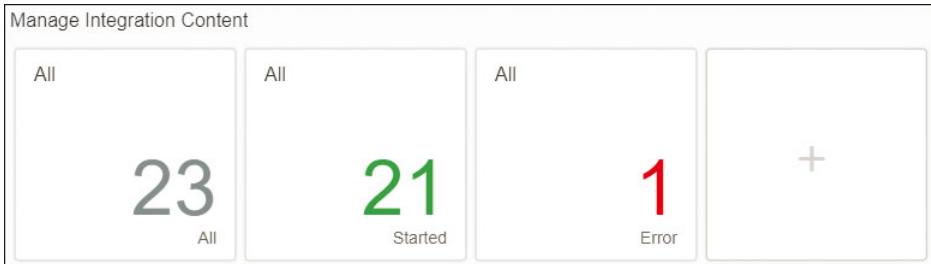


Figure 4.23 Tile for Already-Started Integration Flows on your Tenant

From the list of deployed and started integration content, find your newly deployed integration flow, and click its entry row (Figure 4.24). This takes you to the details of this particular integration flow.

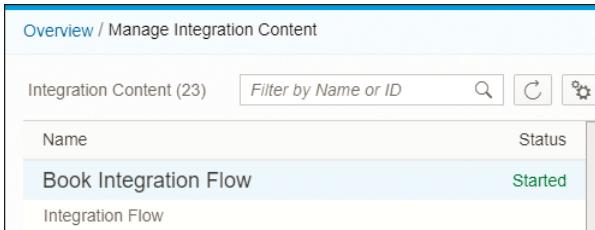


Figure 4.24 Navigate to the Details of a Selected Integration Flow

Besides other useful information, such as the integration flow's version number, the deployed state, and the deployment date/time, the endpoint's URL is of particular interest. It shows the URL by which the integration flow is invoked. Select the URL, and copy the address by clicking on the Copy icon shown in Figure 4.25. Additionally, depending on your needs, you can download the WSDL via one of these options: **WSDL**, **WSDL for ABAP Consumer**, or **WSDL without Policies**. The SOAP tool of your choice requires the URL as the invocation address.

Figure 4.25 Copy the Endpoint's URL into the Clipboard

Now you have all information you need to run the integration flow. Open a SOAP client tool of your choice, such as SoapUI, and paste the copied URL as the invocation target. (We already discussed SoapUI in [Chapter 2, Section 2.3](#).) Prepare a message that looks similar to the one shown earlier in [Figure 4.15](#). Set the **Order Number** field accordingly (e.g., with **10249**), and don't forget to set the correct authentication type in your tool (**Basic Authentication** with the username and password provided by SAP operations for your SAP Cloud Platform Integration instance).

Note

Be aware that the user used in SoapUI needs to have the `ESBMessaging.send` role assigned for the service call to be authorized.

Finally, invoke the integration flow. As a result, you should receive an output comparable to the one shown in [Figure 4.26](#).

```

<soap:Envelope xmlns:soap="http://schemas.xml
    <soap:Body>
        <result>
            <demo:OrderNumber_MT xmlns:demo="htt
                <orderNumber>10249</orderNumber>
            </demo:OrderNumber_MT>
            10249
        </result>
    </soap:Body>
</soap:Envelope>

```

Figure 4.26 Response after Invocation of Integration Flow

In Section 4.1.1, we made you aware of an important difference between storing data in the message's header and storing it in the exchange's property area: the data stored in the message's header will be forwarded to the receiver of the message, whereas the data in the exchange's properties area won't reach the receiver. How can you verify that? In your SOAP client tool, take a look at the headers of your response message. We've done this for SoapUI and created the screenshot shown in [Figure 4.27](#).

Header	Value
Strict-Transport-Security	max-age=31536000; includeSubDomains; preload
OrderNo	10259
Date	Sun, 18 Mar 2018 22:01:57 GMT
Content-Length	348
#status#	HTTP/1.1 200 OK
SAP_MessageProcessingLogID	AFqu4dVfprmUt00JUG_yo10Vi00
Set-Cookie	JSESSIONID=0F2399615D61B296143F2E45A459F6892C02E19DB9B3622EBC3CEB4347EC57E; Path=/cx; Secure; HttpOnly
Set-Cookie	JTENANTSESSIONID_ac965bd8f=eXvJACEI16Wrmqsmu5A5WZalLhi4AA4Q%2FAJkwrtQJSM%3D; Domain=.hana.ondemand.com; Path=/; Secu...
Content-Type	text/xml; charset=UTF-8
Server	SAP

Figure 4.27 The Header Fields of the Response Message

In the second row of the table shown in [Figure 4.27](#), you can identify the **OrderNo** field that was set during message processing. It reached the receiver, but as you can see, the data stored in the exchange's properties area didn't.

The fact that the data stored as header fields are returned in the message response means that we need to carefully use header fields. Storing large messages in header fields will be expensive in terms of resources and can create performance issues.

4.1.7 Troubleshooting

As with every development and runtime environment, errors can occur in SAP Cloud Platform Integration, and there are two main types to expect: errors that occur during deployment, and errors that occur during message handling. In both cases, the starting point for the error's root cause analysis is SAP Cloud Platform Integration's monitoring environment (refer to [Figure 4.22](#) for details on how to open the

monitoring environment). The monitoring environment's homepage consists of several areas, including **Message Processing Monitor**, **Integration Content Monitor**, **Manage Security**, **Manage Stores**, **Access Logs**, and **Manage Locks**. Some of these monitors can be seen in [Figure 4.28](#). We'll discuss monitoring applications in more detail in [Chapter 8](#).

If you're dealing with deployment errors, you'll find appropriate entries in the **Integration Content Monitor** area; the **Error** tile gives you an overview by displaying the number of erroneous artifacts. Click on the tile to learn more about the root cause of each error.

For runtime issues during message processing, the **Message Processing Monitor** section has the appropriate tiles to further analyze the errors. However, the tiles are useful for more than erroneous message handling routes. They also reveal a lot about the exact message processing steps within Apache Camel for successfully completed messages. To further illustrate this, perform the following steps:

1. Click on the **Completed Messages** tile in the **Message Monitor** section of [Figure 4.28](#). Note that the tiles can be personalized and configured to your liking. If the tile has been modified in your tenant, there is a chance that it behaves differently than described here.

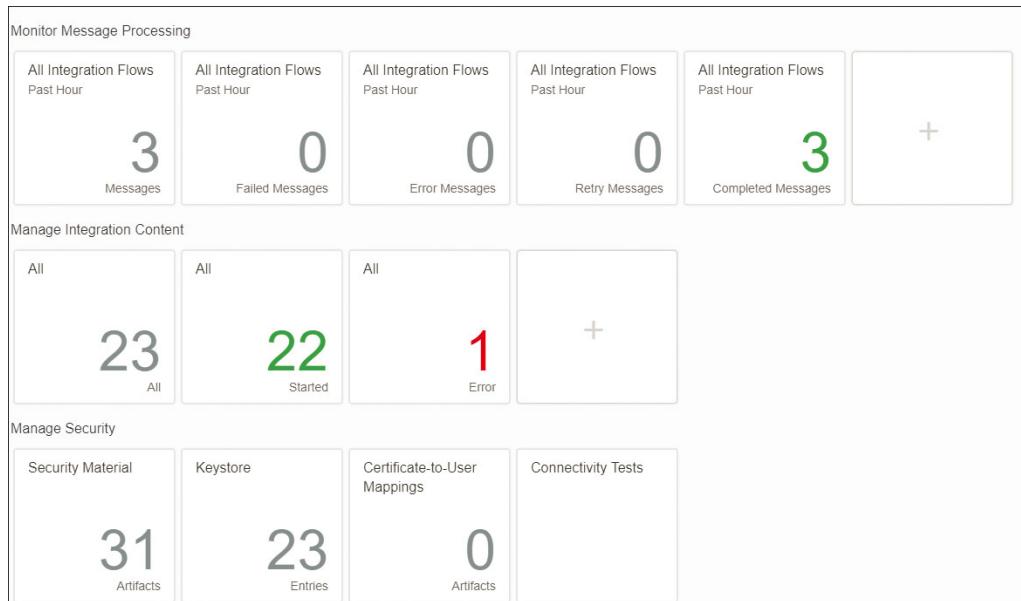


Figure 4.28 Monitoring Homepage

- From the subsequent monitoring page, click on the concerned integration flow on the left side (see [Figure 4.29](#)). Then select the **Logs** tab on the right side to get a page similar to the one displayed in [Figure 4.29](#).

The screenshot shows the 'Monitor Message Processing' overview page. At the top, there are filters for Time (Past Hour), Status (All), Artifact (All Integration Flows), and ID (Message or Application ID). Below the filters, it displays the date range Mar 25, 2018, 21:04:55 - Mar 25, 2018, 22:04:55. The main area shows 'Messages (2)' with a navigation bar. Two messages are listed:

Artifact Name	Status	Timestamp	Duration
Book Integration Flow	Completed	Mar 25, 2018, 21:59:46	53 ms
Book Integration Flow	Completed	Mar 25, 2018, 21:48:01	93 ms

To the right of the message list, there is a summary for the 'Book Integration Flow' with the last update at Mar 25, 2018, 21:59:46. Below the summary, tabs for Status, Properties, and Logs are shown, with 'Logs' being the active tab. Log details include 'Log Level: Debug' and 'Runtime Node: vsa3734030'. A link 'Open Text View' is also present.

Figure 4.29 Details about Successfully Completed Messages

Note

[Chapter 8, Section 8.2](#), discusses the various monitors in more detail.

In this section, you've learned a lot about the inner workings of SAP Cloud Platform Integration. Although we implemented a relatively simple scenario, doing so revealed many details about message handling within the underlying Apache Camel integration framework, including error analysis. This knowledge helps you work more consciously with messages, headers, and attachments. In the next section, we'll discuss how to use externalization in your integration flow.

4.2 Using Externalization to Enable Easy Reuse of the Integration Flow

When designing an integration scenario, it's important to take reusability into consideration. Following are some advantages brought by reusability:

- Reduces the overall cost of software development
- Reduces risks
- Saves time and cost while developing and testing interfaces
- Speeds delivery of software

When it comes to SAP Cloud Platform Integration, it practically means that for every step of your integration flow, you need to ask yourself the following questions:

- Is the concerned step likely to be repeated several times in your integration flow?
- Are there other steps in the integration flow which will reuse the same data?
- Does the step need a dynamic input coming from a variable or another step in the integration flow?
- Does the concerned integration flow need to be deployed in different tenants in the landscape, and do some of its steps need to be configured differently in other environments?
- Does it depend on the tenant which values are to be used for the adapters at run-time? (For example, the endpoint of a connected backend system might depend on whether you configure an integration flow on a test or on a productive tenant.)

Note that there are more potential questions that fit this category, but these are used for the sake of simplicity. If the answer to any of these questions is yes, consider generalizing the concerned step by making it more generic and configurable.

While working with SAP Cloud Platform Integration, one way to generalize and increase the reusability of data within an integration flow is known as externalization. In the next sections, we discuss externalization and explain how it can be used in an integration flow. We'll also discuss how the parameters used for externalization can be configured.

4.2.1 Externalize

If the integration content must be used across multiple landscapes or environments (e.g., development, test, acceptance, production), it can be assumed that the endpoints of the integration flow for each landscape will differ. In such a scenario, externalizing parameters can be used to declare a parameter as a variable at design time. The parameter can then be used later during configuration time to customize the attributes on the sender and receiver side or for a step in the integration flow.

Externalizing a parameter has the advantages of avoiding hard-coded values in your integration flow and providing the flexibility to change the parameter values at configuration time. *Configuration time* refers to a moment after an integration flow has been transported or imported in a tenant in your landscape (test, acceptance, or production). During this time, users might need to adapt some aspects of an integration flow, such as adding adapter-specific endpoints and assigning values to externalized

parameters. The good news is that assigning a value to an externalized parameter during configuration doesn't involve changing the integration flow.

Let's now look at a practical use case by referring and extending the integration flow that we implemented in [Section 4.1](#). We've built an integration flow that accepts a SOAP call. Imagine that you, as the integration developer, decide to have different addresses for calling the integration flow in the different environments (development, test, and production). As shown previously in [Figure 4.11](#), we configured the flow to be called via the SOAP address `/CPL_Book_Demo`. To externalize the address and make it configurable for each environment, follow these steps:

1. Open the concerned integration flow (in the Web UI design environment), click **Edit**, and select the SOAP adapter line, as shown in [Figure 4.30](#).
2. Click on the **Externalize** button (see right corner of [Figure 4.30](#)).

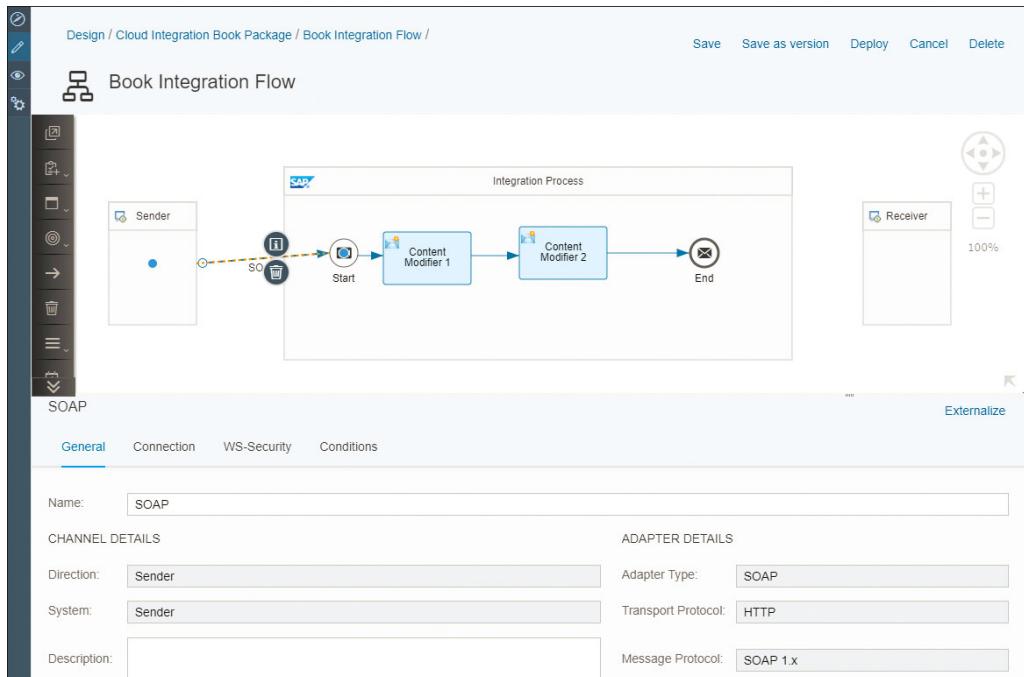


Figure 4.30 Selecting the SOAP Connector and the Externalize button

3. A new screen pops up with the **Externalization** editor, as shown in [Figure 4.31](#). Note that the **Externalization** editor presents all possible configurations of the component that is selected. Let's now create a new parameter on the **Address** field. For

that, you'll need to select the **Connection** tab and remove the existing /CPI_Book_Demo value.

4. The parameter is defined using the format {{<parameter>}}. In our case, let's add a new parameter with the name {{MY_ADDRESS}}, as shown in [Figure 4.31](#).

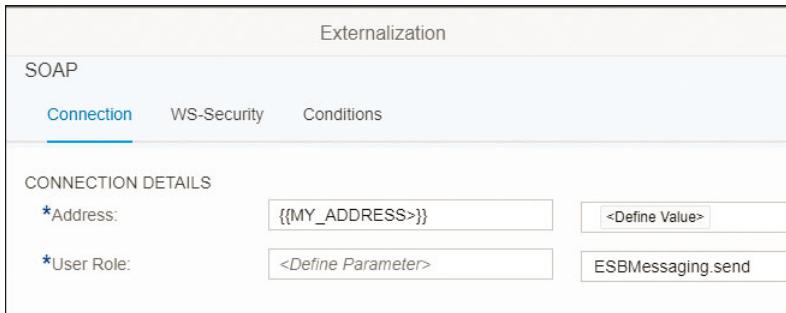


Figure 4.31 Creating a New Parameter in the Externalization Editor

Note

Not all components can be externalized. After selecting the component in the integration flow, the **Externalize** button only appears if the component contains at least one attribute that can be externalized.

Note that the feature to create a parameter is only enabled when the integration flow is in edit mode. As soon as a parameter is created, it becomes available for various components (e.g., sender channel, receiver channel, timer, etc.) within the same integration flow. Furthermore, existing parameters can always be recalled by typing the double curly brackets {{ in the parameter column. SAP Cloud Platform Integration will automatically suggest a list of parameters available in the integration flow.

5. The next step is to create a value for the externalized parameter by clicking on the <Define Value> tag shown in [Figure 4.31](#). The **Define Value** option appears as soon as you've created an externalizable parameter. You then get a new popup screen with the possibility to define a value for the {{MY_ADDRESS}} externalized parameter (see [Figure 4.32](#)). Fill in /CPI_Book_Demo as its new value.
6. Select the **Ok** button to return to the **Externalization** editor.
7. Once back in the **Externalization** editor, select **OK** again to return to the integration flow.

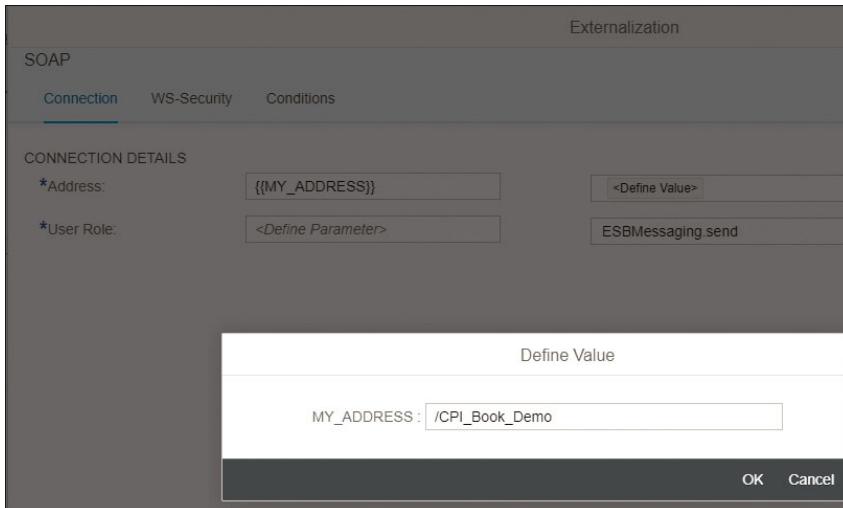


Figure 4.32 Defining a Value for an Externalized Parameter

Note

For some fields in SAP Cloud Platform Integration, it's possible to externalize many parameters at the same time and combine them together to achieve your requirement. Figure 4.33 shows multiple concatenated strings.



Figure 4.33 Example with Multiple Parameters Concatenated

Furthermore, pay special attention when modifying the parameter value because it might result in changing the configuration of other integration flow components using the same parameter.

You can also withdraw from externalizing a field by deleting it. Removing an externalization of a particular field doesn't remove it from the integration flow, however,

because the parameter might be used in another component, step, or adapter. The parameter can completely be removed using the **Externalized Parameters** view, which enables you to manage all externalized parameters of the concerned integration flow.

Following are the steps necessary to manage externalized parameters in the **Externalized Parameters** view:

1. Click outside the integration flow model area to see the **Externalized Parameters** tab.
2. Open the integration flow in edit mode, and select the **Externalized Parameters** view, as shown in [Figure 4.34](#). Note that this shows a list of all existing externalized parameters of the whole integration flow. These parameters aren't necessary for our example exercise; they have been added just for illustration sake.

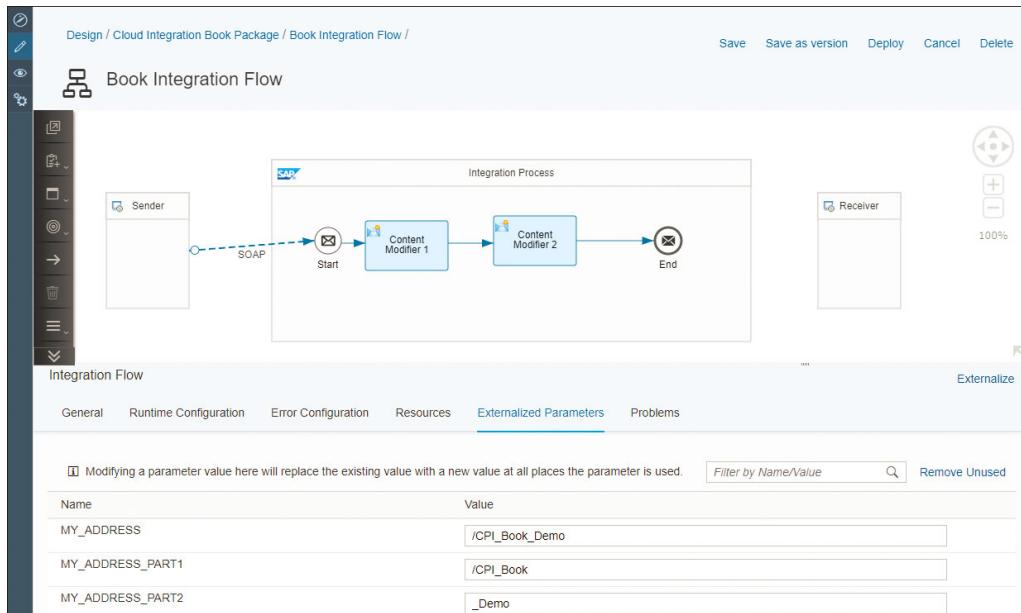


Figure 4.34 Externalized Parameters View in Design Time

3. In this view, you can also filter the parameters using their names or values via the **Filter by Name/Value** field (see right corner of [Figure 4.34](#)).
4. You can also update and adjust the values of any parameters from this central view.

- Lastly, you can remove unused parameters using the **Remove Unused** link on the left side (see [Figure 4.34](#)). When you've done this, SAP Cloud Platform performs a check to identify unused parameters, and a screen appears with a request to confirm the removal of unused fields (see [Figure 4.35](#)). Upon confirmation, all unused parameters will be removed.

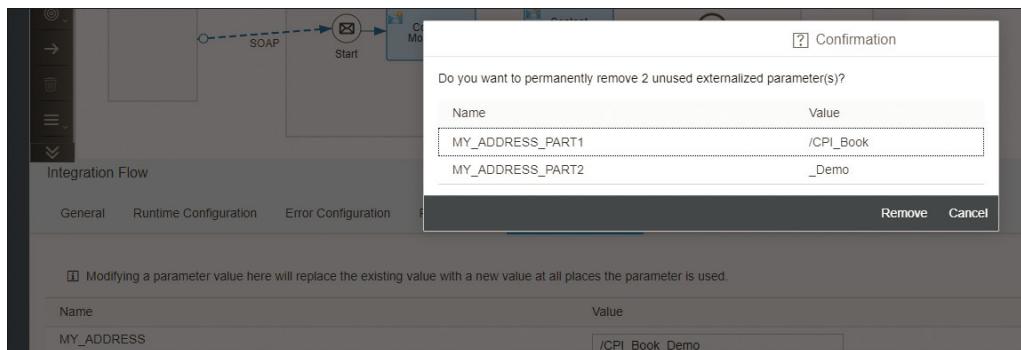


Figure 4.35 Removing Unused Externalized Parameters

- Save and deploy the integration flow.

Now that you know how to add and manage externalized parameters, let's discuss how they can be configured and used for runtime.

4.2.2 Configure and Run the Scenario

As discussed in the previous section, externalization enables us to create a parameter in design time that can later be changed during configuration time, without the need to change the integration flow. Now that you've successfully developed your integration flow and imported it across other tenants in the landscape (e.g., test, acceptance, production), it's time to learn how to adapt the values of its existing externalized parameters to fit the requirements of the actual environment. Note that an integration flow can be transported across other tenants in the landscape (as will be addressed in [Chapter 6](#)).

In the example case that we've been building since [Section 4.1](#), we added a parameter to specify a different SOAP address for each environment ([Section 4.2.1](#)). To change the SOAP address, proceed as follows:

- Transport the integration flow to the other tenant you want to run the integration flow with different parameters.

2. Go to the desired tenant, navigate to the **Design** section, and open the concerned integration flow.
3. Click on the **Configure** link in the top-right corner (see [Figure 4.36](#)).

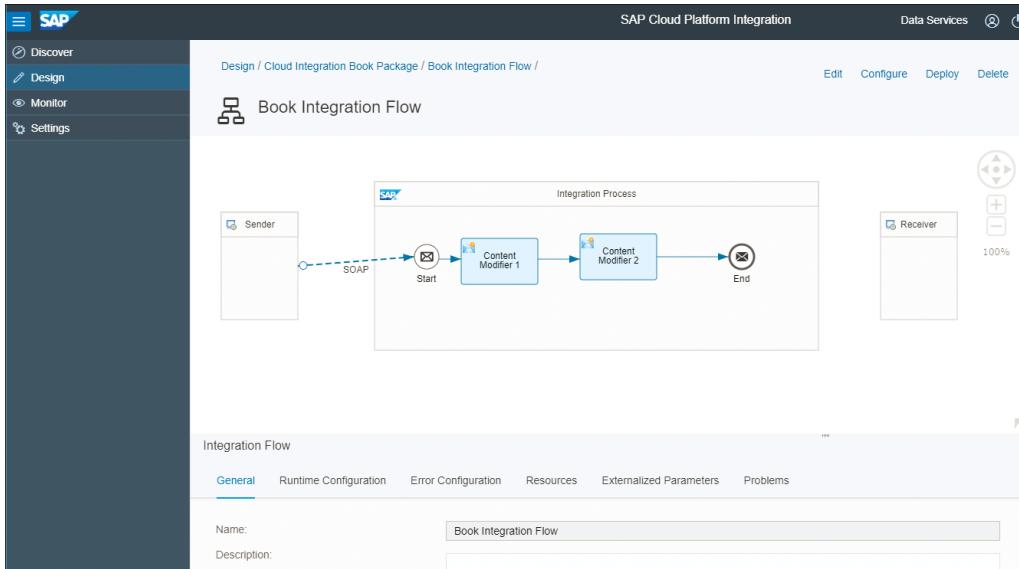


Figure 4.36 Configuring an Integration Flow

4. A **Configure** view is displayed that allows you to maintain values of externalized parameters. In our case, we can now replace the value of the **MY_Address** field with a new value. As illustrated in [Figure 4.37](#), we'll now use the value `/CPI_Book_Demo_Test`.

Configure "Book Integration Flow"	
Sender	
Sender:	<input type="text" value="Sender"/>
Adapter Type:	<input type="text" value="SOAP"/>
Connection	
Address:	<input type="text" value="/CPI_Book_Demo_Test"/>

Figure 4.37 Changing an Externalized Parameter Value

- Click **Save** and then **Deploy** (both buttons are located in the bottom-right corner as shown in [Figure 4.37](#)).

Now that our updated integration flow is deployed, we're ready to test it using SoapUI. Note that the steps to test with SoapUI were already described in [Section 4.1.6](#). The main difference here is that we should now use the new invocation address instead. This new invocation address or endpoint can be found in the **Manage Integration Content** monitor, as shown in [Figure 4.38](#).

The screenshot shows the SAP Cloud Platform Integration interface. On the left, there's a sidebar with 'Discover', 'Design', 'Monitor' (which is selected), and 'Settings'. The main area is titled 'Overview / Manage Integration Content' and shows 'Integration Content (24)'. A search bar and filter options are at the top. Below is a table with columns 'Name' and 'Status'. One row is highlighted: 'Book Integration Flow' (Status: Started). To the right, a detailed view for 'Book Integration Flow' is shown with tabs for 'Endpoints' (selected), 'Status Details', 'Artifact Details', and 'Log Configuration'. The 'Endpoints' tab displays the URL: https://p0262-ifmap.hcisbp.eu1.hana.ondemand.com/cxf/CPI_Book_Demo_Test.

Figure 4.38 Finding the New Endpoint Address after Externalizing the SOAP Address

And voila! Congratulations, you're now equipped to define, manage, and configure externalized parameters.

4.3 Content Enrichment by Invoking an OData Service

SAP Cloud Platform Integration allows for the development of sophisticated integration scenarios. In this section, we provide an example of such a scenario: a synchronous call to an Open Data Protocol (OData) service to retrieve the details of an order for a given order number. The client sends the order number to SAP Cloud Platform Integration using a standard SOAP call.

Our example dives into the details of the communication with external data sources, as well as how to use them in synchronous scenarios. Finally, we explain the Query Editor, which is a tool that helps you generate the sometimes quite complex Representational State Transfer (REST) Uniform Resource Identifiers (URIs) for actually invoking the external OData sources.

Note

URI is a generic term to identify the name of a resource uniquely over the network. A URL is a concrete URI for a web address. In this book, when we talk about the OData service, we use the term URI. When talking about the invocation of the integration flow, which is reachable via SOAP using a concrete web address, we use the term URL.

4.3.1 The Target Scenario

The integration scenario in [Figure 4.39](#) depicts the integration scenario we're going to build in the next section.

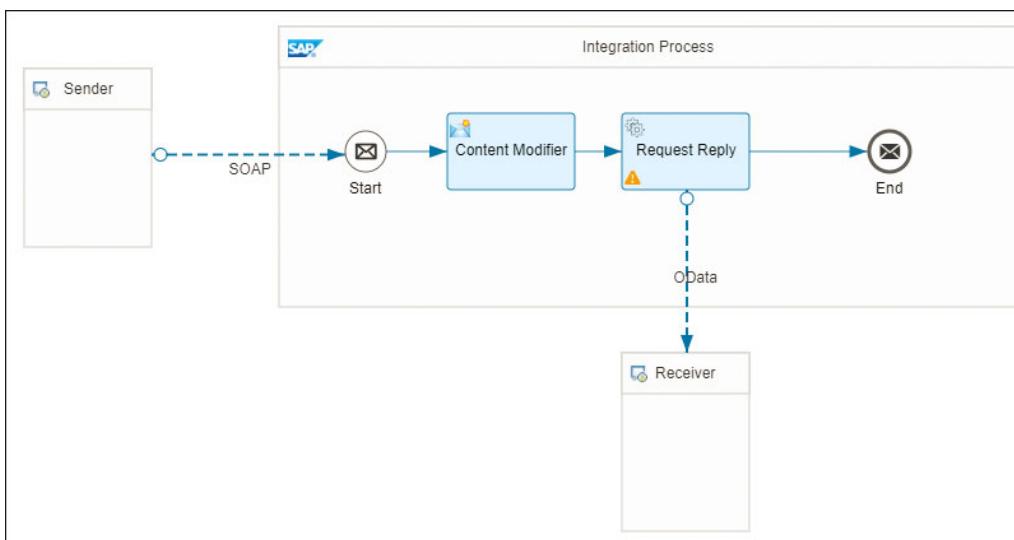


Figure 4.39 Invoking an OData Service

An integration flow covers the following aspects:

- Sending a SOAP request containing an order number from the **Sender** participant to SAP Cloud Platform Integration
- Retrieving the order number from the incoming message and storing it in the message's header field for later reference, using the **Content Modifier** step

- Invoking an OData service to retrieve the details of that particular order, implementing a **Request-Reply** pattern scenario
- Returning the received data to the client

The incoming message's structure is exactly the same as shown earlier in [Figure 4.15](#). It contains the order number for which additional data should be retrieved from an external OData service. You can reuse most of the steps that we used in the previous exercise to create this new integration flow. The **Sender** pool is configured with **User Role** from the **Authorization** dropdown list and with **ESBMessaging.send** as the value for the user role (in the **Connection** tab).

For the message flow between the **Sender** pool and the **Start** event, the adapter-specific attribute **Address** should be set to **/CPI_Book_Demo_OData** (to define a different endpoint compared to the previous exercise). This string is later part of the URL under which the integration flow can be invoked. The **Content Modifier** step is responsible for storing the incoming order number in the message's header area. As a reminder, you can find its configuration in [Figure 4.16](#). The **OrderNo** name was chosen as the storage's name to access it later. So far, the message has arrived at the integration flow, and an important piece of information, the order number, has been stored for later reference. Now the time has come to configure how to call the OData service.

4.3.2 Invoking an OData Service

The modeling environment of SAP Cloud Platform Integration provides a dedicated shape (or step) for accessing external sources synchronously. This shape is the **Request-Reply** step, located in the submenu of the **Call** icon ↗ on the editor's palette ([Figure 4.40](#)).

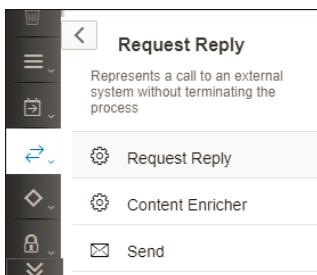
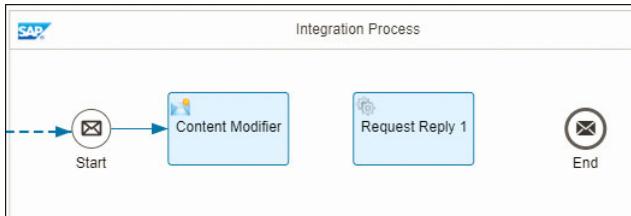


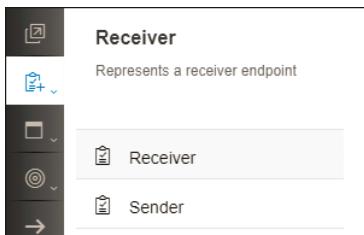
Figure 4.40 Modeling the Request-Reply Step

In the submenu, click the **External Call** entry, and then select the **Request-Reply** entry. Move the mouse pointer into the **Integration Process** pool, and click again to position the step on the canvas. [Figure 4.41](#) shows the result.



[Figure 4.41](#) The Integration Process Pool with the Request-Reply Entry Added

Because the **Request-Reply** step needs access to an additional system, it's necessary to add the target system as an additional pool in the process model. You can find the respective **Receiver** shape in the submenu of the palette's main entry **Participants**. [Figure 4.42](#) shows more details.



[Figure 4.42](#) Picking an Additional Receiver from the Palette

This next step is important: Position **Receiver1** outside the **Integration Process** pool close to the **Request-Reply** step, as shown in [Figure 4.43](#).

Next, connect the **Request-Reply** step with the **Receiver1** pool by using the connection icon from the **Request-Reply** step's context buttons. After dragging the connection to the **Receiver1** pool, a dialog box automatically opens asking you for the **Adapter Type**. Choose **OData**, and you should see something similar to [Figure 4.44](#).

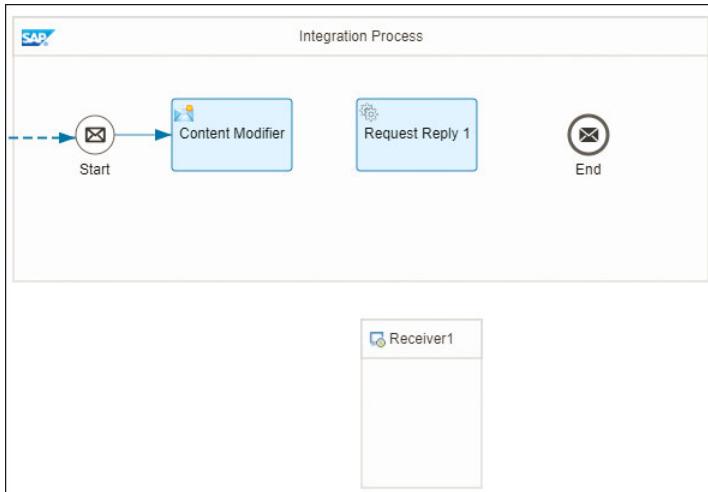


Figure 4.43 Positioning the Receiver1 Pool Close to the Request-Reply Step and Outside the Integration Process Pool

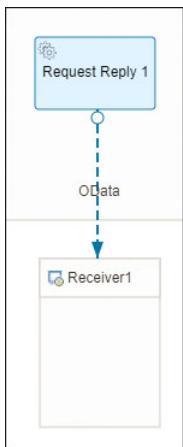


Figure 4.44 Connecting the Request-Reply Step with the Receiver1 Pool

4.3.3 Configuring the OData Connection

To actually invoke an external OData source, you need to configure several parameters. After selecting the message flow between the **Request-Reply** step and the **Receiver1** pool (the connection's color should switch from blue to blue with dotted orange),

the parameters are depicted beneath the process diagram. They are shown in [Figure 4.45](#) for the connection parameters and [Figure 4.46](#) for processing parameters.

OData

Externalize

General Connection Processing

CONNECTION DETAILS

*Address:

*Proxy Type: Internet

*Authentication: None

Figure 4.45 The Connection Parameters of the OData Connection to the External Source

OData

Externalize

General Connection Processing

PROCESSING DETAILS

*Operation Details: Query(GET)

*Resource Path:

Query Options:

Custom Query Options:

Page Size: 200

Process in Pages

*Timeout (in min): 1

Figure 4.46 The Processing Parameters of the OData Connection to the External Source

Let's walk through the configuration fields of the **Connection** tab (see [Figure 4.45](#)) one by one:

■ Address

This field contains the service's root URI of the OData service provider to which you want to connect. In our example, we want to connect to a publicly available service on the Internet: one of the OData demo services available for trying out

OData connectivity and learning more about how to work with OData services. It's reachable under the URI <http://services.odata.org/Northwind/Northwind.svc>.

- **Proxy Type**

This field defines whether you're connecting to a cloud system (**Internet**) or to an on-premise system via the SAP Cloud Platform Connectivity service (**on-premise**). Because we're connecting to a cloud-based OData service, we leave the field on the **Internet** entry.

- **Authentication**

Depending on the OData service, you can choose different authentication types from the dropdown list. **Basic**, **Client Certificate**, **Principal Propagation** (only available for the **ON-PREMISE** proxy type), and **None** are currently supported. For our example, select **None**, as no authentication is required for the test service.

Let's now walk through the configuration fields of the **Processing** tab (see [Figure 4.46](#)) one by one:

- **Operation**

OData is based on HTTP, which supports the following main operations: **GET**, for querying a set of entities or for reading one concrete entity (business object); **PUT**, for updating an entity; **POST**, for inserting data, **Merge** to merge entities; and **Delete**, for deleting an entity. Our requirement is to read exactly one order, so we're fine with the **GET** operation for this field.

- **Resource Path**

In this field, you specify the URI that is appended to the OData service endpoint when connecting to the service provider. You may know this extension by heart (which is rarely the case because of the strict syntax you have to follow and the lengthiness of the string), but it's easier to let the SAP system's built-in Query Editor do the dirty work of creating the resource path for you. Later in this chapter, we'll explain how to work with the Query Editor to create the correct resource path.

- **Page Size**

This final field specifies the total number of records that should be returned in the response from the OData service provider. We're leaving this field empty because we're defining a query that returns only one entry, so no limitation of the response is necessary.

From the preceding description, you can understand the importance of the **Resource Path** entry compared to the other fields of the **Processing** tab. That's why SAP provides the Query Editor to accurately formulate the string representing the resource path.

4.3.4 Creating the Resource Path Using the Query Editor

Click the **Select** button next to the **Resource Path** input field in [Figure 4.46](#) to start the Query Editor. A wizard dialog box opens, asking you for details of the OData service provider to which the Query Editor should connect. Ensure that the **Connect to System** step is populated, as shown in [Figure 4.47](#).

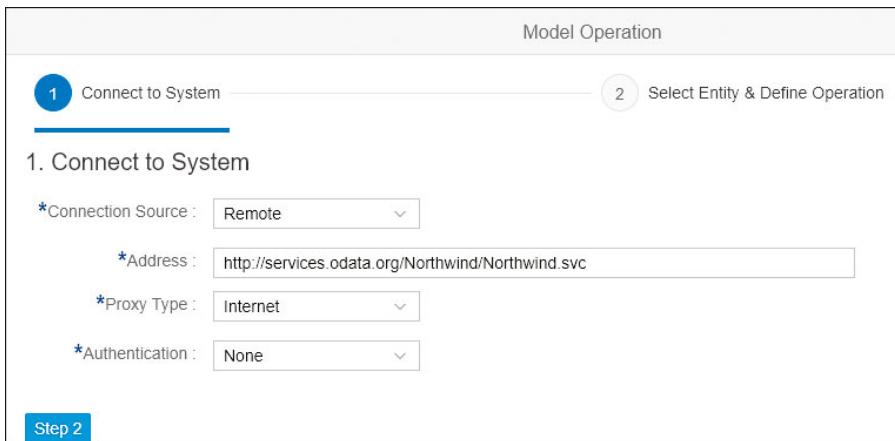


Figure 4.47 Providing Details about the OData Service Provider

Click on the **Step 2** button at the bottom of [Figure 4.47](#) to proceed to the next step of the wizard.

The Query Editor connects to the service and retrieves its metadata information ([Figure 4.48](#)). Hence, it knows exactly which entities can be accessed and how.

Let's start with the entity and the operation. Because the Query Editor has read the details about the available entities, it can list them as part of the **Entity** dropdown list options (see [Figure 4.48](#)). To read exactly one concrete order, pick **Orders** from the list. For **Operation**, stick with the **Query (GET)** operation. Next, specify the fields that should be returned from the service. As you can imagine, an order can consist of hundreds of fields, but, most likely, you need just a handful for your concrete scenario. Therefore, pick just the fields relevant for you.

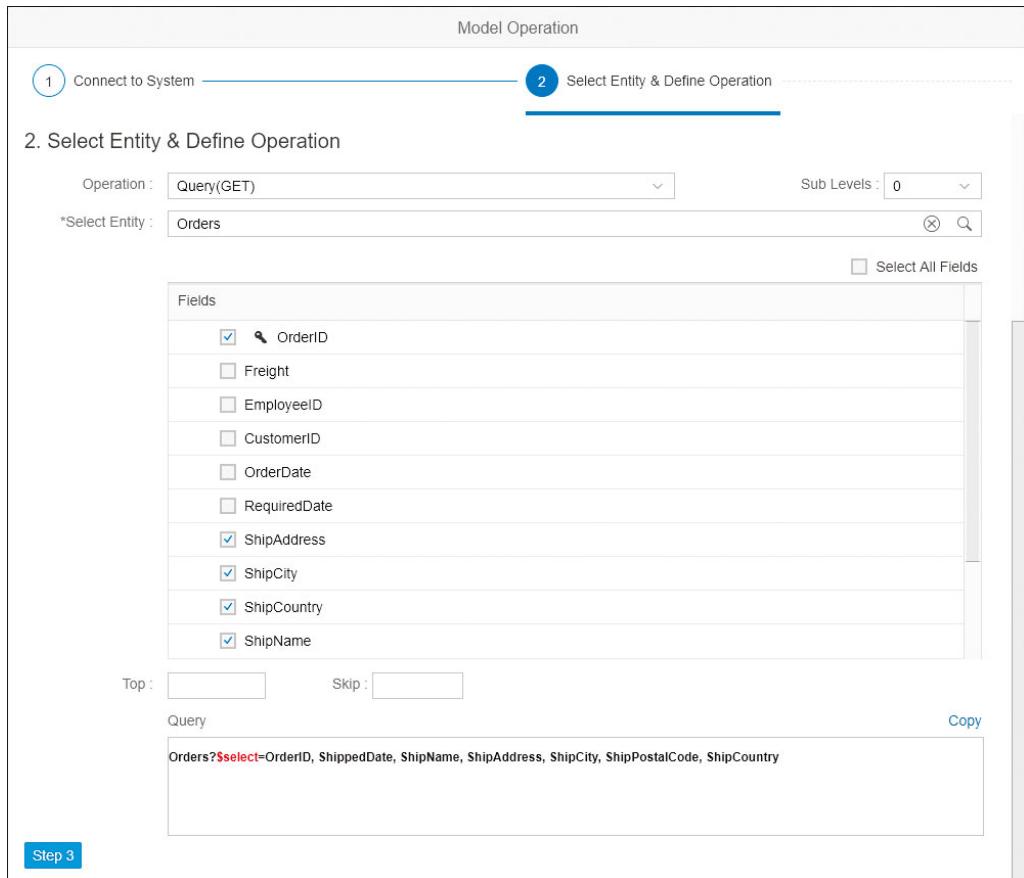


Figure 4.48 Defining the Entity, Operation, and Fields

From the **Fields** table list ([Figure 4.48](#)), select the required fields by checking their respective checkboxes. This information is again retrieved from the service's metadata information. For the sake of simplicity, choose the following fields: **OrderID**, **ShippedDate**, **ShipName**, **ShipAddress**, **ShipCity**, **ShipPostalCode**, and **ShipCountry**. After finishing your selections, the result should look like [Figure 4.48](#).

Note the string being created for you above the circular icons and beneath the **Query Editor** title:

```
Orders?$select=ShippedDate,OrderID,ShipCountry,ShipPostalCode,ShipCity,
ShipAddress,ShipName
```

Then click on the **Step 3** button on the bottom of [Figure 4.48](#) to proceed to the next step of the wizard.

As shown in [Figure 4.49](#), the next screen enables you to configure how the data should be filtered and sorted:

- The filter by condition represents a condition to be used to select the entries that you're interested in. It works similarly to a WHERE clause in an SQL statement.
- The order by condition is used to sort data based on a particular field in ascending or descending order.

Note that to both filter by and order by area of [Figure 4.49](#), it's possible to add as many entries as are required to specify your selection criterion. In our example, you just need one line because you're searching for an order with a concrete, well-defined order number. Check the **OrderID** field within the **Orders** object. This is the exact entry you find in the **Filter By** column of the screen shown in [Figure 4.49](#).

The screenshot shows the 'Model Operation' interface with three steps: 1. Connect to System, 2. Select Entity & Define Operation, and 3. Configure Filter & Sorting. Step 3 is currently active. The 'Configure Filter & Sorting' section includes a 'Filter By' section with a dropdown for 'Select field' containing 'OrderID', an operator 'Equal', and a value ' \${header.OrderNo}' with a 'Remove All' link. It also includes a 'Sort By' section with a dropdown for 'Select field' containing 'Select field', an operator 'Ascending', and a 'Remove All' link. Below these, a 'Query' section displays the generated SQL-like query: 'Orders?\$select=OrderID, ShippedDate, ShipName, ShipAddress, ShipCity, ShipPostalCode, ShipCountry&\$filter=OrderID eq \${header.OrderNo}' with a 'Copy' link.

Figure 4.49 Defining the WHERE Clause in the Query Editor

Next, define the operator. Here, several logical operators are possible, but you only need the **eq** (equals) operator. Next, you need to decide which value you want to compare the entity's **OrderID** field against to find the right order. Presumably, you want to find the order whose number you received in the original incoming message, which you stored in the header area of the message under the variable name **OrderNo** (refer to [Figure 4.16](#)).

In this case, set the value to \${header.OrderNo} to point to the value of **OrderNo**, which was stored in the header attributes. Now, hopefully you understand why you had to store the number in the message header and how to retrieve that number for the where clause. Notice, also, the additional string appended to the previous **Resource Path**. The Query Editor uses the Simple Expression Language to access Camel variables in the header area (\${header.OrderNo}), which we introduced in [Section 4.1](#).

Depending on your scenario, you can also complete the **Resource Path** by using the Order By condition.

The configuration is fairly straightforward: Enter the field's name by which the found entries should be sorted. Again, the Query Editor narrows down the fields after you start typing. You want the list (which later actually consists of exactly one entry) to be sorted by the **OrderID**. You could also define whether the list should be in ascending or descending order by selecting the **Ascending** or **Descending** in the dropdown, but this doesn't play a role in our scenario.

After finishing this step, you've finally completed the configuration of the OData Channel (ODC). Click the **Ok** button to close the Query Editor. You'll then see the message displayed in [Figure 4.50](#).

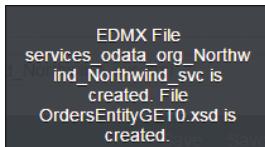


Figure 4.50 Query Editor Confirming That the EDMX and XSD Files Are Automatically Created

The message informs you that two files are automatically created: an Entity Data Model Designer (EDMX) file and an XML Schema Definition (XSD) file. The EDMX file contains the Entity Data Model, which stores the schema of the entities encapsulated in the OData service, including their fields and relationships (e.g., one-to-one, one-to-many, etc.). In our example, we selected an entity earlier in [Figure 4.48](#) for a dropdown field named **Select Entity**.

But what is the XSD file good for? The OData service returns the found entity in the Atom format. However, SAP Cloud Platform Integration continues working with the result in the XML format. Hence, the runtime requires a description of the resulting XML message that fits the returned values of the newly configured query. As the Query Editor knows all of the details about the selected fields, as well as the format of

each field (thanks to the EDMX file), it can generate an associated XSD file representing the returned data of the service in XML format. In the end, this helps the engine map the returned Atom format to the XML format.

The generated **Resource Path** is added to the **Query Options** field channel's configuration. The completed configuration of the connection to the OData service should look like [Figure 4.51](#) for the **Processing** tab and [Figure 4.52](#) for the **Connection** tab.

The screenshot shows the 'OData' configuration interface with the 'Processing' tab selected. The 'Externalize' button is visible in the top right corner. The 'PROCESSING DETAILS' section contains fields for 'Operation Details' (set to 'Query(GET)'), 'Resource Path' (set to 'Orders'), and 'Query Options' (containing the query '\$select=OrderID,ShippedDate,ShipName,ShipAddress,ShipCity,ShipPostalCode,ShipCountry&\$filter=OrderID eq \${header:OrderNo}')'. Below these are sections for 'Custom Query Options', 'Page Size' (set to 200), and checkboxes for 'Process in Pages' and 'Timeout (in min)' (set to 1). The 'General' and 'Connection' tabs are also visible at the top.

Figure 4.51 Completed Processing Configuration of the ODC

The screenshot shows the 'OData' configuration interface with the 'Connection' tab selected. The 'Externalize' button is visible in the top right corner. The 'CONNECTION DETAILS' section contains fields for 'Address' (set to 'http://services.odata.org/Northwind/Northwind.svc'), 'Proxy Type' (set to 'Internet'), and 'Authentication' (set to 'None'). The 'General' and 'Processing' tabs are also visible at the top.

Figure 4.52 Completed Connection Configuration of the ODC

All that's left is to complete the integration flow itself by positioning the **Request-Reply** step in between the **Start** and **End** events. [Figure 4.53](#) shows the end result.

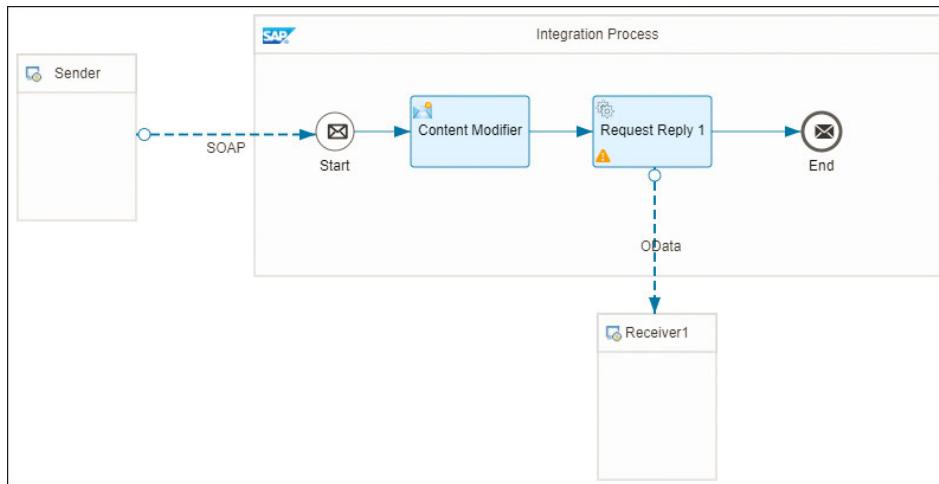


Figure 4.53 The Completed Configuration of the Integration Flow

Now, save and deploy your integration flow. After successful deployment, retrieve the integration flow's endpoint URL and invoke it via a SOAP client (with the order number 10249), such as SoapUI (see a detailed description of the invocation of integration flows in [Section 4.1.6](#)). As a result, you should see the response message displayed in [Figure 4.54](#).

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Orders>
      <Order>
        <ShipPostalCode>44087</ShipPostalCode>
        <ShippedDate>1996-07-10T00:00:00.000</ShippedDate>
        <OrderID>10249</OrderID>
        <ShipCity>Münster</ShipCity>
        <ShipAddress>Luisenstr. 48</ShipAddress>
        <ShipCountry>Germany</ShipCountry>
        <ShipName>Toms Spezialitäten</ShipName>
      </Order>
    </Orders>
  </soap:Body>
</soap:Envelope>
```

Figure 4.54 Response Message from the OData Service Formatted in XML

4.3.5 Using the Content Enricher Step

An interesting variant on this scenario is the use of the **Content Enricher** step inside the route, instead of the **Request-Reply** step. Look at [Figure 4.40](#) again, which shows

the selection of **Request-Reply** from the palette. Below the **Request-Reply** entry, you'll find the **Content Enricher** step. In contrast to the **Request-Reply** activity, which simply returns the response message from the external data source to the caller, the **Content Enricher** merges the content of the returned external message with the original message. In other words, it converts the two separate messages into a single enhanced payload. You can try this scenario out yourself, as its configuration is pretty straightforward. All you need to do is replace the **Request-Reply** step with the **Content Enricher** step (Figure 4.55) and ensure that the OData adapter is selected.

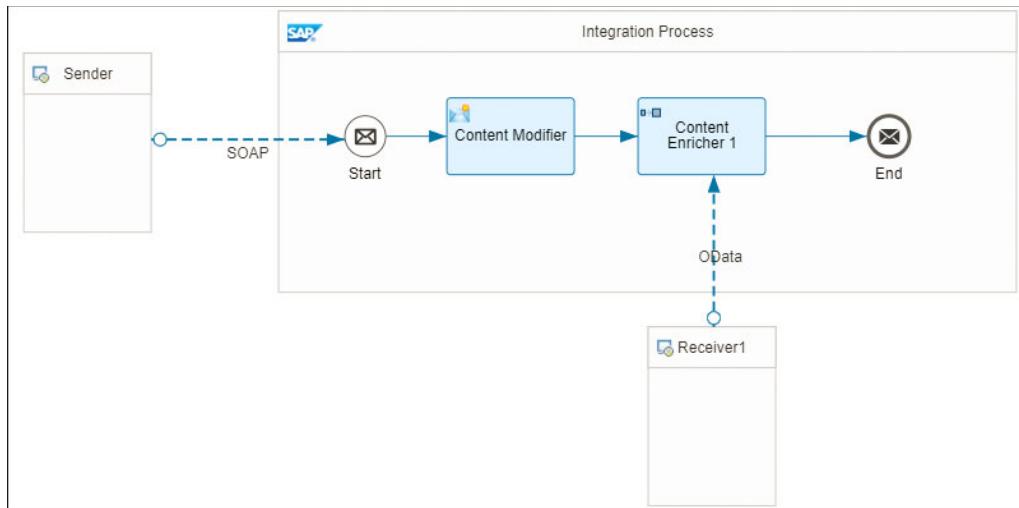


Figure 4.55 Replacing the Request-Reply Step with the Content Enricher Step

Note that there is one little, but very important, deviation from the configuration of the connection between the **Receiver1** pool and the **Content Enricher** activity: the arrow points from the **Receiver1** pool to the **Content Enricher** step. This is the opposite direction, as compared to the connection between the **Request-Reply** step and the **Receiver1** pool (refer to Figure 4.53 for a comparison). Ensure that you don't mix up the direction of the arrows. The **Content Enricher** step also includes a **Processing** tab with an **Aggregation Algorithm** field that should be left with its default value, namely **combine**. We'll revisit the **Aggregation Algorithm** field later in the section.

Once deployed, you can run the same input message against the newly configured integration flow. As a result, you'll see the output shown in Figure 4.56.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <multipmap:Messages xmlns:multipmap="http://sap.com/xi/XI/SplitAndMerge">
      <multipmap:Message1>
        <demo:OrderNumber_MT xmlns:demo="http://hci.sap.com/demo">
          <orderNumber>10249</orderNumber>
        </demo:OrderNumber_MT>
      </multipmap:Message1>
      <multipmap:Message2>
        <Orders>
          <Order>
            <ShipPostalCode>44087</ShipPostalCode>
            <ShippedDate>1996-07-10T00:00:00.000</ShippedDate>
            <OrderID>10249</OrderID>
            <ShipCity>Münster</ShipCity>
            <ShipAddress>Luisenstr. 48</ShipAddress>
            <ShipCountry>Germany</ShipCountry>
            <ShipName>Toms Spezialitäten</ShipName>
          </Order>
        </Orders>
      </multipmap:Message2>
    </multipmap:Messages>
  </soap:Body>
</soap:Envelope>

```

Figure 4.56 Output after Running the Integration Flow with the Content Enricher Step

You can identify the original input message at the top of the reply, followed by the returned detailed message coming from the external OData call. This allows you to choose the pattern that best fits your needs.

Let's now revisit the **Processing** tab of the **Content Enricher** step (see [Figure 4.57](#)).

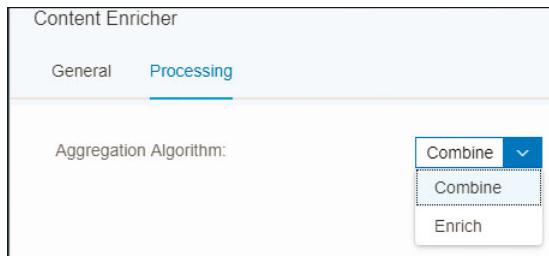


Figure 4.57 Processing Tab of the Content Enricher

As [Figure 4.57](#) shows, the **Content Enricher** step has two different aggregation algorithms that it uses to combine two payloads as a single message: the **Combine** and **Enrich** aggregation algorithms. The next sections discuss both algorithms.

Combine

The **Combine** aggregation algorithm simply creates a new target message by adding two original messages next to each other. You don't have control to define how messages should be combined. This is the algorithm used in our previous example, which produced the response message shown in [Figure 4.56](#). To better illustrate how the **Combine** algorithm differs from the **Enrich** algorithm, let's use a completely different example message (which doesn't relate to the example of [Figure 4.56](#)). Consider that you have an original message that looks like the one presented in [Figure 4.58](#).

```
<StudentCollection>
  <Student>
    <id>100</id>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
    <score_refid>a11</score_refid>
  </Student>
  <Student>
    <id>101</id>
    <firstname>Satoshi</firstname>
    <lastname>Nakamoto</lastname>
    <score_refid>a12</score_refid>
  </Student>
</StudentCollection>
```

Figure 4.58 Example of the Original Message

In addition, consider that the message presented in [Figure 4.59](#) represents the response of the service called by the **Content Enricher** step. Note that this message is also sometimes referred to as a lookup message within the context of the **Content Enricher** step.

```
<StudentScores>
  <Score>
    <id>a11</id>
    <course>Mathematics</course>
    <score>70</score>
    <year>2018</year>
    <passed>true</passed>
  </Score>
  <Score>
    <id>a12</id>
    <course>Mathematics</course>
    <score>40</score>
    <year>2018</year>
    <passed>false</passed>
  </Score>
</StudentScores>
```

Figure 4.59 Example of the Response Message of the Content Enricher Step (Lookup Message)

Using the **Content Enricher** algorithm, the combination of the two messages shown in [Figure 4.58](#) and [Figure 4.59](#) results in a message similar to the one presented in [Figure 4.60](#). Notice from [Figure 4.60](#) that the two messages have been each wrapped by the nodes `message1` and `message2`, respectively. Another node named `messages` is used as the root of the message.

```

<multimap:messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
  <message1>
    <StudentCollection>
      <Student>
        <id>100</id>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
        <score_refid>all</score_refid>
      </Student>
      <Student>
        <id>101</id>
        <firstname>Satoshi</firstname>
        <lastname>Nakamoto</lastname>
        <score_refid>a12</score_refid>
      </Student>
    </StudentCollection>
  </message1>
  <message2>
    <StudentScores>
      <Score>
        <id>all</id>
        <course>Mathematics</course>
        <score>70</score>
        <year>2018</year>
        <passed>true</passed>
      </Score>
      <Score>
        <id>a12</id>
        <course>Mathematics</course>
        <score>40</score>
        <year>2018</year>
        <passed>false</passed>
      </Score>
    </StudentScores>
  </message2>
</multimap:messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">

```

Figure 4.60 Enriched Message Using the Content Enricher Combined Algorithm

Let's next look at how the result would have looked with the **Enrich** aggregation algorithm.

Enrich

The **Enrich** algorithm is used to provide you more control of how the original message and the one returned by the **Content Enricher** (lookup message) should be

merged. You can specify the path to the node and key element based on which the original message is enriched with the lookup message.

Looking at the message presented earlier in [Figure 4.58](#), and note that the element <score_refid> represents the reference ID to be used as a link to the lookup message (refer to [Figure 4.59](#)). While using the **Enrich** algorithm, you need a way to specify how the correct record should be retrieved from the lookup message. This is done with the help of the fields shown in [Table 4.2](#). Note that these fields appear on the screen as soon as you select the **Enrich** value from the **Aggregator Algorithm** dropdown (see [Figure 4.61](#)).

Category	Field	Description	Example
Original Message	Path to Node	Path to the reference node in the original message	StudentCollection/Student
	Key Element	Key element in the original message	score_refid
Lookup Message	Path to Node	Path to the reference node in the lookup message	StudentScores/Score
	Key Element	Key element in the lookup message	id

Table 4.2 Properties Available for the Enrich Aggregator Algorithm

The screenshot shows the 'Content Enricher' configuration screen. At the top, there are tabs for 'General' and 'Processing', with 'Processing' being the active tab. To the right, there is a button labeled 'Externalize'. Below the tabs, the 'Aggregation Algorithm' is set to 'Enrich'. The configuration is divided into two main sections: 'ORIGINAL MESSAGE' and 'LOOKUP MESSAGE'. In the 'ORIGINAL MESSAGE' section, there are two required fields: '*Path to Node' and '*Key Element', both of which are highlighted with red boxes. In the 'LOOKUP MESSAGE' section, there are also two required fields: '*Path to Node' and '*Key Element', also highlighted with red boxes.

Figure 4.61 Example of Fields Required for the Enrich Algorithm

If you configure the **Enrich** properties with the values used in [Table 4.2](#) (Example column), you'll get a result similar to the one depicted in [Figure 4.62](#).

```
<StudentCollection>
  <Student>
    <id>100</id>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
    <score_refid>all</score_refid>
    <Score>
      <id>all</id>
      <course>Mathematics</course>
      <score>70</score>
      <year>2018</year>
      <passed>true</passed>
    </Score>
  </Student>
  <Student>
    <id>101</id>
    <firstname>Satoshi</firstname>
    <lastname>Nakamoto</lastname>
    <score_refid>a12</score_refid>
    <Score>
      <id>a12</id>
      <course>Mathematics</course>
      <score>40</score>
      <year>2018</year>
      <passed>false</passed>
    </Score>
  </Student>
</StudentCollection>
```

Figure 4.62 Enriched Message Using the Content Enricher's Enrich Algorithm

Notice how the resulting message of using the **Enrich** algorithm (see [Figure 4.62](#)) is different from the one using the **Combine** algorithm (refer to [Figure 4.60](#)). This closes our discussion on using the **Content Enricher** step.

In this section, we showed you how to configure the connection to an external OData-based service provider, how to benefit from the Query Editor to construct the sometimes complex string for the resource path to actually retrieve the data you're interested in, and how to add a **Request-Reply** (or **Content Enricher**) step to your message processing pipeline. In the next section, we'll explain how to actually format the message to a target format of your choice by using mappings.

4.4 Working with Mappings

Typically, integration projects require mapping functionality due to the many different message formats in the systems that need to be connected. Hence, integration

solutions such as SAP Cloud Platform Integration must provide capabilities to solve this problem elegantly. In this section, you'll learn how to apply mappings in your integration flows. We'll explain the configuration of the mapping step in detail. In addition, we'll show you how SAP Cloud Platform Integration message processing differs from message processing within SAP Process Integration, and what consequences this has on the mapping's configuration.

In every integration scenario, mapping between different data formats of the participating systems is a hot topic. In fact, it may be one of the most important tasks you have to complete for every integration project, and it always requires a certain amount of effort to implement. Supporting the modeler with a convenient mapping environment and a performance mapping engine is of the highest importance for every integration framework.

SAP gained some experience in building a mapping engine through its SAP Process Integration product. From the beginning, SAP Process Integration included a Java-based mapping engine. Consequently, it became an obvious choice to include the same mapping engine in SAP Cloud Platform Integration, as well. It's a stable and reliable engine that has been in productive use for many years. In addition, because of this reuse, you can also reuse your SAP Process Integration mappings in SAP Cloud Platform Integration, which saves your investment in developed mapping logic.

This section, however, isn't about the mapping engine itself, the functionality it provides, or how to solve certain mapping challenges because a plethora of material is already publicly available, either in SAP's online documentation for SAP Process Integration, or in SAP Community. Instead, we'll address the question of how to apply the mapping engine within SAP Cloud Platform Integration so that you know how to use a mapping step in your message processing chain correctly.

This is of particular importance because SAP Cloud Platform Integration is based on the Apache Camel integration framework, which can handle almost any message format—it's a payload-agnostic routing and mediation engine. Apache Camel doesn't follow the interface concept from SAP Process Integration, where you have to precisely define XML-based inbound and outbound interfaces in the Enterprise Services Repository (ESR) before you can actually start modeling an integration scenario.

The overhead of defining interfaces prior to modeling the integration isn't necessary for SAP Cloud Platform Integration. You can push anything to SAP Cloud Platform Integration, and, as long as you don't need access to the actual payload (e.g., for routing purposes), SAP Cloud Platform Integration forwards the message to the receivers

as-is. However, the mapping engine has the same history as SAP Process Integration with its XML background. Therefore, it works on XML transformations only, which requires a conversion to XML before invoking the mapping engine in SAP Cloud Platform Integration. As such, you must provide the XML schema of the source and the target message by uploading respective XSD or WSDL files to SAP Cloud Platform Integration. Let's see what this looks like in a concrete example.

4.4.1 The Scenario

For this section, we continue from where we left off in [Section 4.3](#). In that section, you learned how to invoke an OData service to retrieve order details for a given order number. The scenario is shown in [Figure 4.53](#).

As part of the configuration of the OData connection to the receiver, the Query Editor was used to model the access to the OData source. The editor allows you, for example, to define the entities for which a query should search, the individual fields of the entity you're interested in, the filter criteria of the query, and the order by condition for sorting the retrieved entities accordingly.

One important part of configuring the connectivity to the OData service provider by the Query Editor is the automatic generation of an XSD file representing the return message of the service in XML format. (See the confirmation message in SAP Cloud Platform Integration's graphical modeling environment after finishing the configuration of the connection's properties with the Query Editor in [Figure 4.50](#).)

This occurs even though an OData service typically returns its results in either JavaScript Object Notation (JSON) or Atom format. Obviously, the **Request-Reply** step of [Figure 4.53](#) implicitly invokes a mapping from JSON/Atom to XML so that the message processing chain can continue working on XML in the forthcoming steps. But what if the current format within the route isn't in XML? In these cases, you have to explicitly call a transformation step. Currently, SAP Cloud Platform Integration supports steps for converting comma-separated values (CSV) and JSON to XML, and vice versa. SAP Cloud Platform, Enterprise Edition, is additionally equipped with an Electronic Data Interchange (EDI)-to-XML converter. In the web-based modeling environment of SAP Cloud Platform Integration, you find the respective converter beneath the **Message Transformers** icon  ([Figure 4.63](#)).

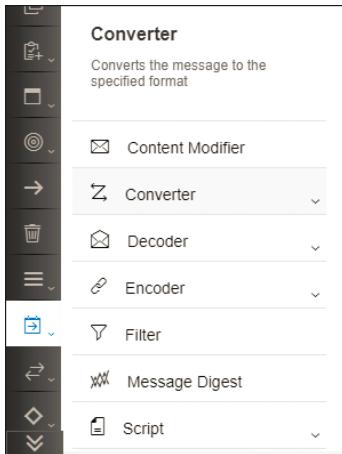


Figure 4.63 Pick a Converter Step from the Palette

Our goal for our example scenario is to map the returned entity into an XML format of our choice. The business scenario behind this assumption is that, quite frequently, data needs to be returned to a consumer in a specific XML format. You have to convert the automatically generated XML format of the **Request-Reply** step in your route to the target format the consumer expects.

Typically, the required target format is defined either by an XSD or a WSDL file. If you have an SAP Process Integration background, you can also easily define those files using the Enterprise Services Builder—the tool SAP Process Integration provides for designing interfaces in the ESR. Any other XML tool works as well. For our example, we've created such an XML file, called *GetOrderShipDetails_Sync.wsdl*, which we used for our first exercise in [Section 4.1](#). For your convenience, you'll find an example message following the WSDL's description in [Figure 4.64](#).

```
<soapenv:Envelope xmlns:soapenv="http://s
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4.64 Example Message

To summarize, the source message for the mapping step is the structure defined in the automatically generated XSD file *OrdersEntityGET0.xsd* (refer to [Figure 4.50](#)), and

the target message is the response part of the synchronous service interface described in *GetOrderShipDetails_Sync.wsdl*.

Before diving deeper into the mapping topic, let's first import the *GetOrderShipDetails_Sync.wsdl* WSDL file into our integration flow via the **Resources** view in the next section.

4.4.2 Adding and Using Resources via the Resources View

A typical integration scenario uses a variety of resources, such as mappings, archives, scripts, and schemas, in its steps. Any external file or artifact that can be imported or referred to for the purpose of being used in the integration flow can be considered as a resource. Let's first start by exploring the features of the integration flow. Later in the section, we'll discuss how to extend our example integration flow by adding a WSDL file using the **Resources** view.

The resources are typically grouped under the different categories listed in Table 4.3.

Category	Type	Supported Extensions	Source
Archives	Archive	.jar	Can be added from the file system
Mappings	Operation mapping	.opmap	Can be added from the ESR
	Message mapping	. mmap	
	XSLT mapping	.xslt .xsl	Can be added from the file system or another integration flow
Schemas	WSDL	.wsdl	Can be added from the file system or another integration flow
	XSD	.xsd	
	EDMX	.edmx	
Scripts	Groovy script	.gsh .gy .groovy	Can be added from the file system or another integration flow
	JavaScript	.js	

Table 4.3 Types of Files That the Resources View Supports

The columns used in [Table 4.3](#) are explained here:

- **Category**

A grouping under which the different resources can be classified.

- **Type**

The types of resource files that can be added for each category.

- **Supported Extensions**

The supported file extensions for each resource type.

- **Source**

The source from where the resources can be added. Some resources can be uploaded from your local file system. It's also possible to add some resources from another existing integration flow. This reuse of resources from another integration flow prevents the duplication of resources.

The **Resources** view is a location from which all of the artifacts listed in [Table 4.3](#) can be centrally managed. From this view, resources can be opened in their respective editors, downloaded, and deleted. After opening an integration flow, click on the area outside the integration flow model for the **Resources** view to appear at the bottom tabs, as shown in [Figure 4.65](#).

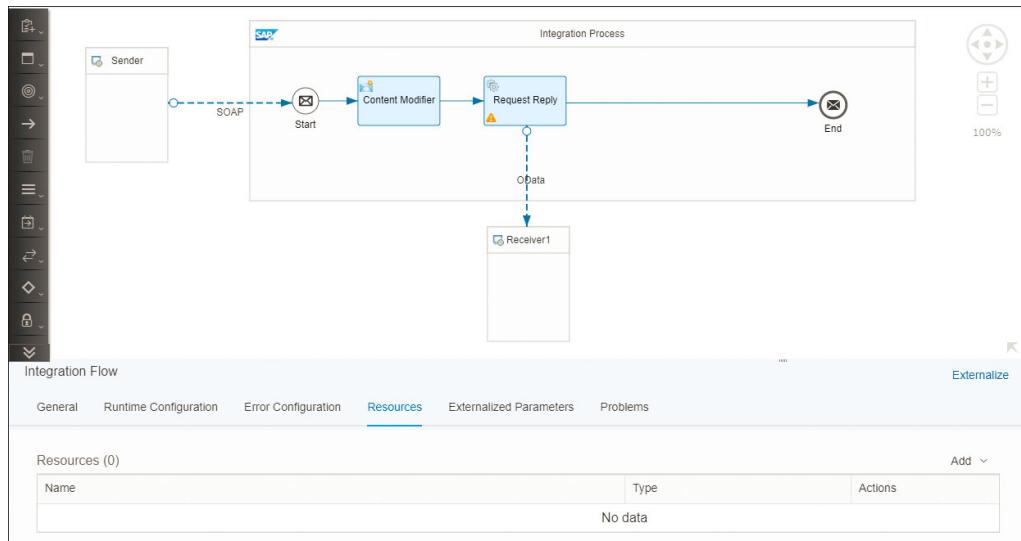


Figure 4.65 The Resources View Tab

Looking at the scenario we've been implementing in [Section 4.1.4](#), the integration flow needs to be enriched by adding the WSDL resource that will be used to perform some mapping later in [Section 4.4.3](#). In [Section 4.1.4](#), we downloaded the WSDL file named *GetOrderShipDetails_Sync.wsdl*. You can now import it by following these steps:

1. Click Edit, and then click the area outside the integration flow model.
2. Select the **Resources** tab at the bottom section of the integration flow (refer to [Figure 4.65](#)).
3. Click the **Add** button. A menu appears from which you need to choose **Schema** and then **WSDL**, as shown in [Figure 4.66](#).

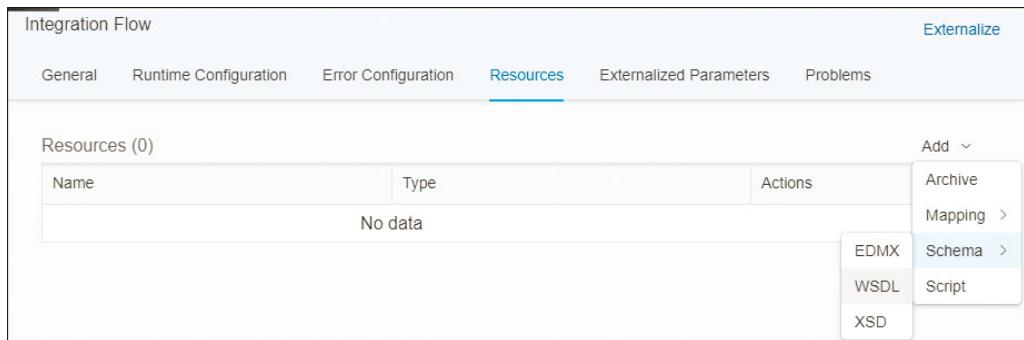


Figure 4.66 Adding an Object in the Resources View

4. You're redirected to a page from which you can browse to the WSDL on your local file system and upload it. As a result, the WSDL file is now added to the **Resources** tab, as shown in [Figure 4.67](#).

Resources (1)		
Name	Type	Actions
▼ Schemas (1)		
GetOrderShipDetails_Sync	WSDL	

Figure 4.67 Resource View after Adding a WSDL

Note

Because the WSDL can be categorized as a schema (as discussed in [Table 4.3](#)), note that it's placed under the **Schemas** node element in [Figure 4.67](#).

You can also hover over the name of a resource shown in the **Resources** View to view its access path, or you can click on it to view the resource details via its editor.

A number of actions can be performed on the uploaded resources, including the following:

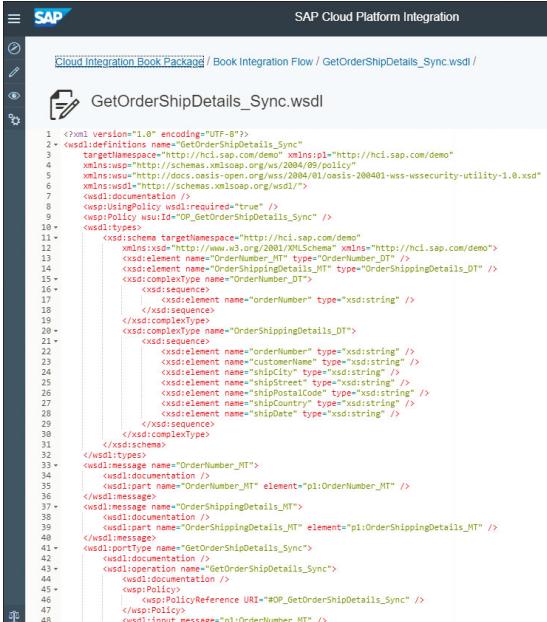
- **Delete**

As shown in [Figure 4.67](#), the delete icon  can be used to delete the resource from the **Resources** tab.

- **Download**

As shown in [Figure 4.67](#), the download icon  can be used to download the resource from the **Resources** tab.

By clicking on the name of the WSDL file resource in [Figure 4.67](#), the WSDL file is loaded in the special view, as shown in [Figure 4.68](#).



```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="GetOrderShipDetails_Sync"
  targetNamespace="http://hci.sap.com/demo" xmlns:p1="http://hci.sap.com/demo"
  xmlns:wsdl="http://schemas.xmlsoap.org/ws/2004/09/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:osis="2004/01/osis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsdl1="http://schemas.xmlsoap.org/wsdl"/>
<wsdl:documentation />
<wsdl:policy wsdl:required="true" />
<wsdl:policy wsdl:id="#P_GetOrderShipDetails_Sync" />
<wsdl:types>
  <xsd:schema targetNamespace="http://hci.sap.com/demo" />
  <xsd:element name="OrderNumber_MT" type="OrderNumber_DT" />
  <xsd:complexType name="OrderShippingDetails_MT" type="OrderShippingDetails_DT" />
  <xsd:complexType name="OrderNumber_DT">
    <xsd:sequence>
      <xsd:element name="orderNumber" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="OrderShippingDetails_DT">
    <xsd:sequence>
      <xsd:element name="orderNumber" type="xsd:string" />
      <xsd:element name="customerName" type="xsd:string" />
      <xsd:element name="customerCity" type="xsd:string" />
      <xsd:element name="shipStreet" type="xsd:string" />
      <xsd:element name="shipPostalCode" type="xsd:string" />
      <xsd:element name="shipCountry" type="xsd:string" />
      <xsd:element name="shipDate" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:message name="OrderNumber_MT">
  <wsdl:documentation />
  <wsdl:part name="OrderNumber_MT" element="p1:OrderNumber_MT" />
</wsdl:message>
<wsdl:message name="OrderShippingDetails_MT">
  <wsdl:documentation />
  <wsdl:part name="OrderShippingDetails_MT" element="p1:OrderShippingDetails_MT" />
</wsdl:message>
<wsdl:portType name="GetOrderShipDetails_Sync">
  <wsdl:documentation />
  <wsdl:operation name="GetOrderShipDetails_Sync">
    <wsdl:documentation />
    <wsdl:policyReference URI="#P_GetOrderShipDetails_Sync" />
    <wsdl:input message="p1:OrderNumber_MT" />
  </wsdl:operation>
</wsdl:portType>

```

Figure 4.68 Viewing the Contents of the WSDL File

Now that you've successfully imported the WSDL file using the **Resources** view, let's see how this WSDL file can be useful to your integration flow by using it in the sender channel. You'll need to select the sender channel, which displays its details at the bottom of the screen (see [Figure 4.69](#)).

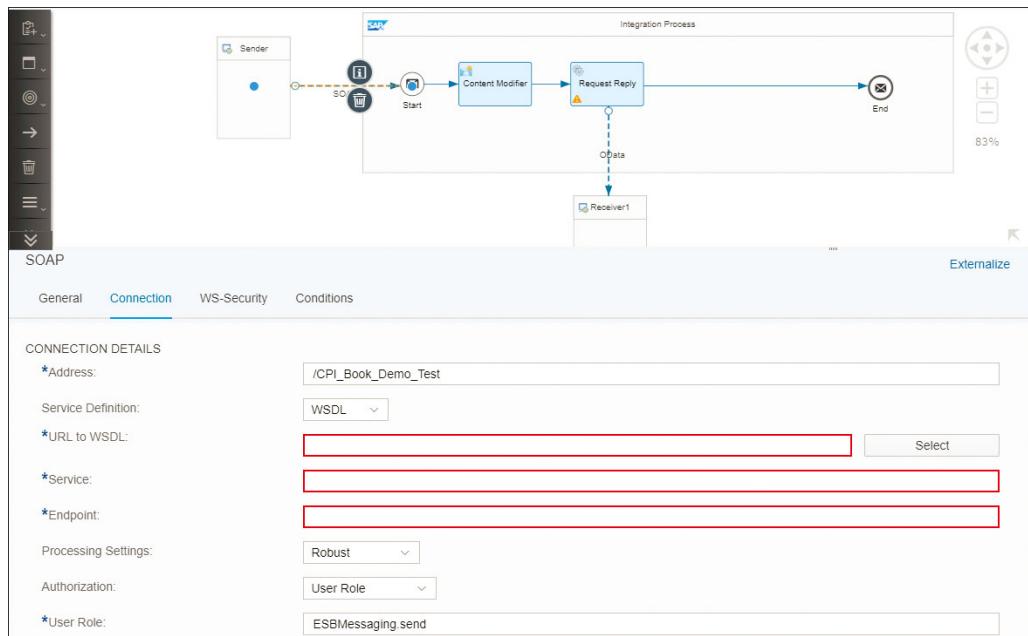


Figure 4.69 Adding a WSDL File to the Sender Channel

Then click on the **Select** button ([Figure 4.69](#)) to be presented with a new screen giving you the possibility to select a WSDL file (see [Figure 4.70](#)). Then select the **GetOrderShipDetails_Sync** row.



Figure 4.70 Selecting the WSDL File to Be Used in the Sender Channel

The screen is now populated with details of the WSDL file (see [Figure 4.71](#)).

SOAP		Externalize	
General	Connection	WS-Security	Conditions
*Address:	/CPI_Book_Demo_Test		
Service Definition:	WSDL		
*URL to WSDL:	/wsdl/GetOrderShipDetails_Synd.wsdl		Select
*Service:	p1:GetOrderShipDetailsService		
*Endpoint:	p1:GetOrderShipDetailsServiceSoap		
Processing Settings:	Robust		
Authorization:	User Role		
*User Role:	ESBMessaging.send		

Figure 4.71 Sender Channel Populated with WSDL Details

You can now save and deploy your integration flow. We'll discuss the mapping step next.

4.4.3 Applying the Mapping Step in the Message Processing Chain

To use the mapping engine in the route, select the **Mapping** palette entry to open the submenu, as shown earlier in Figure 65. You first have to position the mapping step in the integration flow pool. In the palette, a dedicated mapping icon is available ([Figure 4.72](#)).

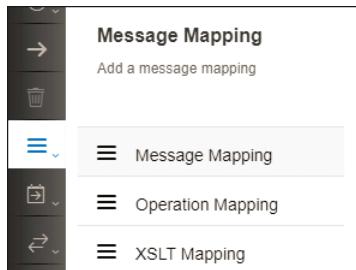


Figure 4.72 Select the Mapping Step from the Palette

Click the icon and then move the mouse pointer (which changed to three parallel horizontal bars) into the **Integration Process** pool ([Figure 4.73](#)). Then select the step from the integration flow and click on the **+** icon to create a message mapping ([Figure 4.74](#)). By doing this, you have to name the message mapping, as shown in [Figure 4.75](#), and then click on the **Create** button.

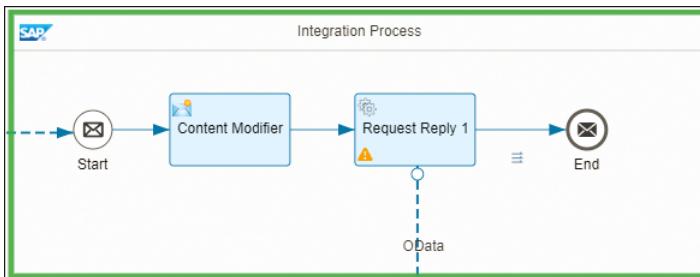


Figure 4.73 Positioning the Mapping Step in the Integration Flow (Mouse Pointer Changed to Three Horizontal Bars)

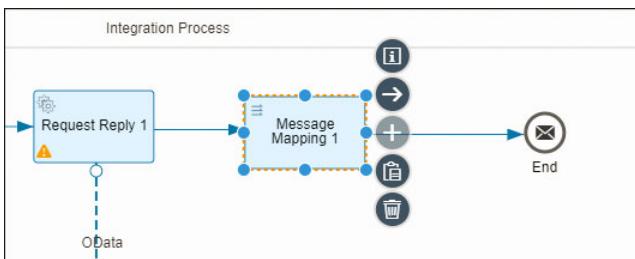


Figure 4.74 Creating a New Message Mapping

Create Message Mapping	
Name:	<input type="text" value="OData2XML"/>
Create Cancel	

Figure 4.75 Naming the Message Mapping

After clicking on the **Create** button, the mapping editor opens immediately ([Figure 4.76](#)).

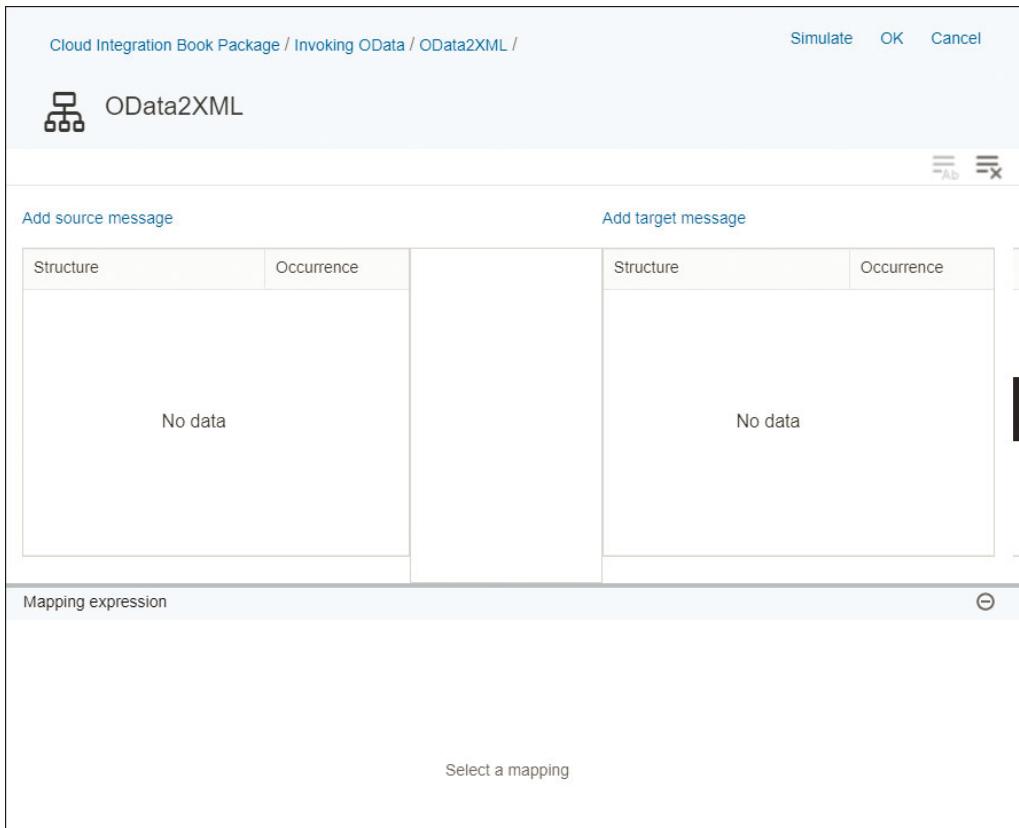


Figure 4.76 Mapping Editor Opens after Positioning the Mapping Task

As you can see, both the source and the target messages are missing. Next, we describe how to use either a pregenerated XSD file or a self-developed WSDL file for the definition of source and target messages. We'll start with the assignment of the source message. Click the **Add source message** link in Figure 4.76 to opens the **Select source message** dialog box shown in Figure 4.77.

The dialog box lists all the files that either have already been generated or have been uploaded from the file system. In our example, the file describing the OData service, as well as the file that has been automatically generated by the Query Editor, are listed. You know from the summary of the previous section that *OrdersEntity-GETO.xsd* is the file that describes the source message. Click it once. You'll immediately return to the mapping editor, which shows the **Structure** of the source message on the left side of the screen (Figure 4.78).

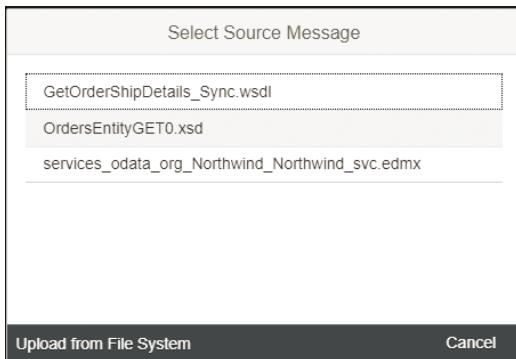


Figure 4.77 Dialog Box for Selecting the Target Message

A screenshot of the Mapping Editor. On the left, under the heading "Orders", there is a tree view of the source message structure: "Orders" (1..1), "Order" (1..*), "ShippedDate" (0..*), "OrderID" (1..*), "ShipCountry" (0..*), "ShipPostalCode" (0..*), "ShipCity" (0..*), and "ShipAddress" (0..*). To the right, there is a table with columns "Structure" and "Occurrence", which is currently empty. A red dashed box highlights the first row of this table. In the top right corner, there is a link labeled "Add target message". At the bottom, there is a section labeled "Mapping expression".

Figure 4.78 Mapping Editor Displaying the Structure of the Source Message after Assigning the XSD File

Next, assign the target message by clicking the **Add target message** link in the upper-right corner of the mapping editor. Again, the dialog box for assigning the appropriate file opens ([Figure 4.79](#)).

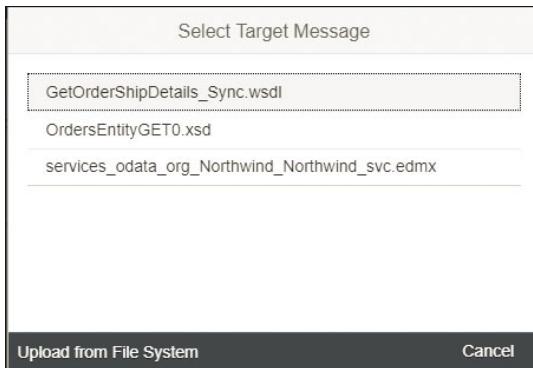


Figure 4.79 Selecting the Target Message's Structure

Note that the WSDL that we need is already available (*GetOrderShipDetails_Sync.wsdl*) because we already imported it via the **Resources** view in [Section 4.4.2](#). As the WSDL file contains the description of a synchronous interface, another dialog box for selecting an element opens ([Figure 4.80](#)).

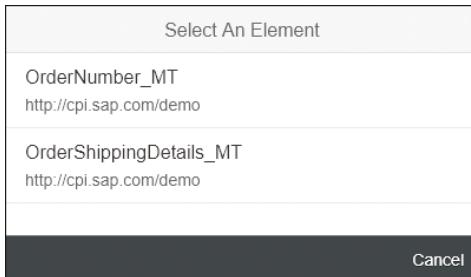


Figure 4.80 Selecting the Correct Target Structure from a Synchronous Interface

This second dialog appears because synchronous interfaces consist of two parts: a request and a response. Typically, these are two completely different structures.

Take our scenario as an example: We're retrieving details for an order based on an order number coming in via the request message, so the request message contains just one field, the order number. The response message, on the other hand, contains the details of the order. Hence, it comprises a number of fields. This is exactly what you see in [Figure 4.80](#). The first entry, **OrderNumber_MT**, reflects the request message containing just the order number as the only field. The second entry, **OrderShippingDetails_MT**, stands for the response message and contains all details that make up the order.

Note

SAP Process Integration specialists will recognize the message type (MT) suffix (the string _MT after the data type's name) for both elements. It's a typical habit if you create message types in the ESR. And, yes, we've created the synchronous interface in the ESR. As such, the interoperability between SAP Process Integration and SAP Cloud Platform Integration on a data level is ensured.

You must pick the second entry from the list as your target message structure. Click it once, and the user interface (UI) immediately returns to the mapping editor. The mapping editor analyzes the structure of the response message and depicts it accordingly. [Figure 4.81](#) shows the result on the right side of the screen.

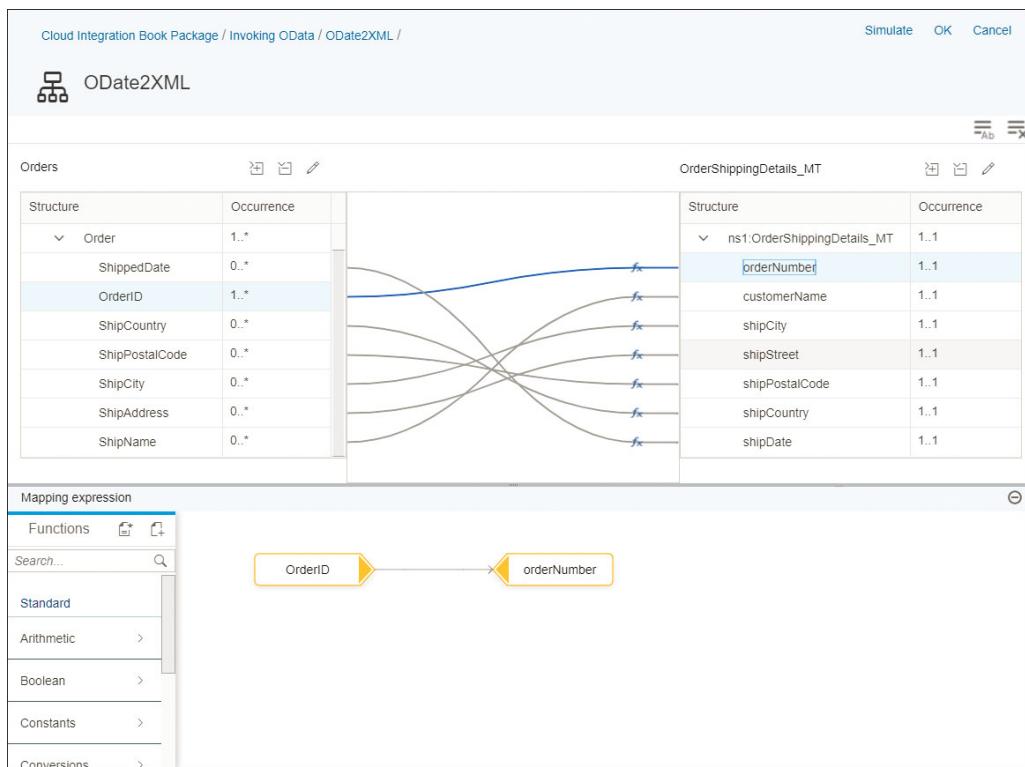


Figure 4.81 The Mapping Editor after Opening Both Structures and after Finishing the Mapping Exercise

The mapping editor itself should look familiar to those who have worked with SAP Process Integration before. You can now assign field mappings by simply dragging and dropping source fields from the left to the corresponding target fields on the right. We did this for our simple scenario, and the result is also depicted in [Figure 4.81](#).

You can also benefit from several predefined functions that are shipped with the mapping editor and that are listed in the **Functions** area of [Figure 4.81](#).

If you're dealing with a complex mapping, and the predefined functions don't have what you need, you also have the choice to build your own custom-built Java functions using user-defined functions (UDFs). UDFs are discussed in [Chapter 9, Section 9.3](#). An alternative is to build an Extensible Stylesheet Language Transformation (XSLT) mapping.

As we've already explained, you'll find plenty of material in SAP Community and in the SAP Process Integration online help (https://help.sap.com/viewer/product/SAP_NETWEAVER_PI/ALL/en-US) when it comes to mappings that are more complex. You'll find the handling of the graphical mapper quite intuitive and convenient. It shouldn't be a problem to define the mapping between the two structures as it's shown in the figure.

Notice the **Simulate** button in the top-right corner of the mapping editor. You can immediately check your mapping for accuracy by uploading an example message to the mapping editor, which, in turn, runs your mapping and displays the result. This results in short turnaround cycles between development and test.

After you're done with your mappings, click the **Ok** button (also located in the top-right corner of the mapping editor), and you're back in the UI for modeling your integration flow.

Finally, drag the mapping step on the arrow connecting the **Request-Reply** step with the **Message-End** event shown earlier in [Figure 4.53](#) to position the mapping activity between the two. The result of your process model is shown in [Figure 4.82](#).

That's it! You've finished the configuration of the mapping activity and can now save, deploy, and run your new integration flow. Use a SOAP tool of your choice (e.g., SoapUI), and invoke your solution. An appropriate request message might look like the one shown in [Figure 4.83](#).

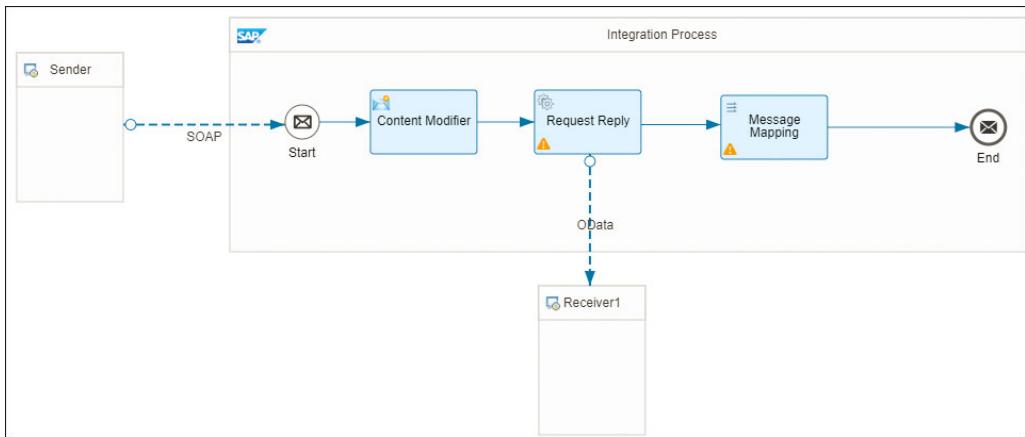


Figure 4.82 The Final Result of the Process Model after Positioning the Mapping Step before the End Event

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>

```

Figure 4.83 Example Request Message

After passing through the integration flow, the result message should look similar to the one shown in [Figure 4.84](#).

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <ns1:OrderShippingDetails_MT xmlns:ns1="http://hci.sap.com/demo">
      <orderNumber>10249</orderNumber>
      <customerName>Toms Spezialitäten</customerName>
      <shipCity>Münster</shipCity>
      <shipStreet>Luisenstr. 48</shipStreet>
      <shipPostalCode>44087</shipPostalCode>
      <shipCountry>Germany</shipCountry>
      <shipDate>1996-07-10T00:00:00.000Z</shipDate>
    </ns1:OrderShippingDetails_MT>
  </soap:Body>
</soap:Envelope>

```

Figure 4.84 Reply Message after Successful Invocation of the Integration Flow, Including the Mapping Step

To summarize, mappings are an important aspect in every integration project. SAP Cloud Platform Integration benefits from a mapping engine originally developed for SAP Process Integration. You've learned how the mapping engine is integrated into SAP Cloud Platform Integration and how you can invoke it in your integration flows by applying the mapping activity. You've also seen how the message processing in SAP Cloud Platform Integration differs from SAP Process Integration (the payload-agnostic behavior in SAP Cloud Platform Integration) and that this difference requires the explicit definition of message structures by using either XSD or WSDL files to configure the mapping step. With this knowledge, you're now well equipped to tackle even more sophisticated integration challenges.

4.4.4 Using Value Mapping to Enhance Your Scenario

Value mappings are used to represent multiple values of an object. To better illustrate it, let's imagine that the consumer of the service who receives the response returned in [Figure 4.84](#) doesn't accept country names in the `shipCountry` field but rather prefers to have a country ISO code. This means, for example, that instead of returning the value `Germany`, the consumer would rather receive the value `DE`. This is a common use case in integration scenarios and means that the object country can be represented in many ways (following our case here, it can be represented using a country name or an ISO code). It's in such cases where value mapping comes to the rescue.

Let's now try to extend the example of [Figure 4.84](#) to return an ISO code in the field `shipCountry` instead of the country name. To achieve that, you'll need to create a **Value Mapping** artifact by following these steps:

1. Go to the **Design** section of SAP Cloud Platform Integration, and open your package in edit mode.
2. From your package, click on the **Add** button, and select **Value Mapping** from the resulting menu ([Figure 4.85](#)).
3. Provide a name for the value mapping, and click **Ok** ([Figure 4.86](#)).
4. Navigate to the right package, and open the value mapping that you just created in the previous step. You end up with a value mapping editor as shown in [Figure 4.87](#).
5. Click on **Edit** to switch to an editable mode.

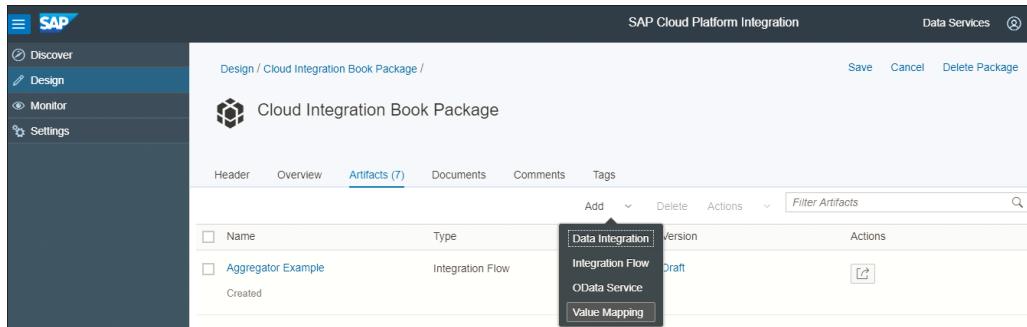


Figure 4.85 Creating a Value Mapping Artifact

This is a dialog box titled 'Add value mapping artifact'. It has two radio button options: 'Create' (selected) and 'Upload'. The 'Name' field contains 'CountryNameToCountryCode'. The 'ID' field also contains 'CountryNameToCountryCode'. Below these fields is a large text area labeled '<Description>'. Underneath the description area are 'Source:' and 'Target:' fields, both containing '<Source>' and '<Target>' respectively. At the bottom right are 'OK' and 'Cancel' buttons.

Figure 4.86 Naming the Value Mapping Artifact

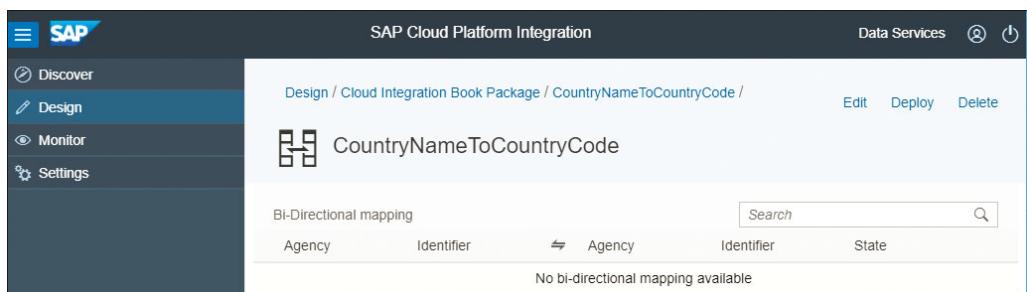


Figure 4.87 Value Mapping Editor

6. Click on the **Add** button in [Figure 4.88](#). You're then presented with a list of fields that are explained in [Table 4.4](#). For the sake of simplicity, fill in the columns as shown in [Figure 4.88](#).

Figure 4.88 Adding Values to the Value Mapping

Column Name	Description
Agency	Represents the organization/scheme responsible for managing and issuing an identifier, for example, a company. Agency A is a country that is responsible for issuing the identifier “passport number” as a means to identify a person. At the same time, we have Agency B representing a company that identifies the same person using the identifier “employee number.”
Identifier	A unique value issued by an agency, for example, a passport number.

Table 4.4 Columns of Value Mapping

7. It's now time to add a list of values for the source and target systems to provide a mapping of values between source and target. Click on the **Add** button at the bottom of the screen shown earlier in [Figure 4.88](#).
8. You then get another screen similar to the one presented in [Figure 4.89](#). After populating the value mapping with all required country names and country code values, click on the **Save** button to persist the changes.
9. Click on **Deploy** in the top -right corner of [Figure 4.89](#).

The screenshot shows the SAP Cloud Integration Value Mapping editor. At the top, there's a header with 'Design / Cloud Integration Book Package / CountryNameToCountryCode /' and buttons for 'Save', 'Deploy', 'Cancel', and 'Delete'. Below the header, the title 'CountryNameToCountryCode' is displayed. Underneath, a section titled 'Bi-Directional mapping' lists mappings between 'Agency' and 'Identifier'. The mappings are: AgencyA / CountryName ↔ AgencyB / Identifier. Below this, there are tabs for 'Value Mappings' (which is selected) and 'Default Values'. The 'Value Mappings' table shows three entries:

AgencyA, CountryName	↔	AgencyB, CountryCode
Germany	↔	DE
France	↔	FR
Netherlands	↔	NL

On the right side of the table, there's usage information: 'ValueMap (Source agency, Source identifier, Source value, Target agency, Target identifier) = Target value;' and examples: 'ValueMap (AgencyA, CountryName, Germany, AgencyB, CountryCode) = DE;' and 'ValueMap (AgencyB, CountryCode, DE, AgencyA, CountryName) = Germany;'. There are also 'Add' and 'Delete all' buttons.

Figure 4.89 Populating the Value Mapping

After the value mapping is deployed, it's time to use it in our mapping. For that, we need to open the mapping that we developed in [Figure 4.65](#) (refer to [Figure 4.81](#)).

To get back to that mapping, follow these steps:

1. Open the integration flow, click the **Edit** button, and select the **Mapping** step ([Figure 4.90](#)). The property of the mapping appears on the bottom section.
2. Select the **Processing** tab, and click on the mapping name (in our case **/ODate2-XML.mmap**) next to the resource field to navigate to the mapping editor.

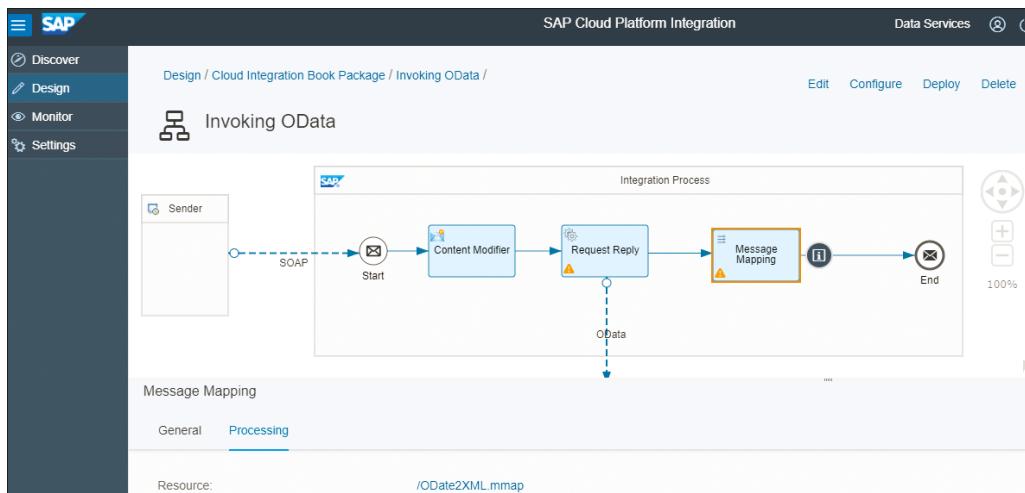


Figure 4.90 Overview of the Integration Flow

3. When you're redirected to the mapping editor, select the **shipCountry** field on the target structure (right side).
4. Select the **valueMapping** function under the **Conversions** function category (bottom-left corner of Figure 4.91).

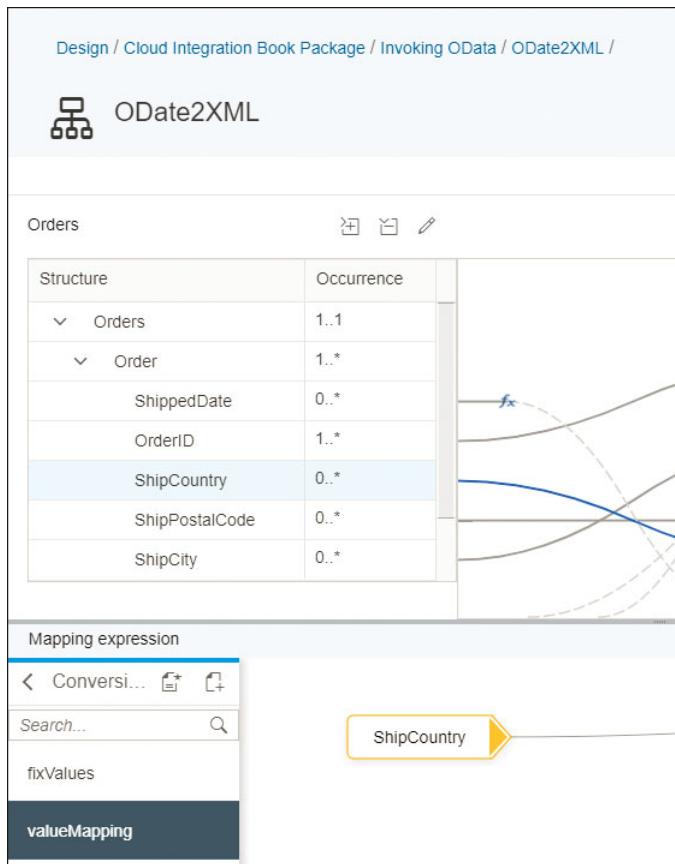


Figure 4.91 Adding a Value Mapping Function in the Mapping Logic

5. Place the **valueMapping** function on the screen. For the input fields **Source Agency**, **Source Identifier**, **Target Agency**, and **Target Identifier**, use the same values as you specified earlier in Figure 4.89. At the end, the configuration should look like Figure 4.92.

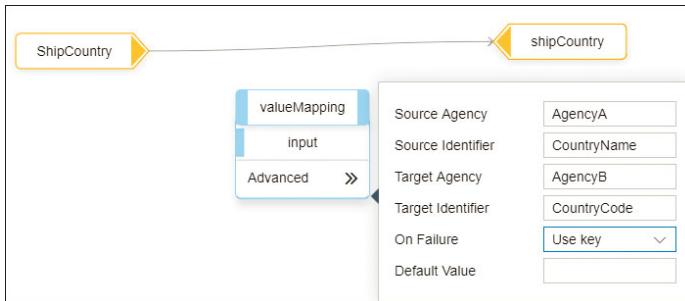


Figure 4.92 Configuring the Value Mapping Function

Note

By using the **Default Value** field in [Figure 4.92](#), it's possible to specify a default value to be used if no matches are found for the incoming value. But for that to work, you'll need to set the **On Failure** dropdown option to **Use Default Value**.

Following is the full list of possible values for the **On Failure** dropdown:

- **Use Key**

If there is no match, the incoming value will simply be passed to the target value without translation.

- **Throw exceptions**

If there is no match, an exception will be thrown by the mapping framework.

- **Use Default Value**

If there is no match, the default value (specified in the **Default Value** field) will be assigned to the target value.

6. After configuring the value mapping function, place it between the two **ShipCountry** fields to get to the final result, which is similar to [Figure 4.93](#).

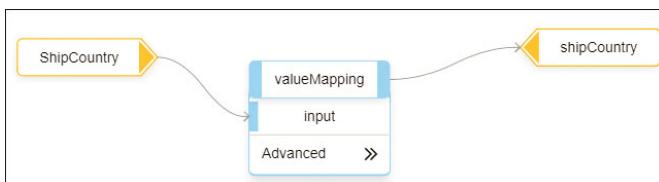


Figure 4.93 Performing a Value Mapping Translation

7. Save and deploy the integration flow.

Congratulations! You can now retest the service using SoapUI. The returned response is shown in [Figure 4.94](#). Note that the value of ShipCountry is no longer Germany, but DE, as configured in our value mapping.



```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <nsl:OrderShippingDetails_MT xmlns:nsl="http://hci.sap.com/demo">
      <orderNumber>10249</orderNumber>
      <customerName>Toms Spezialitäten</customerName>
      <shipCity>Münster</shipCity>
      <shipStreet>Luisenstr. 48</shipStreet>
      <shipPostalCode>44087</shipPostalCode>
      <shipCountry>DE</shipCountry>
      <shipDate>1996-07-10T00:00:00.000</shipDate>
    </nsl:OrderShippingDetails_MT>
  </soap:Body>
</soap:Envelope>
```

Figure 4.94 Updated Response of the Service after Using Value Mapping

4.5 Defining and Providing an OData Service

In [Section 4.3](#), we explored how to consume an external OData service from SAP Cloud Platform Integration. In this case, we used a receiver OData adapter to call the external OData service. There might also be cases where you, as an integration developer, are asked to provide an OData service in SAP Cloud Platform Integration. This will require using a sender OData adapter.

Next, let's explore how you can provide such an OData service from SAP Cloud Platform. Let's start by presenting the target scenario that will be used to explore the OData provisioning capabilities.

4.5.1 The Target Scenario

The integration flow in [Figure 4.95](#) depicts the integration scenario we're going to build in the next section.

We're going to create an OData service that calls a SOAP web service. For simplicity, we'll use a publicly available SOAP web service provided by <http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso>. This is just for illustration's sake, but you can use any web service of your choice.

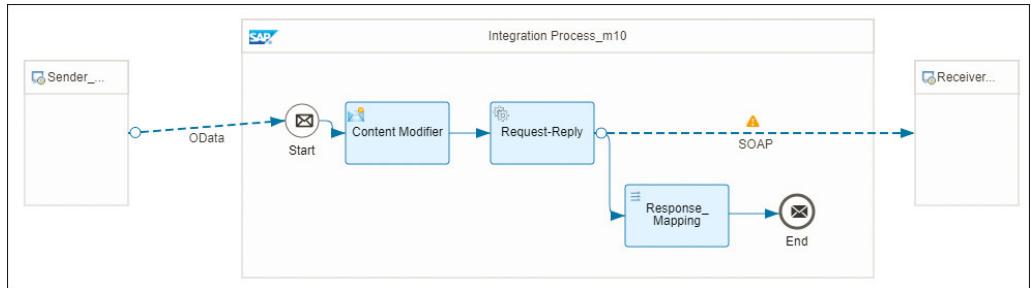


Figure 4.95 OData Integration Flow

One of the first steps is to download the suggested WSDL file available at <http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL> and save it to our local computer. Let's now dive into the details of how to provide such an OData service.

4.5.2 Providing an OData Service

SAP Cloud Platform facilitates the provision of an OData service with a powerful wizard that leads to an automatic generation of an integration flow in the background. While using the wizard, the following main steps are involved:

- **Import from Data Source**
Helps in creating or updating an OData model by importing its definition from a SOAP, OData, or ODC data source.
- **Edit OData Model**
Enables you to create or update an OData model using the OData Model Editor.
- **Bind to Data Source**
Supports in binding entity sets and function imports to a data source such as SOAP, OData, or ODC endpoints.
- **Edit Integration Flow**
Changes and customizes the generated integration flows with additional business logic.
- **Deploy OData Service**
Deploys and starts using your OData service when it's ready.

Before exploring each one of the preceding steps in detail, let's first create the needed project by following these steps:

1. From the **Design** tab of your SAP Cloud Platform Integration tenant (left side of [Figure 4.96](#)), go to the respective package, and switch to the edit mode.
2. Select the **Add** button, and then click **OData Service** from the resulting menu (see [Figure 4.96](#)).

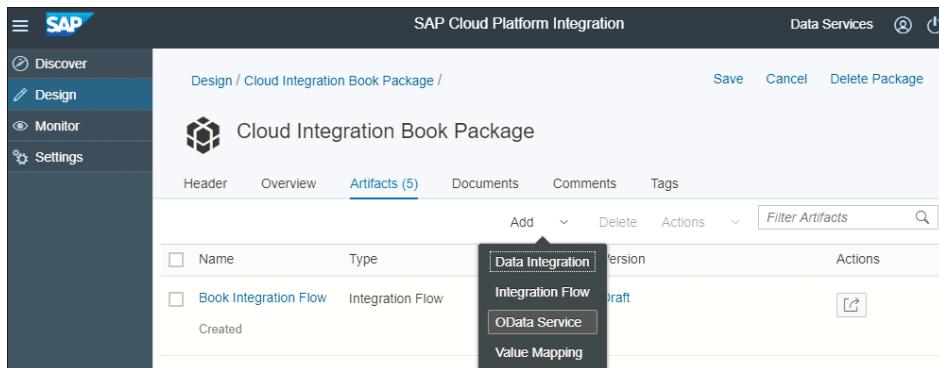


Figure 4.96 Creating an OData Service Artifact

3. Give the artifact a suitable name, such as “Country OData Service”, and leave the other fields with their default values. Note that, by default, the namespace is set with the value **SAP** (see [Figure 4.97](#)). Then select the **OK** button.

The dialog box is titled 'Add OData Service Artifact'. It has two radio buttons: 'Create' (selected) and 'Upload'. Required fields are marked with asterisks (*). The fields are:

- *Name: 'Country Odata Service'
- *Namespace: 'SAP'
- *ID: 'Country_Odata_Service_SAP_1'
- Description: A text area containing '<Description>'.
- *OData Version: A dropdown menu showing 'OData V2'.

 At the bottom are 'OK' and 'Cancel' buttons.

Figure 4.97 Details of the OData Service Artifact

You then get redirected to a page that looks like [Figure 4.98](#) and [Figure 4.99](#). This is the landing page and the main place to perform all tasks related to defining and provisioning an OData Service.

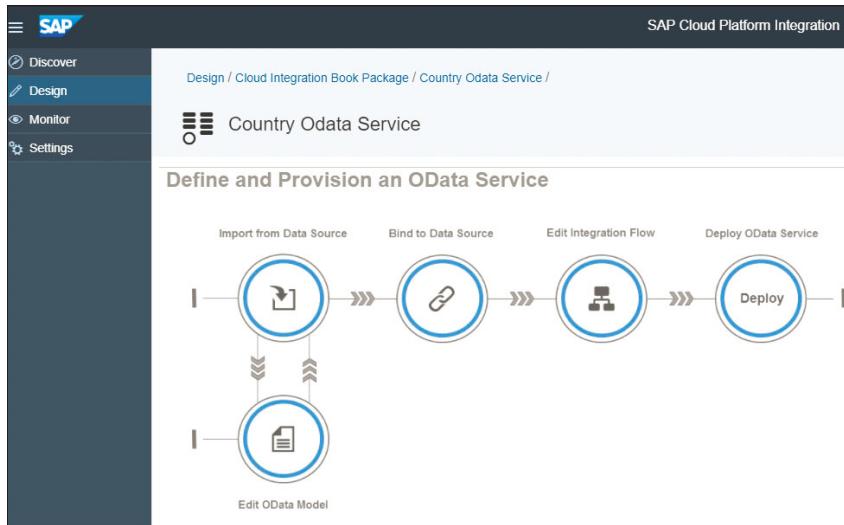


Figure 4.98 Main Landing Page to Define and Provision an OData Service A

This screenshot shows a different view of the SAP Cloud Platform Integration interface. The top bar includes "Data Services", a user icon, and a power button. Below are buttons for "Save", "Save as version", "Deploy", "Cancel", and "Delete". A "Import Model Wizard" button is also present. On the left, there's a "Learn How To" section with links to "Import from Data Source", "Edit OData Model", "Bind to Data Source", "Edit Integration Flow", and "Deploy OData Service". Each link has a brief description below it. The overall layout is more focused on specific service management tasks.

Figure 4.99 Main Landing Page to Define and Provision an OData Service B

As [Figure 4.98](#) and [Figure 4.99](#) show, this page lists the activities that need to be performed for the definition of an OData service. These activities were also listed in the beginning of [Section 4.5.2](#), and we're going to explore them in detail in the next sections.

Import from Data Source

The first step is to import the definition of an existing service (data source) to be used as the basis for our OData model. When this book was written, you could import definitions from a SOAP, OData, or ODC (provided by SAP Gateway) data source. For simplicity, let's illustrate the provisioning of an OData service using the import of an existing SOAP web service—the WSDL that we downloaded in [Section 4.5.1](#). Proceed as follows to import a WSDL data source:

1. Click the import model wizard button  shown earlier on the right-top corner of [Figure 4.99](#).
2. Select **SOAP** from the **Data Source Type** dropdown, and click on the **Browse** button to select the WSDL file that you downloaded in [Section 4.5.1](#) ([Figure 4.100](#)).
3. Click on the **Step 2** button to proceed with the wizard.

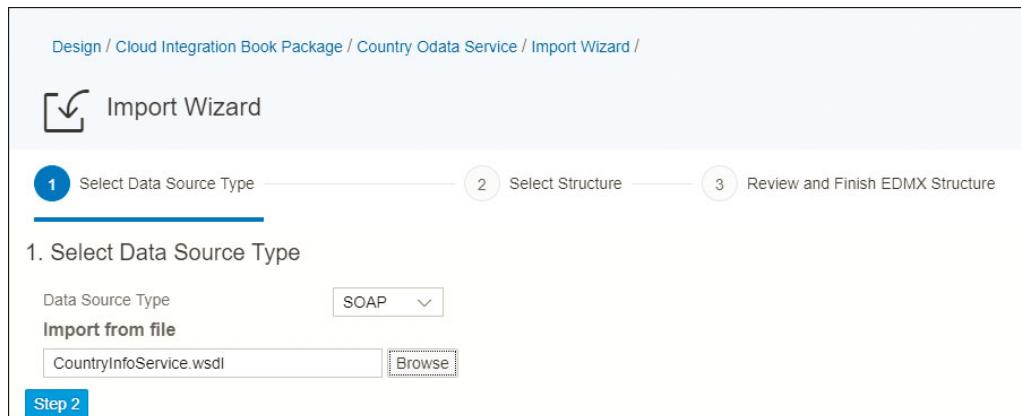


Figure 4.100 Importing a WSDL Data Source

4. Select the needed element from the data source. Note that we're going to use the `CountryISOCode` operation. Therefore, select its request and response structures (see [Figure 4.101](#)).
5. Click on the **Step 3** button.

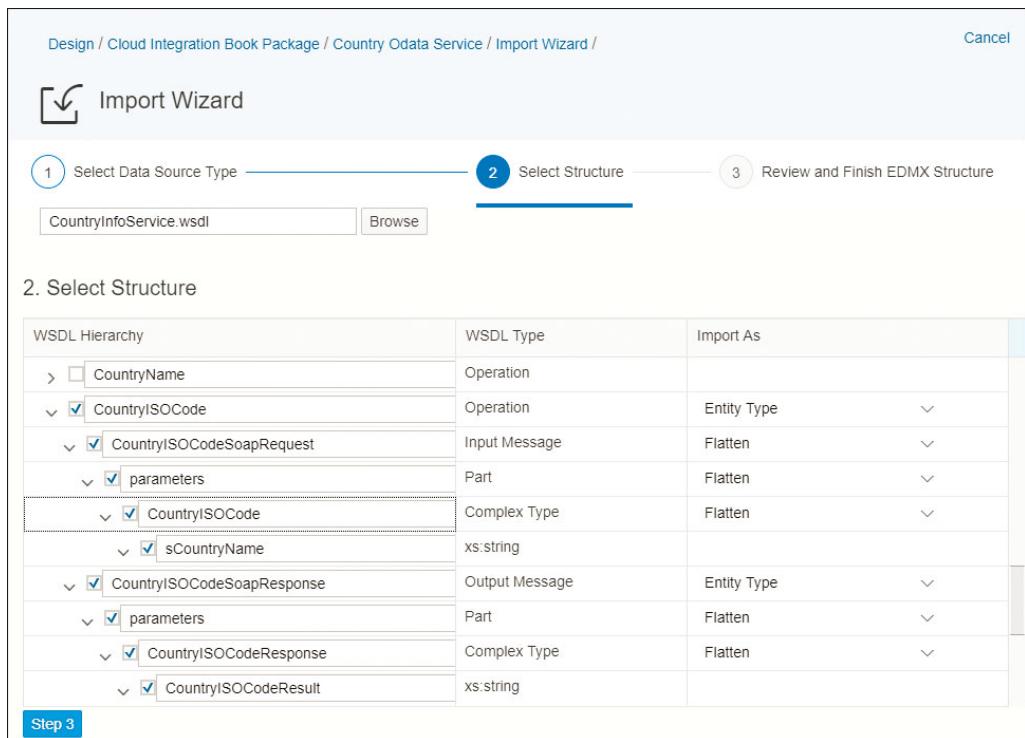


Figure 4.101 Selecting the Desired Request and Response Structures from the Data Source

6. You're redirected to the next page in the wizard where you can review the generated EDMX structure. You can change the names or EDM type of any element and add comments and descriptions for each element using the **Documentation** field (Figure 4.102). Be aware that it's mandatory to select one of the elements as the primary key. When done, click on the **Finish** button.

Note

In the step presented in Figure 4.102, it's mandatory to select a primary key that represents a unique identifier for the OData model. Without selecting a primary key, you're confronted with an error, and it's not possible to proceed to the next screen of the wizard.

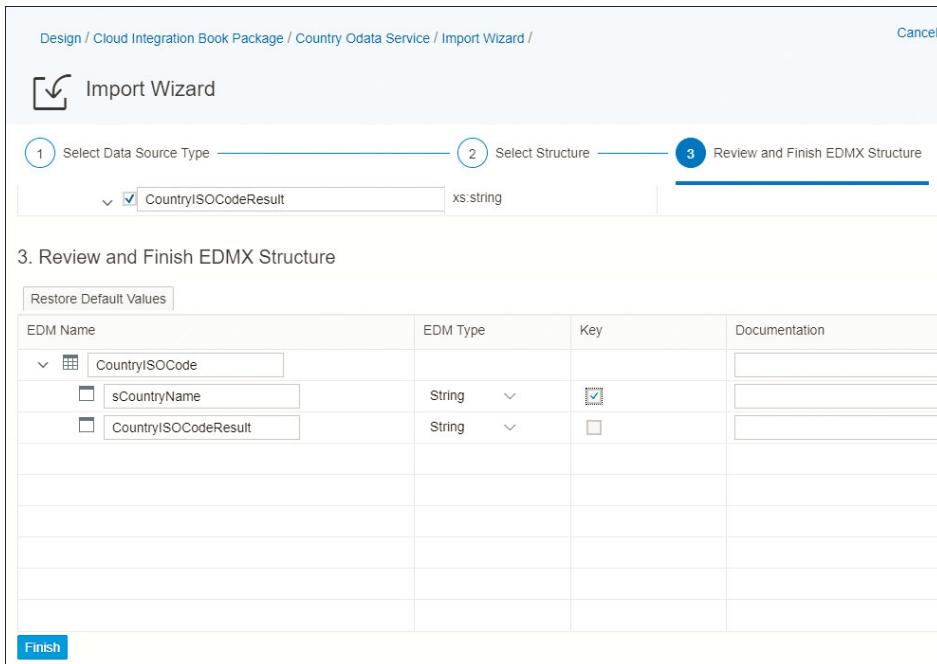


Figure 4.102 Reviewing the EDMX Structure

- Now that you've imported the WSDL model, you're redirected to the **Define and Provision an OData Service** view (see [Figure 4.103](#)). This view presents the different operations possible in an OData Service, including **Query**, **Create**, **Read**, **Update**, and **Delete**, as shown in [Figure 4.103](#).

The screenshot shows the 'Define and Provision an OData Service' view for the 'Country Odata Service'. At the top, there are buttons for 'Save' (highlighted), 'Save as version', 'Deploy', 'Cancel', and 'Delete'. Below is a table:

Name	Data Source	Details	Action
CountryISOCodeSet	SOAP		
Query			
Create			
Read			
Update			
Delete			

Figure 4.103 Define and Provision an OData Service View

Let's now explore how to edit the OData Model generated in the preceding steps.

Edit the OData Model

After importing a model based on a WSDL data source as discussed in the previous section, let's look at how to modify the generated OData model. This can be achieved by first returning to the **Define and Provision an OData Service** view (refer to [Figure 4.103](#)) and following these steps:

1. From the right corner of the screen shown previously in [Figure 4.103](#), click on the OData Model Editor button . This opens the OData Model editor (see [Figure 4.104](#)).
2. If you're comfortable with OData models, you can edit and change the model to fit your requirements. The editor makes use of an auto-complete function to help you while editing the model. You need to press **[Ctrl]+[Spacebar]** to get a suggestion, as shown in [Figure 4.104](#). Then save the final result using the **Ok** button.

The screenshot shows the 'Edmx Editor' interface. At the top, there is a breadcrumb navigation: 'Cloud Integration Book Package / Country Odata Service / Edmx Editor /'. Below it is a title bar with a pencil icon and the text 'Edmx Editor'. The main area contains the XML code of an OData model, specifically the 'CountryISOCodeResult' entity type definition. A context menu is open over the XML code, listing options: 'Association', 'ComplexType', 'EntityContainer', 'EntityType', and 'Function'. The 'EntityType' option is highlighted. The XML code is as follows:

```

1 <edmx:Edmx
2   xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
3   xmlns:sap="http://www.sap.com/Protocols/SAPData" Version="1.0"
4   <edmx:DataServices
5     xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m:DataServiceVersion="2.0"
6     <Schema
7       xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="S1">
8
9   <EntityType Name="CountryISOCodeResult" Type="Edm.String" Nullable="false"></EntityType>
10  <EntityType Name="CountryISOCode" Type="Edm.String" Nullable="false"></EntityType>
11  <EntityType Name="Country" Type="Edm.String" Nullable="false"></EntityType>
12  <EntityType Name="CountryISOCodeResult" Type="Edm.String" Nullable="false"></EntityType>
13  <EntityType Name="CountryISOCode" Type="Edm.String" Nullable="false"></EntityType>
14  <EntityType Name="Country" Type="Edm.String" Nullable="false"></EntityType>
15  <EntityType Name="CountryISOCodeResult" Type="Edm.String" Nullable="false"></EntityType>
16  <EntityType Name="CountryISOCode" Type="Edm.String" Nullable="false"></EntityType>
17  <EntityType Name="Country" Type="Edm.String" Nullable="false"></EntityType>
18  <EntityType Name="CountryISOCodeResult" Type="Edm.String" Nullable="false"></EntityType>
19  <EntityType Name="CountryISOCode" Type="Edm.String" Nullable="false"></EntityType>
20  <EntityType Name="Country" Type="Edm.String" Nullable="false"></EntityType>
21  </Schema>
22 </edmx:DataServices>
</edmx:Edmx>

```

Figure 4.104 OData Model Editor

You can get back to the main page (refer to [Figure 4.103](#)) by clicking on the OData service name at the top of [Figure 4.104](#). In our example, click on **Country OData Service**.

Note that it's also possible to see a graphical representation of the generated model. The graphical representation can be accessed using the  button in the top-right corner of [Figure 4.103](#). You then get something similar to [Figure 4.105](#).

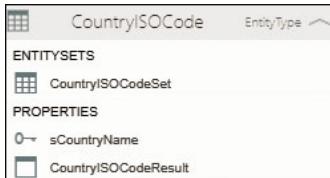


Figure 4.105 Graphical Model Viewer

Now that our model is ready, it's time to bind it to the SOAP web service.

Bind to the Data Source

Binding a model means that you need to link entity sets and function imports to a data source such as SOAP, REST, or OData endpoints. To perform the binding, first return to the **Define and Provision an OData Service** view. Then follow these steps:

1. Click on the bind button  on the same row as the desired operation. For now, let's do that for the **Query** operation (refer to [Figure 4.103](#)). You're then redirected to a page similar to [Figure 4.106](#).



Figure 4.106 Binding an Operation to a Data Source

2. From this page, you can specify the operation to which the **Query** operation should be bound. As [Figure 4.106](#) shows, choose **CountryISOCode** in the **Operation** dropdown of the SOAP service. Note that the **End Point** textbox is automatically filled in with the correct value.

- Click on the **Ok** button to return to the **Define and Provision an OData Service** view (see [Figure 4.107](#)). Note that during this step, an integration flow is automatically generated in the background.

Name	Data Source	Details	Action
CountryISOCodeSet	SOAP	CountryISOCode	
Query			
Create			
Read			
Update			
Delete			

Figure 4.107 Define and Provision an OData Service View after Binding an Operation

- Click on the **Save** button in [Figure 4.107](#) to persist your changes.
- Let's now navigate to the integration flow editor by clicking on the button on the same row as the **Query** operation (see [Figure 4.107](#)).

We're redirected to the integration flow editor. Now we'll enhance and change the generated integration flow in the next section.

Edit the Integration Flow

As depicted in [Figure 4.108](#), you're presented with the integration flow that was automatically generated based on the actions performed in the previous sections.

The generated integration flow has a basic setup with sender and receiver channels already configured automatically.

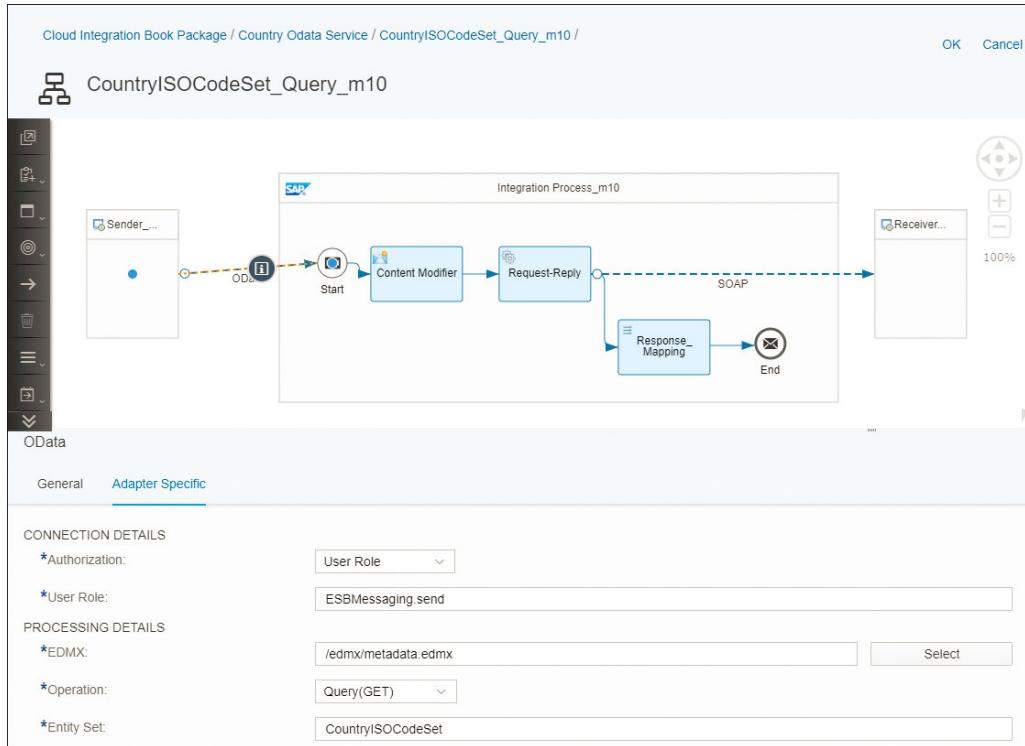


Figure 4.108 Generated Integration Flow for the OData Service

A look in the **Resources** view ([Figure 4.109](#)) shows that a number of artifacts are automatically included:

- **Resource_Mapping1**

The message mapping artifact used in the **Message Mapping** step named **Response_Mapping** (refer to [Figure 4.108](#)).

- **CountryInfoService**

WSDL of the Country Info SOAP service that we're consuming in our integration flow.

- **Error**

XSD representing an error, which can be used to throw an exception.

- **Metadata**

The EDM file that was automatically added by the OData service generation.

Integration Flow		
General	Runtime Configuration	Resources
Resources (4)		
Name	Type	Actions
Mappings (1)		
Response_Mapping1	Message Mapping	
Schemas (3)		
CountryInfoService	WSDL	
Error	XSD	
metadata	EDMX	

Figure 4.109 Overview of the Resources View

Let's now enhance the integration flow to suit our needs. A good start is to replace the **Content Modifier** with a **Message Mapping** step. We've already discussed how to use a **Message Mapping** step in [Section 4.4](#). As a result, we're not going to repeat every step here in detail. Proceed as follows:

1. After opening the integration flow, click on the **Edit** button.
2. Remove the **Content Modifier** step, and replace it with a **Message Mapping** step (see [Figure 4.110](#)).

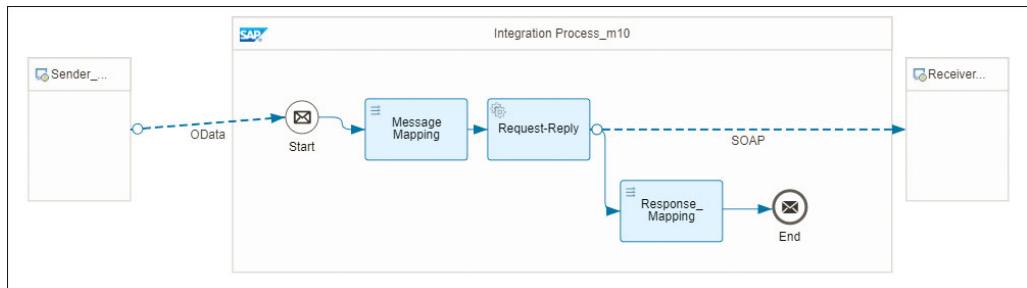


Figure 4.110 Integration Flow with Message Mapping Instead of Content Modifier

3. Create a message mapping, and assign the EDMX structure on the source message and the XSDL on the target message. In our case, the EDMX file is called *metadata.edmx*, and the WSDL file is called *CountryInfoService.wsdl*, as explained earlier and shown in [Figure 4.109](#). The end result of the newly created message mapping

is shown in [Figure 4.111](#). Note that in the target message, the `CountryISOCode` message structure needs to be selected.

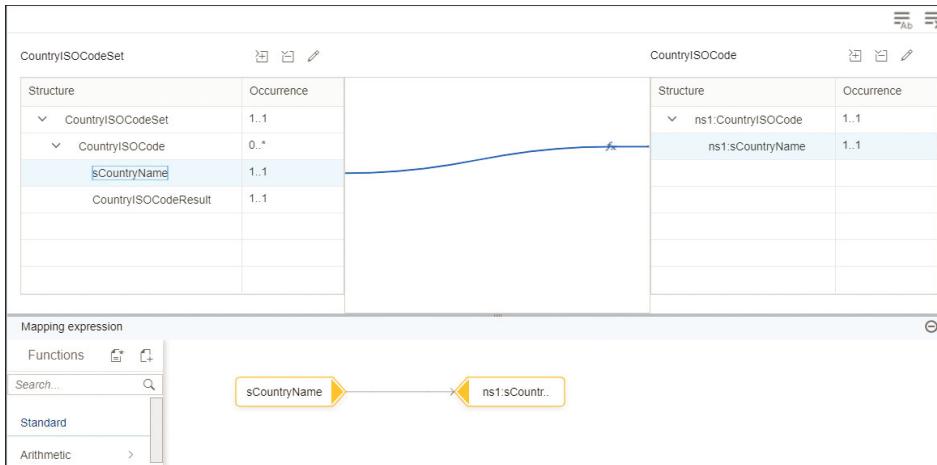


Figure 4.111 New Message Mapping

Finally, let's adjust the response mapping. As previously indicated, a response mapping called **Response_Mapping** (refer to [Figure 4.108](#)) was included in the **Resources** view. By default, there is no mapping logic defined in this message mapping.

[Figure 4.112](#) depicts an example of the mapping logic.

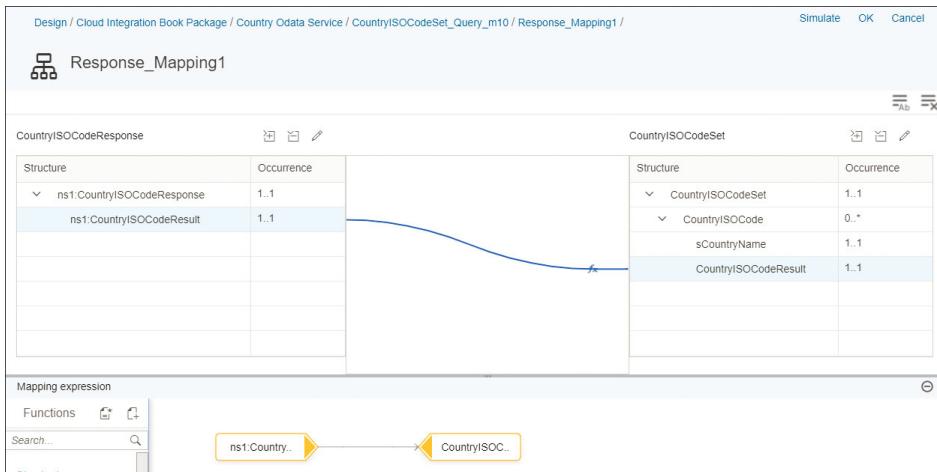


Figure 4.112 Defining the Response Mapping Logic

The integration flow is now enhanced and adjusted to suit our needs. Let's save it.

Note

At the time of authoring this book, the OData adapter only supports synchronous communication, which means that every request must have an associated response.

Deploy the OData Service

After the integration flow has been adjusted to your needs in the previous steps, the OData service can now be deployed. To do so, you'll first need to return to the **Define and Provision an OData Service** view (refer to [Figure 4.107](#)). Click on **Save** and then on **Deploy**. After a successful deployment, the OData Service should now be available in the **Manage Integration Content** section of the **Monitor** view, as shown in [Figure 4.113](#).

Name	Status
Country Odata Service	Started

Country Odata Service

Deployed On: Apr 21, 2018, 16:45:09 ID: Country_Odata_Service_SAP_1
Deployed By: S0011540061 Version: 1.0.0

Endpoints Status Details Artifact Details Log Configuration

There are no endpoints configured.

Status Details

The OData Service is deployed successfully.

Artifact Details

Monitor Message Processing
Name: COUNTRY_ODATA_SERVICE Namespace: SAP Version: 1

Figure 4.113 The Deployed OData Service from the Monitor

You can now call and test the newly provisioned OData service using the endpoint URL format:

`https://<IFLMAP_URL>/gw/odata/<OData_Service_Namespace>/<OData_Service_Name>;V=<Version>`

Note the following:

- <IFLMAP_URL> should be replaced with your runtime URL, for example, `https://<yourinstance>-iflmap.hci.eu1.hana.ondemand.com`.
- <OData_Service_Namespace> should be replaced by the value of the **Namespace** field found in the **Artifact Details** section shown in [Figure 4.113](#).
- <OData_Service_Name> should be replaced by the value of the **Name** field found in the **Artifact Details** section shown in [Figure 4.113](#).
- <Version> should be replaced by the value of the **Version** field found in the **Artifact Details** section shown in [Figure 4.113](#).

Using our example, the resulting endpoint will be similar to the following:

`https://<yourinstanc>-iflmap.hci.eu1.hana.ondemand.com/gw/odata/SAP/COUNTRY%20ODATA%20SERVICE;v=1`

Note

The spaces included in the service name (*COUNTRY ODATA SERVICE*) are replaced by %20.

Well done, this was the last step in defining and provisioning an OData service.

4.6 Working with an Aggregator

According to the Enterprise Integration Patterns, an aggregator pattern answers the question of how to combine the results of individual but related messages so that they can be processed as a whole.

As [Figure 4.114](#) depicts, imagine that you have a scenario involving system A, which needs to send an order message to system B. However, system A is capable of sending a message with a maximum of only 1 item at a time. This means that for system A to send an order with 10 items, it will need to send 10 different messages, each containing 1 item. For illustration's sake, let's assume that the incoming order item message looks like the one presented in [Figure 4.115](#).

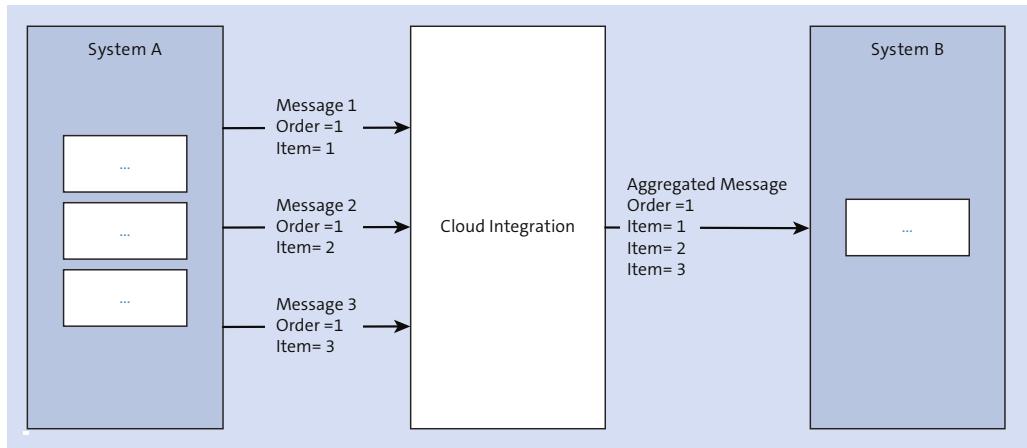


Figure 4.114 Sample Aggregator Situation

```
<OrderItem xmlns:demo="http://hci.sap.com/demo">
  <orderNumber>AA2345</orderNumber>
  <Item>
    <ItemNo>1</ItemNo>
    <Quantity>1</Quantity>
    <Unit>1</Unit>
    <LastStatus>false</LastStatus>
  </Item>
</OrderItem>
```

Figure 4.115 Sample Incoming Order Item Message

To solve this challenge, you'll need to use a stateful filter, also called an aggregator, to collect and store individual messages (each containing 1 item as shown in [Figure 4.115](#)) until a complete set of related messages has been received. Then, the aggregator sends a single message (with all 10 items) extracted from the individual messages.

When dealing with an aggregator, the following questions need to be addressed:

- How do we identify that an incoming message is related to another one?
- How do we identify that a complete set of messages has been received? In other words, how do we know that it's time to stop collecting incoming messages because we've received the last one?
- What do we do if the sender system never sends the last message?

Luckily, SAP Cloud Platform Integration provides an **Aggregator** step to combine multiple incoming messages into a single message. This step helps you solve the preceding challenges.

4.6.1 Sample Scenario

Let's use a sample scenario to illustrate the use of the **Aggregator** step. [Figure 4.116](#) depicts what the final integration flow looks like.

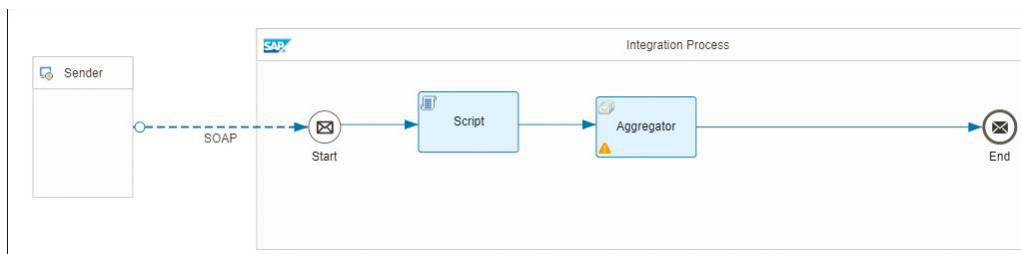


Figure 4.116 Sample Target Integration Flow

Note that in the sample integration flow presented in [Figure 4.116](#), there are no receiver systems. This is purely for illustration purposes. In a real-life scenario, you'll generally want to send the aggregated message to a receiver system. Furthermore, the **Message Exchange Pattern** field of the SOAP adapter is set to **One-Way** for this example ([Figure 4.117](#)). This setting ensures that the message is asynchronous. We'll discuss about how to deal with asynchronous processing in [Chapter 5](#).

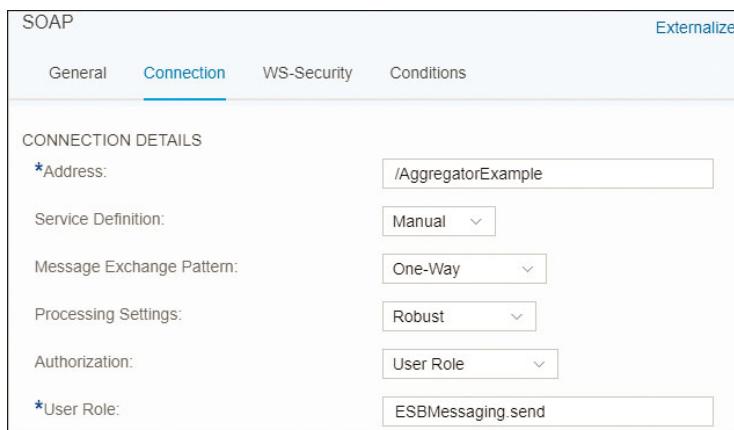


Figure 4.117 Settings of the Sender SOAP Channel

We won't spend time describing how to create the integration flow here because this has already been explained in the previous chapters. Instead, we'll mainly focus on the configuration of the aggregator step in SAP Cloud Platform Integration. Perform the following steps:

1. Navigate to the **Design** section of SAP Cloud Platform Integration, and select the **Aggregator** step from the palette, as shown in [Figure 4.118](#).
2. Drag the step to your integration flow.

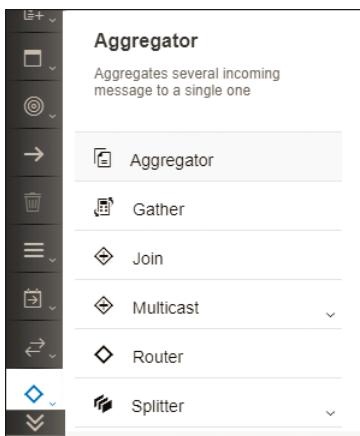


Figure 4.118 Aggregator Step in the Design Palette

3. Select the **Aggregator** step from the integration flow, and specify the different attribute properties (see [Figure 4.119](#) and [Figure 4.120](#)). The description and meaning of the attributes of an **Aggregator** step are specified in [Table 4.5](#).



Figure 4.119 Aggregator Correlation Tab

Aggregator

General Correlation **Aggregation Strategy**

*Incoming Format: XML (Same Format)

*Aggregation Algorithm: Combine in Sequence

*Message Sequence Expression (XPath): /OrderItem/Item/ItemNo

*Last Message Condition (XPath): /OrderItem/Item/LastStatus = 'true'

*Completion Timeout (in min): 5

*Data Store Name: Aggregator-1

Externalize

Figure 4.120 Aggregation Strategy Tab

Name	Description
Correlation Expression (XPath)	An XPath expression that points to an element to be used to match all correlated incoming messages. This field provides a solution related to the question we've asked in the beginning of the section regarding how to identify that an incoming message is related to another one.
Incoming Format	Specifies the content type of the incoming message. At the time of publishing, , only XML (Same Format) can be selected.
Aggregation Algorithm	Specifies how the correlated messages will be aggregated. Possible aggregation methods include the following: <ul style="list-style-type: none"> ■ Combine All incoming and correlated messages are aggregated using a random order. ■ Combine in Sequence All incoming and correlated messages are aggregated according to a sequence order defined by the Message Sequence Expression (XPath) field.
Message Sequence Expression (XPath)	Specifies the sequencing order by which the messages need to be sorted during the aggregation. Note that this field is only present if the Aggregation Algorithm dropdown is set to Combine in Sequence .

Table 4.5 Attributes of the Aggregator Step

Name	Description
Last Message Condition (XPath)	An XPath value to specify how to identify the last message to be aggregated. This field provides a solution related to the question regarding how to identify that a complete set of messages has been received or how to know that it's time to stop collecting incoming messages because the last one has been received.
Completion Timeout (in min)	Specifies the maximum time between the processing of two messages before the aggregation is automatically stopped. The default value is set to 60 minutes. This field provides a solution related to the question about what to do if the sender system never sends the last message. Specifying a timeout ensures that the process doesn't wait forever for incoming messages. However, it goes without saying that the value to be used in this field needs to be agreed upon with the sender system. Messages arriving after the completion timeout has elapsed will start a new message processing.
Data Store Name	Specifies the name of the temporary data store to be used for storing the aggregated message. By default, SAP Cloud Platform Integration specifies a randomly generated name, but it can be changed to any other unique name. Note that it uses only local data stores. Global data stores currently aren't supported. See Chapter 5, Section 5.4.1 , where the data store is explained.

Table 4.5 Attributes of the Aggregator Step (Cont.)

After you finish building the integration flow that includes the **Aggregator** step, save and deploy it.

4.6.2 Sending Messages via SoapUI

Let's now run the scenario by sending messages via SoapUI and observing how our integration flow behaves in the SAP Cloud Platform Integration monitor. The first message to be triggered is shown in [Figure 4.121](#).

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <OrderItem xmlns:demo="http://hci.sap.com/demo">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>1</ItemNo>
        <Quantity>1</Quantity>
        <Unit>1</Unit>
        <LastStatus>false</LastStatus>
      </Item>
    </OrderItem>
  </soapenv:Body>
</soapenv:Envelope>

```

Figure 4.121 First Message for the Aggregation Scenario

Note that the example message in [Figure 4.121](#) uses the correlation (`orderNumber`) AA2345 and sequence number (`ItemNo`) value 1.

After triggering the message, look in the **Monitor Message Processing** section of SAP Cloud Platform Integration. As [Figure 4.122](#) depicts, notice that two new log entries are added to the monitor.

The screenshot shows the SAP Cloud Platform Integration interface for monitoring message processing. At the top, there are filters for Time (Custom, from Apr 22, 2018, 08:13 to Apr 22, 2018, 08:15), Status (All), Artifact (All Integration Flows), and ID (Message or Application ID). Below this, a message list displays two entries:

- Aggregator Example** (Completed, Apr 22, 2018, 08:13:16, 93 ms) - Status: Completed. A green box indicates "Message processing completed successfully." Below it, "Processing Time: 93 ms".
- Aggregator Example** (Processing, Apr 22, 2018, 08:13:16) - Status: Processing.

On the right, a detailed view for the first entry is shown under the heading "Aggregator Example" (Last Updated at: Apr 22, 2018, 08:13:16). It includes tabs for Status, Properties, Logs, and Attachments. The Status tab shows the message processing completed successfully. The Properties tab lists the message ID (AFrcJ_sLs3kvwL2W7VailZM1gn_a) and sender (Sender_SOAP). The Logs tab shows a single log entry with Log Level: Info and Process ID: e6d20fe. An "Open Text View" link is also present.

Figure 4.122 First Message Arriving in SAP Cloud Platform Integration

In an aggregation scenario, log entries are added in pairs:

- The first log entry represents the message received into the **Aggregator** step.
- The second log entry represents the confirmation that the message has been persisted into the data store. In our case, the data store is named Aggregator-1, as previously shown in [Figure 4.120](#). Data stores are discussed in detail in [Chapter 8, Section 8.4](#).

We can also observe from the **Manage Stores** monitor tiles that there are two entries in the **Data Stores** tile ([Figure 4.123](#)).



Figure 4.123 View of the Manage Stores Tiles

Click on the **Data Stores** tile to view its content. Note that the two entries include:

- **Data Store Aggregation Repository**
In this repository, one entry is created for each new correlation ID ([Figure 4.124](#)).
- **Data Store**
One entry is created for each correlated message ([Figure 4.125](#)).

Overview / Manage Data Stores			
Data Stores (2)		Entries (1)	
Aggregator-1	1	Aggregator-1	Aggregator_Example/DataStoreAggregationRepository
Aggregator_Example			Delete
Aggregator-1	1	Entries (1)	
Aggregator_Example/DataStoreAggregationRepository		ID	Status
		AA2345	Waiting
		Retain Until: Jul 21, 2018, 08:21:11	Apr 22, 2018, 09:26:11
			Apr 22, 2018, 08:21:11

Figure 4.124 View of the Data Store Aggregation Repository Entries after the First Message

The screenshot shows the SAP Cloud Platform Integration Data Store Management interface. At the top left, there's a link to 'Overview / Manage Data Stores'. Below it, a search bar labeled 'Filter by Name' and some navigation icons. The main area has two sections: 'Data Stores (2)' and 'Entries (1)'. Under 'Data Stores (2)', there are two entries: 'Aggregator-1' (with ID '1') and 'Aggregator-Example'. Under 'Entries (1)', there is one entry named 'Aggregator-1' with ID 'AA2345-1'. This entry has columns for 'Status' (Waiting), 'Due At' (Feb 05, 2292, 07:21:11), and 'Created At' (Apr 22, 2018, 08:21:11). A note at the bottom says 'Retain Until: Jul 21, 2018, 08:21:11'.

Figure 4.125 View of the Data Store Entries after the First Message

Assuming three messages are sent to SAP Cloud Platform Integration with the same correlation ID; you can expect to have one entry in the **Data Store Aggregation Repository** and three entries in the **Data Store**.

Send the Second Message

Let's now send a second message ([Figure 4.126](#)) via SoapUI.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <OrderItem xmlns:demo="http://hci.sap.com/demo">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>2</ItemNo>
        <Quantity>5</Quantity>
        <Unit>1</Unit>
        <LastStatus>false</LastStatus>
      </Item>
    </OrderItem>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4.126 Second Message for the Aggregation Scenario

After sending the message in [Figure 4.126](#), monitor the data store one more time ([Figure 4.127](#)). Notice that because the correlation ID of the second message is the same as the first message, no new entry is added to the **Data Store Aggregation Repository**. As a result, the second row of [Figure 4.127](#) remains unchanged with one entry. However, a new entry has been added in the **Data Store**, as shown in the first row of [Figure 4.127](#).

Entries (2)			
ID	Status	Due At	Created At
AA2345--2	Waiting	Feb 05, 2292, 07:23:45	Apr 22, 2018, 08:23:45
AA2345--1	Waiting	Feb 05, 2292, 07:23:40	Apr 22, 2018, 08:23:40

Figure 4.127 View of the Data Store Entries after the Second Message

Send the Third Message

Let's now send a third message (Figure 4.128) via SoapUI. Notice that this third message has the element `LastStatus` set to `true`. This wasn't the case for the other messages that we sent previously.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <OrderItem xmlns:demo="http://hci.sap.com/demo">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>3</ItemNo>
        <Quantity>25</Quantity>
        <Unit>1</Unit>
        <LastStatus>true</LastStatus>
      </Item>
    </OrderItem>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4.128 Third Message for the Aggregation Scenario

According to the configuration of the **Aggregator** step (as shown earlier in Figure 4.120), when a message is received with the element `LastStatus` set to `true`, this represents the last message. This completion condition can be seen via the **Last Message Condition (XPath)** field (refer to Figure 4.120). This condition has now been met with our last message of Figure 4.128.

As the left panel of the screen shown in [Figure 4.129](#) demonstrates, there are four messages. The first three (from the bottom), represent the messages sent via SoapUI, whereas the last message (top) is automatically created after the last message is received or after the completion timeout condition has been met.

The screenshot shows the SoapUI Monitor Message Processing interface. At the top, there are filters for Time (Custom, from Apr 22, 2018, 08:23 to Apr 22, 2018, 08:32), Status (All), Artifact (All Integration Flows), and ID (Message or Application ID). Below the filters, a message list titled "Messages (4)" shows four entries:

Artifact Name	Status
Aggregator Example	Completed
Apr 22, 2018, 08:25:55	2 min 14 sec
Aggregator Example	Completed
Apr 22, 2018, 08:25:55	60 ms
Aggregator Example	Completed
Apr 22, 2018, 08:23:45	40 ms
Aggregator Example	Completed
Apr 22, 2018, 08:23:40	47 ms

Below the message list is a section for the artifact "Aggregator Example" last updated at Apr 22, 2018, 08:25:55. It includes tabs for Status, Properties (selected), Logs, and Attachments. The Logs tab displays "Runs (3)" with the following data:

#	Started At	Duration	Log Level	Process ID	Status
3	Apr 22, 2018, 08:25:55	28 ms	Info	e6d20fe	Completed
2	Apr 22, 2018, 08:23:45	1 min 9 sec	Info	e6d20fe	Processing
1	Apr 22, 2018, 08:23:40		Info	e6d20fe	Processing

The Attachments tab shows one entry:

Name	Type	Modified At
Account Payload	application/xml	Apr 22, 2018, 08:25:55

Figure 4.129 All Received Messages in the Monitor Message Processing

Also notice that the list of the correlated messages is mentioned under the **Logs** section ([Figure 4.129](#)). You can view more details by clicking on the **Open Text View** link (right side of [Figure 4.129](#)). The **Message Processing Log Attachment** opens where you can see the field **AggregateCompletedBy** ([Figure 4.130](#)). Possible values for this field include the following:

- **Predicate**: Indicates that the processing of the aggregate has been finished because the completion condition has been fulfilled.
- **Timeout**: Indicates that the processing of the aggregate is finished because the configured completion timeout has been reached.

Overview / Monitor Message Processing / Message Processing Log Attachments

Artifact Name: Aggregator Example	Status: Completed	Processing Time: 2 min 14 sec
Last Updated at: Apr 22, 2018, 08:25:55	Log Level: Info	

Log Account Payload

```

Message Processing Log:
StartTime = Sun Apr 22 06:25:55.046 UTC 2018
StopTime = Sun Apr 22 06:25:55.074 UTC 2018
OverallStatus = COMPLETED
MessageGuid = AFrcKmwQ08ufZ2Kb-bEYO9eAF9g-
AggregateCompletedBy= predicate
AggregateCorrelationId= AA2345
ChildCount = 0
ChildrenCounter = 27
ContextName = Aggregator_Example
CorrelationId = AFrcKmyuIVoE2mVXbpv4oCLF3qyH
IntermediateError = false
LastMessageNumber = 3
Node = vsa3953224
ProcessId = e6d20fef3a4137169155f19b0896c01f08575bed
ReceivedSequenceNumbers= [1-3]

```



```

Message Processing Log:
StartTime = Sun Apr 22 06:23:45.835 UTC 2018
StopTime = Sun Apr 22 06:24:55.233 UTC 2018
OverallStatus = PROCESSING
MessageGuid = AFrcKmwQ08ufZ2Kb-bEYO9eAF9g-
AggregateCorrelationId= AA2345
ChildCount = 0
ChildrenCounter = 1
ContextName = Aggregator_Example
CorrelationId = AFrcKmyuIVoE2mVXbpv4oCLF3qyH
IntermediateError = true
LastError = Messageprocessing has been blocked for more than 69s
Node = vsa3953224
ProcessId = e6d20fef3a4137169155f19b0896c01f08575bed
ReceivedSequenceNumbers= [1,2]

```



```

Message Processing Log:
StartTime = Sun Apr 22 06:23:40.213 UTC 2018
StopTime = Sun Apr 22 06:23:40.213 UTC 2018
OverallStatus = PROCESSING
MessageGuid = AFrcKmwQ08ufZ2Kb-bEYO9eAF9g-
AggregateCorrelationId= AA2345

```

Figure 4.130 Message Processing Log Attachment

4.6.3 Viewing the Aggregated Message

To view the resulting aggregated message, click on the **Account Payload** tab (refer to Figure 4.130). You're then presented with a message that includes all received and correlated messages, as shown in Figure 4.131.

Artifact Name: Aggregator Example Status: Completed Processing Time: 2 min 14 sec
 Last Updated at: Apr 22, 2018, 08:25:55 Log Level: Info

Log Account Payload

```
<?xml version="1.0" encoding="UTF-8"?>
<multipmap:Messages xmlns:multipmap="http://sap.com/xi/XI/SplitAndMerge">
  <multipmap:Message1>
    <OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>1</ItemNo>
        <Quantity>1</Quantity>
        <Unit>1</Unit>
        <LastStatus>false</LastStatus>
      </Item>
    </OrderItem>
    <OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>2</ItemNo>
        <Quantity>5</Quantity>
        <Unit>1</Unit>
        <LastStatus>false</LastStatus>
      </Item>
    </OrderItem>
    <OrderItem xmlns:demo="http://hci.sap.com/demo" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <orderNumber>AA2345</orderNumber>
      <Item>
        <ItemNo>3</ItemNo>
        <Quantity>25</Quantity>
        <Unit>1</Unit>
        <LastStatus>true</LastStatus>
      </Item>
    </OrderItem>
  </multipmap:Message1>
</multipmap:Messages>
```

Figure 4.131 Final Aggregated Order Item Message

Note

Using the **Aggregator** step in combination with a polling SFTP sender adapter can generate a high message volume and can consume a lot of resources. Use it cautiously.

4.7 Summary

Congratulations! You've now mastered the first basic integration scenarios using SAP Cloud Platform Integration. Look back and be proud of your achievements: you now know how to work with data within SAP Cloud Platform Integration, how to invoke external OData services, how to provision an OData service, how to map different data structures to each other, and finally how an **Aggregator** step works. You're now ready to tackle the next challenges: content-based message routing, the handling of messages containing lists of entries, and asynchronous message handling. These will all be covered in the next chapter.

Chapter 5

Advanced Integration Scenarios

So far, we've introduced the development environment, as well as the runtime, of SAP Cloud Platform Integration. The time has come to address more sophisticated, real-life integration scenarios. Here, more advanced patterns are supported for the integration developer.

In the previous chapter, we learned a lot about the inner workings of SAP Cloud Platform Integration. You've seen how to work with its data model, how to enrich messages with data retrieved from an external OData service, and how to solve mapping challenges using the built-in mapping engine. We'll continue our journey in this chapter with topics to help you address more advanced integration scenarios, such as the following:

- Message routing
- Working with lists
- Asynchronous message handling
- Sending messages to multiple receivers using multicast
- Asynchronous decoupling of inbound and outbound processing

Let's get started!

5.1 Message Routing

Cloud computing is currently one of the most talked-about topics in the IT industry. However, this trend of migrating toward cloud computing leads to an increased heterogeneity of a company's IT landscape, which itself brings increased need for integration. Messages need to be exchanged between on-premise and cloud applications. Fortunately, cloud-based integration solutions, such as SAP Cloud Platform Integration, can help companies solve this integration challenge.

If we take a closer look at how messages are treated within SAP Cloud Platform Integration, one question comes up repeatedly: How can we model different message handling execution paths (i.e., routes) in a single integration scenario? This question stands in the middle of what is known as content-based routing, the topic of this section. Content-based routing (CBR) takes care of forwarding messages to the right recipient depending on the contents of a message. As an example, let's look at an order. Depending on the type of item, an order might require different treatment within the processing chain or by dedicated backend systems. So, depending on the message's content, the order will need to be transferred to the respective system. That's what CBR is all about.

CBR is nothing new. It's one of the famous Enterprise Integration Patterns described in Hohpe and Woolf's *Enterprise Integration Patterns* (Addison Wesley, 2003). As we know from previous chapters, Apache Camel is the basic integration framework on which SAP Cloud Platform Integration is built. One major goal of the Apache Camel project was, from the beginning, the implementation of Enterprise Integration Patterns. Hence, we find the implementation of the CBR in SAP Cloud Platform Integration, as well. Let's see how you can apply the CBR pattern in your integration projects.

5.1.1 The Scenario

Let's start with a look at the scenario we want to build. An example integration flow using the CBR is shown in [Figure 5.1](#).

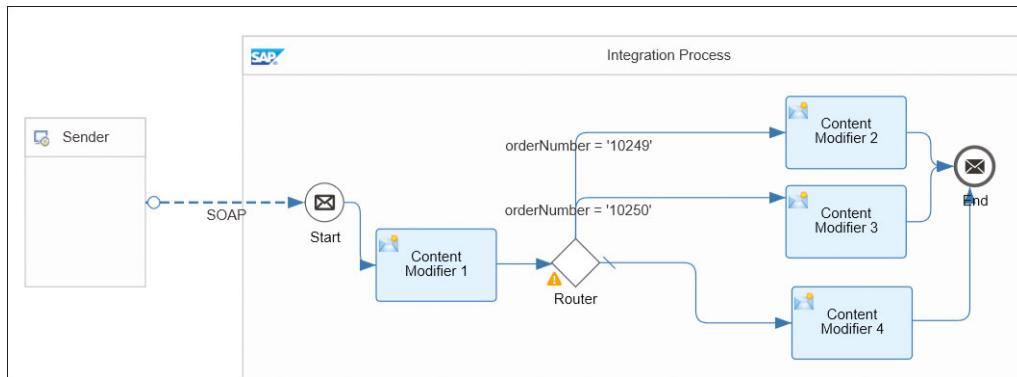


Figure 5.1 Example Integration Flow Using the Content-Based Router

The depicted integration flow shows different message handling execution paths after the diamond shape. The integration flow's semantical behavior can be described

as follows: the sender on the left (represented by the **Sender** pool) sends a message via the **SOAP** channel to the integration flow. Again, we reuse the same input message as Chapter 4. Its structure is shown in [Figure 5.2](#).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.2 Example Message

The incoming message starts the integration flow on the SAP Cloud Platform Integration server. The first **Content Modifier** step, **Content Modifier 1**, takes the order number from the message and stores it in the message's header area. [Figure 5.3](#) shows the **Content Modifier**'s configuration.

Action	Name	Type	Data Type	Value	Default
Create	OrderNo	XPath	java.lang.String	//OrderNumber	

Figure 5.3 Writing Data into the Message's Header Area

The order number is stored in the newly created header variable, `OrderNo`. We can later access this value to define routing conditions. Next, the CBR comes into the picture (refer to [Figure 5.1](#)). It's modeled using a Business Process Model and Notation (BPMN)-exclusive gateway (the diamond shape). As you know from [Chapter 4](#), the entire modeling environment of SAP Cloud Platform Integration is based on BPMN. In BPMN, the exclusive gateway is used to indicate the split of the sequence flow in several independent execution paths. Exactly one of the paths leaving the gateway (which is also known as a gate in BPMN nomenclature) will later be executed at runtime, depending on some conditions that are attached as labels to each of the outgoing sequence flows. However, if you take a close look at the gates, you'll recognize one

exception: the sequence flow leaving the gateway vertically, which is decorated with the tick mark  , has no condition associated with it. This is a default gate, which is executed during runtime if none of the other conditions meet the Boolean value TRUE. Now, we can describe the behavior of the gateway as follows:

- If the incoming order number equals 10249, the upper path will be followed.
- If the incoming order number equals 10250, the gate in the middle will be taken.
- In all other cases, the default gate will be activated.

To verify the correct behavior of the gateway during runtime, we'll set the body of the message via the respective **Content Modifier** shapes, which are connected with each of the three sequence flows leaving the gateway. The **Content Modifier** steps write the following messages as reply into the message's body:

- orderNumber = 10249 for the upper sequence flow
- orderNumber = 10250 for the sequence flow in the middle
- orderNumber unknown for the default gate

Figure 5.4 shows an example configuration for the uppermost **Content Modifier**.

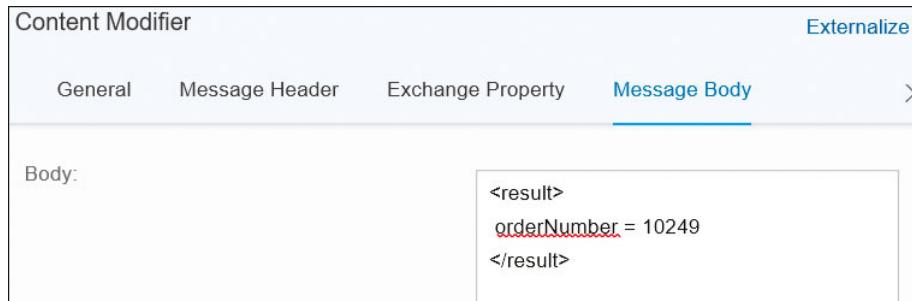


Figure 5.4 Configuration of the Content Modifier for the Uppermost Sequence Flow

5.1.2 Configuration of the Content-Based Router

Now we know how the CBR should behave during runtime. But how is this achieved during design time? Where can you find the gateway in the modeling palette of SAP Cloud Platform Integration's graphical editor? In the main menu of the palette shown in Figure 5.5, you'll find the **Message Routing** icon .

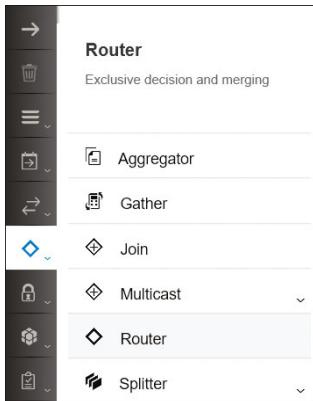


Figure 5.5 Router Shape in the Modeling Environment Palette

After you click on the **Message Routing** diamond, a submenu opens, revealing different routing options including the **Router** symbol (Figure 5.5). Click on it, move the mouse into the pool for the integration flow, and click again to position the shape. Then, model the three **Content Modifiers**, and connect them with sequence flows from the diamond shape to the respective **Content Modifier** activities. Note that you can only configure the gateway after you've connected it with the three previous steps; otherwise, you won't be able to configure the gates correctly because you won't have access to the sequence flow's properties to define the labels and evaluation conditions. So, let's configure each gate, one after another. We'll start with the uppermost one. Click on the sequence flow, leaving the gateway so that its color turns to orange (Figure 5.6).

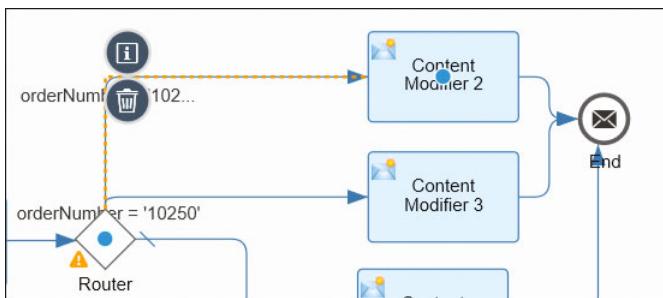


Figure 5.6 Selecting a Sequence Flow for Configuration

As always, you'll be able to configure the attributes of the selected shape in the **Properties** section, found beneath the process model. In our case, we want to tell the

runtime engine that the execution of the model should be continued on the upper path of our model, if the order number equals 10249. You have two options for defining such a routing condition:

- Directly access the contents of the message (in the body area of Camel's message model), and retrieve the value that should be used for the decision from there.
- Use header variables that have been declared and set before.

Note

The first option is only possible for XML-based message content. If your incoming message isn't available in XML, you'll have to convert it first.

Let's begin with the first option. Here, you have to define an XML Path Language (XPath) expression to the field you want to access. In our case, it's the **orderNumber** field of the incoming message (refer to [Figure 5.2](#)). Hence, the configuration looks like the one depicted in [Figure 5.7](#).

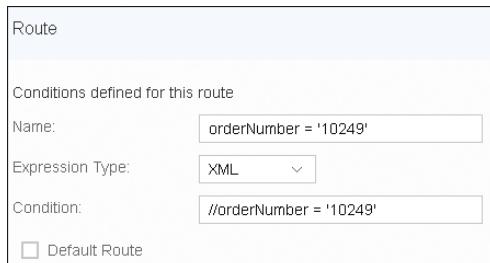


Figure 5.7 Defining the Condition for the Uppermost Sequence Flow

The **Name** field holds the string that shows up as a label attached to the sequence flow shown earlier in [Figure 5.1](#). The **Expression Type** dropdown list shown in [Figure 5.7](#), which contains the values **XML** and **Non-XML**, is of particular importance. The selected value influences how the **Condition** field is interpreted by the execution engine during runtime. If **XML** is chosen, the **Condition** is interpreted as an XPath expression. If **Non-XML** is chosen, it's interpreted as an expression using the Simple Expression Language. We'll see an example for the second case when we define the other gate. For now, though, stick with the XML case. The **Condition** is formulated using a classic XPath expression. You can also combine several expressions using the logical operators and or (e.g., `//orderNumber = '10249'` or `//orderNumber = '10250'`) to formulate more sophisticated routing logic.

For the second gate, we'll make use of the header variable `OrderNo`, which we created by the invocation of the very first **Content Modifier** in Figure 5.1, in conjunction with the configuration shown in Figure 5.3. The condition can be formulated now, as indicated in Figure 5.8.

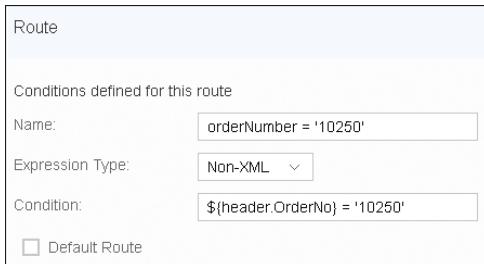


Figure 5.8 Configuring the Gate's Condition Using the Content of a Header Variable

We can easily identify the typical Camel Simple Expression Language for accessing variables (e.g., `$` or `{}`). The string `header` in `$(header.OrderNo)` indicates the area from which we want to load the value (the message's header area), and the `OrderNo` after the dot indicates the name under which we stored the value previously. Note that the **Expression Type** dropdown list has been changed to **Non-XML**. Because you use this dropdown list to define how the **Condition** string is interpreted, it should be clear that you can't mix XML-based variables with Camel-based variables. If you try to mix them, for example, `$(header.OrderNo) = '10250'` or `//orderNumber = '10251'` you'll receive a validation error (see Figure 5.9).

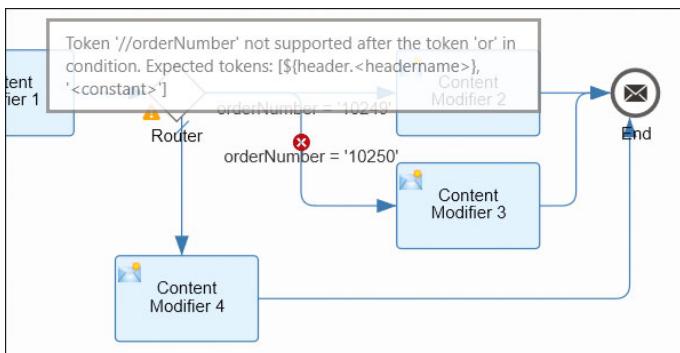


Figure 5.9 Validation Error if the Expression Contains a Mixed Expression of XML and Non-XML Parts

The definition of the last gate is probably the easiest part of the CBR's configuration. We simply have to set the **Default Route** checkbox (see [Figure 5.10](#)) to define the gate, which should be followed during runtime if none of the explicit conditions of the other gates evaluate to TRUE.

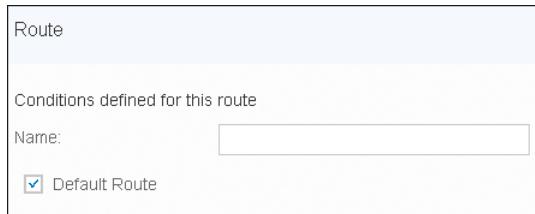


Figure 5.10 Defining the Default Route

From what you've learned so far, you know how to formulate expressions for the XML setting of the **Expression Type** field shown earlier in [Figure 5.7](#). You now need to apply the rules laid out in the XPath specification defined by the World Wide Web Consortium (W3C).

But what do you have to consider for the non-XML expressions, and which operators are allowed here? For your convenience, the table from the SAP Cloud Platform Integration help website has been reproduced (the original can be found at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud and search for "Define Router") in [Figure 5.11](#), which summarizes the operators allowed for formulating non-XML expressions.

Operator	Example
=	<code> \${header.SenderId} = '1'</code>
!=	<code> \${header.SenderId} != '1'</code>
>	<code> \${header.SenderId} > '1'</code>
>=	<code> \${header.SenderId} >= '1'</code>
<	<code> \${header.SenderId} < '1'</code>
<=	<code> \${header.SenderId} <= '1'</code>
and	<code> \${header.SenderId} = '1' and \${header.ReceiverId} = '2'</code>
or	<code> \${header.SenderId} = '1' or \${header.ReceiverId} = '2'</code>
contains	<code> \${header.SenderId} contains '1'</code>
not contains	<code> \${header.SenderId} not contains '1'</code>
in	<code> \${header.SenderId} in '1,2'</code>
not in	<code> \${header.SenderId} not in '1,2'</code>
regex	<code> \${header.SenderId} regex '1.*'</code>
not regex	<code> \${header.SenderId} not regex '1.*'</code>

Figure 5.11 Usage of Operators in Non-XML Expressions

5.1.3 Running the Content-Based Router Scenario

Now that our configurations are complete, we can finally run the scenario. Use a Simple Object Access Protocol (SOAP) tool of your choice (e.g., SoapUI), and invoke the solution. We'll use the input message shown earlier in [Figure 5.2](#). Depending on the order number's value, you'll receive respective replies from the integration flow. If your order number is 10250, the reply should look similar to [Figure 5.12](#).

If you provide a number for which no routing rule exists, you'll see the response shown in [Figure 5.13](#) because the default route of the gateway was fired.

```
<soap:Envelope xmlns:soap="http://schemas.xml  
    <soap:Body>  
        <result>orderNumber = 10250</result>  
    </soap:Body>  
</soap:Envelope>
```

Figure 5.12 Reply Message if Order Number of Input Message Was Set to 10250

```
<soap:Envelope xmlns:soap="http://schemas.xml  
    <soap:Body>  
        <result>orderNumber unknown</result>  
    </soap:Body>  
</soap:Envelope>
```

Figure 5.13 Reply Message if an Order Number Was Provided for Which No Routing Rule Exists

At this point, we could stop with the description of the CBR. However, one interesting question hasn't yet been answered: What happens if the routing rules contain overlapping conditions? Mistakes are always possible, and especially for complex routing conditions, these mistakes may sometimes result in overlapping conditions, so that potentially two or more of the conditions could evaluate to true during runtime. Hence, more gates may be triggered. On the other hand, we know that the exclusive gateway will trigger one—and only one—gate. So, in the case of overlapping conditions, which of the gates will be triggered, and can we influence the sequence in which the expressions will be evaluated? Let's try a little experiment. We'll change the conditions in such a way so that they overlap. Let's change the condition of the gate labeled with **orderNumber = '10250'** to **\${header.OrderNo} = '10249'**. This overlaps with the gate already labeled with **orderNumber = '10249'** and its condition **// orderNumber = '10249'**. Both are checking against order number 10249.

Now, save your changes, deploy them, and run the scenario again using 10249 as the input value for the order number. After we run the scenario in our own environment, we see the result shown earlier in [Figure 5.12](#). Thus, the changed path was executed, although if you compare our design of the scenario shown earlier in [Figure 5.1](#), you'll see it's positioned in the middle of the three gates. One might think the conditions are evaluated from top to bottom in the visual diagram, and so the model's visual appearance has something to do with execution sequence. Our experiment proves that this isn't the case. We also stress that *our* scenario works this way. It may be that *your* scenario is still working correctly!

What else influences the execution sequence, then? The answer is hidden behind the gateway shape itself. Select the diamond shape of the router and take a look at its properties. In our example, the gateway has the properties shown in [Figure 5.14](#).

Router				
Routing Condition				
Order	Route Name	Condition Expression	Default Route	
1	orderNumber = 10250	<code>#{header.OrderNo} = '10249'</code>	<input type="checkbox"/>	
2	orderNumber = 10249	<code>//orderNumber = '10249'</code>	<input type="checkbox"/>	
3			<input checked="" type="checkbox"/>	

Figure 5.14 Configuration of the Exclusive Gateway

Take note of the **Order** column: it tells us the sequence in which the conditions will be evaluated. The route labeled with **orderNumber = 10250** will be evaluated first. Additionally, because the condition is true, we get the expected result. The second row will no longer be evaluated because the gateway has already found a valid gate, and no more gates are allowed to fire due to the exclusive behavior of the gateway.

This explains the gateway's behavior. But how can we influence the evaluation's sequence? The answer is rather straightforward: The order of the rows is determined by the connection's modeling sequence. Every connection you're modeling from the gateway to any task following the gateway adds a new row to this table. Note that every new row will be added at the bottom of the table. You can conclude from this description how we created the process model shown earlier in [Figure 5.1](#). We first drew the connection to the **Content Modifier** in the middle (resulting in the first row in the table), then to the one at the top (second row in the table), and finally to the

Content Modifier at the bottom (third row in the table). If we want to change the execution sequence, what do we need to do? Take a look at [Figure 5.14](#) again. We want the second row to be at the first position. So, in your process model, delete the connection responsible for the first table row: the connection labeled with **orderNumber = 10250**. The second row moves up to first place automatically, exactly like we want. Next, draw the connection that we just deleted again, add the label and the condition in its properties, and verify the condition's list at the gateway. It should now look like [Figure 5.15](#).

Router				
Routing Condition				
Order	Route Name	Condition Expression	Default Route	
1	orderNumber = 10249	//orderNumber = '10249'	<input type="checkbox"/>	
2			<input checked="" type="checkbox"/>	
3	orderNumber = 10250	#{header.OrderNo} = '10249'	<input type="checkbox"/>	

Figure 5.15 Evaluation Sequence after Deleting and Redrawing the Connection with the Route Name `orderNumber = 10250`

Note the changed order sequence in comparison to the one shown earlier in [Figure 5.14](#). If you invoke the route again with order number 10249, you'll see the expected (correct) result, as shown in [Figure 5.16](#).

```
<soap:Envelope xmlns:soap="http://schemas.xml  
    <soap:Body>  
        <result>orderNumber = 10249</result>  
    </soap:Body>  
</soap:Envelope>
```

Figure 5.16 Returned Message after Correcting the Evaluation Sequence at the Gateway

To summarize, routing messages to different message handling paths is an important aspect in every integration project. SAP Cloud Platform Integration is based on Apache Camel, which implements typical Enterprise Integration Patterns. One of those patterns is the CBR, whose task is to split the sequence flow into different independent execution paths, which can then be activated based on certain conditions. Exactly one of those execution paths will be selected during runtime. You've learned

how to model the CBR in SAP Cloud Platform Integration's graphical modeling environment and how to configure the conditions correctly. To define the expressions, you have two options at your disposal: XML and non-XML. You learned when to use which option, and how the condition's evaluation sequence can be influenced. Now it's your turn to work with the CBR in your own integration projects.

5.2 Working with Lists

So far, you've learned quite a bit about SAP Cloud Platform Integration's functionality, the basic concepts behind it, and the various modeling techniques for solving typical integration problems, such as message enrichment, message mapping, and message routing. However, in the examples so far, we focused on handling messages containing just one item, such as a single order. In this section, we'll dive into the details of coping with messages comprising a list of entries. Questions such as the following will be answered in the next sections:

- How do I split up such a message into individual pieces?
- How do I iterate over each list item?
- How do I handle the resulting single messages in a SAP Cloud Platform Integration message processing chain?
- How do I combine the results of each single message handling sequence back into one response message?

5.2.1 The Scenario

In real-life scenarios, integrators are quite frequently confronted with input messages consisting of several items of the same message structure, grouped in a list (e.g., order line items). The integrator wants to iterate over the list: individual list items have to be separated and individually managed by the integration flow. Finally, the result of each individual message handling procedure needs to be consolidated into one response message, sent to either the sender of the message (in the case of a synchronous scenario) or to the final recipient (in the case of an asynchronous scenario). To illustrate this functionality using SAP Cloud Platform Integration, we'll use the input message shown in [Figure 5.17](#) throughout this section.

[Figure 5.17](#) is based on a Web Services Description Language (WSDL) file, which was created using the Enterprise Services Builder of SAP Process Integration. You can, of

course, use any XML tool supporting the WSDL standard. We've included our *SendOrderList_Async.wsdl* WSDL file with the book downloads at www.sap-press.com/4650. The message contains a list of order numbers; the other fields aren't yet relevant.

```

<soapenv:Envelope xmlns:soapenv="http://schemas
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderList_MT>
      <!--Zero or more repetitions:-->
      <orders>
        <orderNumber>10248</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10249</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10250</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
    </demo:OrderList_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.17 Example Message Comprising a List of Order Numbers

Our goal is to split the message into individual order messages, enrich each individual order with order details (e.g., shipping date, shipping city, shipping address, etc.), and send back the enriched message as a reply to the sender in a synchronous scenario. We'll solve this problem in two steps. The first step is to understand splitting one input message into several individual messages (splitter pattern) and then joining back the pieces into one large message (gather/merge pattern). The second step explains how to enrich the individual messages with order details (enricher pattern) and then collect those results into one large message. You can see by this example how patterns help to build more complex integration solutions. We encourage you to recognize (and to implement) solutions based on patterns. After you understand the basic principle, you can apply this knowledge to even more complex scenarios.

The key to splitting large messages into individual pieces, iterating over them, and joining them back again into one large message is the use of two new steps from SAP Cloud Platform Integration's web-based graphical modeler and positioning them correctly in your message processing chain (i.e., route). The **Splitter** and **Gather** steps are used in the integration process model depicted in [Figure 5.18](#). The figure shows the message processing chain for the first step of the implementation plan, as outlined previously.

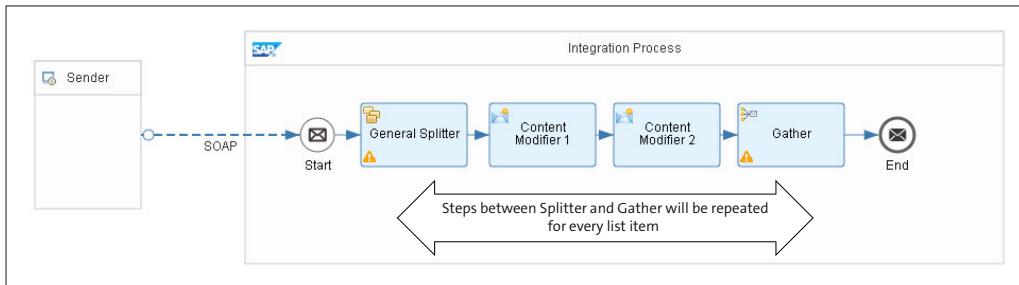


Figure 5.18 Splitting a Large Message into Single Pieces and Collecting them Back Using Gather

You can find the **Splitter** step at the beginning and the **Gather** step at the end of the processing chain. It's important for the overall understanding of the entire splitter-gather construct to recognize the following:

- The steps between a **Splitter** and a **Gather** step (the two **Content Modifier** steps in our case) are executed as many times as the **Splitter** creates individual messages.
- Each step within a **Splitter/Gather** pair will receive the separated individual messages that the **Splitter** has created as an input message, one after another.
- It's possible to model a splitter scenario without a **Gather** step. In those cases, all steps following the **Splitter** will be executed repeatedly until an **End** event is reached. So, the repeated execution of steps is dependent on the **Splitter** step, not on the **Gather** step.

5.2.2 Configuring the Integration Flow

Let's see what the configuration looks like for our scenario. As a reminder, we'll repeat the settings of the **Sender** pool and the message flow from the **Sender** pool to the **Start** message event.

When drawing the line for the message flow between the **Sender** pool and the **Start** message event, you select **SOAP** as **Adapter Type** and **SOAP 1.x** as **Message Protocol**. [Figure 5.19](#) shows the **General** tab of the SOAP adapter's channel configuration, which opens in the **Properties** section after creation of the message flow.

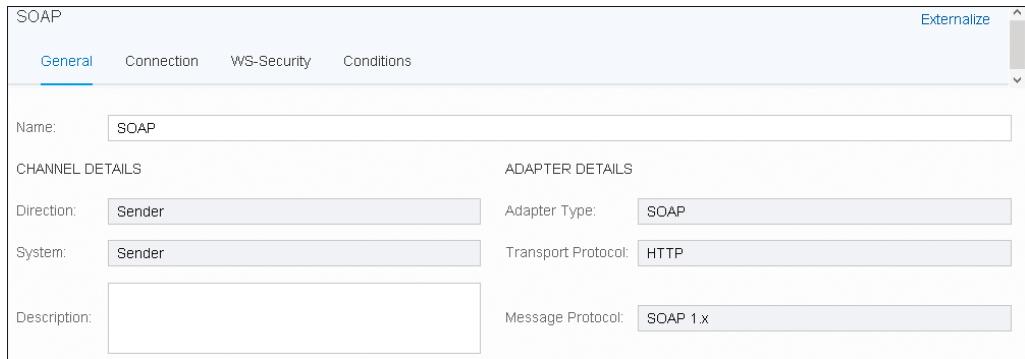


Figure 5.19 SOAP Channel: General Configuration

The connection details you can now configure in the **Connection** section of the SOAP adapter's configuration, as shown in [Figure 5.20](#). The **Address** field is important here because the address entered will later be part of the URL used to call the flow.

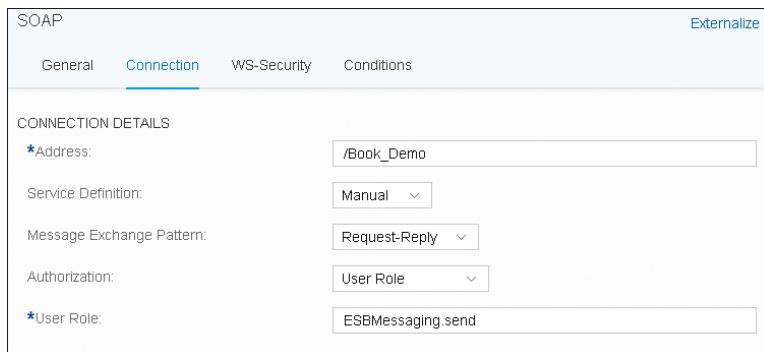


Figure 5.20 SOAP Channel: Connection Configuration

Because we want to get a response to our SOAP requests, we keep **Manual** as the **Service Definition** and **Request-Reply** as the **Message Exchange Pattern**. In [Section 5.3.1](#), we'll go into more detail about message exchange patterns (MEPs). We'll try out different configuration options and see the different behaviors.

We set the **Authorization** dropdown field to **User Role** to allow the sender to call the integration flow with username and password credentials while invoking the integration flow. In the **User Role** field we keep the default value `ESBMessaging.send`. The different authorization options and available user roles were described already in [Chapter 2](#).

Let's continue with the integration flow itself. The first step in the flow, after instantiating it, is the **Splitter** activity. As this step is new, we'll explain it in more detail. During development, you can find the activity in the palette on the left of the modeling environment following the path **Message Routing** • **Splitter** ([Figure 5.21](#)) • [**General Splitter**](#) ([Figure 5.22](#)).

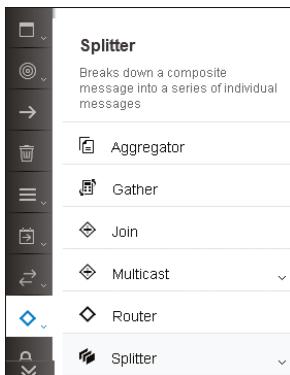


Figure 5.21 Opening the Splitter Submenu from the Palette

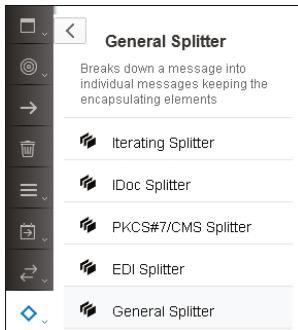


Figure 5.22 Selecting the General Splitter from the Splitter Submenu

SAP Cloud Platform Integration supports several different splitter types; however, in this section, we'll focus solely on the **General Splitter**.

Other Splitter Implementations in SAP Cloud Platform Integration

SAP Cloud Platform Integration provides several splitter implementations (see [Figure 5.22](#)). The **General Splitter** will be described in more detail throughout this chapter, but the following splitters are also available:

- **Iterating Splitter**

This splitter behaves most like the **General Splitter** as it also splits up a composite message into a series of individual messages; however, it doesn't copy the enveloping parts of a message to the single split messages. But what are enveloping elements, exactly? They are the message parts of the original incoming message above the nodes that are used for splitting. [Figure 5.23](#) contains a visual depiction of the differences between the **General Splitter** and the **Iterating Splitter**.

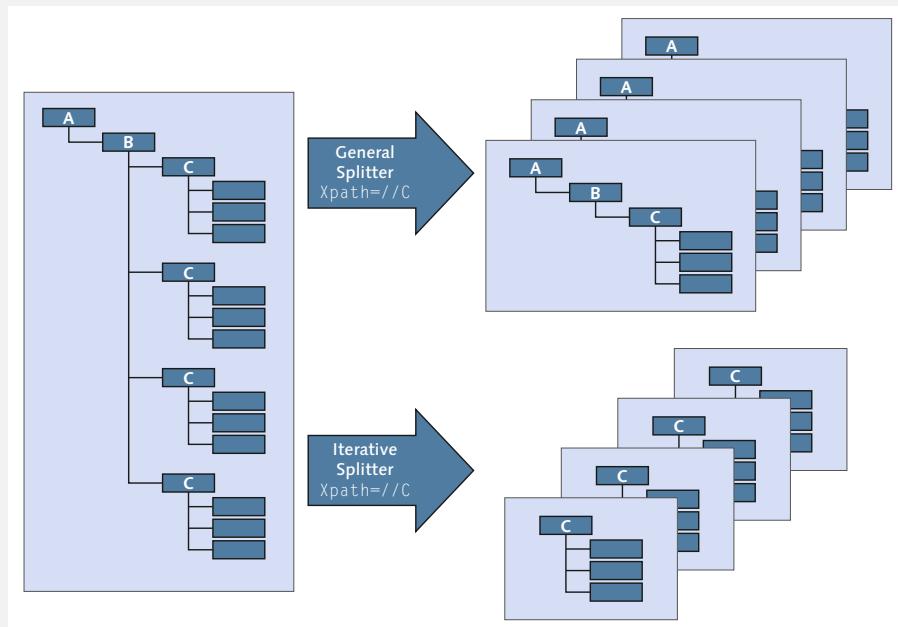


Figure 5.23 Differences between General Splitter and Iterating Splitter

The **Iterating Splitter** simply copies the parts beginning with the splitting tag (**C**, in the example shown in [Figure 5.23](#)) and the subnodes, whereas the **General Splitter** also copies the nodes above the splitting tag (**A** and **B** in the example). This is especially important if you want to navigate to elements in the tree structure using absolute XPath expressions.

- **IDoc Splitter**

This dedicated splitter takes care of composite IDoc messages. By splitting the composite IDoc into a series of individual IDoc messages, including the enveloping elements of the composite IDoc message. There are no special configuration settings for this splitter.

- **PKCS#7/CMS Splitter**

PKCS stands for Public-Key Cryptography Standard, which is used to sign and encrypt messages. This splitter is useful if a client sends a message that is PKCS#7-signed and contains both a signature and content. This splitter type breaks down the signature and the content into separate files. For configuration, you can provide names for the files that should contain either the payload or the signature after the splitting step. You can also prescribe which file should be handled first after splitting (the signature or the content file), and you can decide whether the payload/signature should be Base64 encoded after splitting.

- **EDI Splitter**

The **EDI Splitter** may not appear in your list, as it's available only with the SAP Cloud Platform, Enterprise Edition. It can be used to split composite Electronic Data Interchange (EDI) messages, and you can configure that inbound EDI messages are validated and acknowledged. The **EDI Splitter** will be described in more detail in [Chapter 7](#), where we'll use it in a business-to-business (B2B) scenario.

After positioning the **General Splitter** shape in the main pool named **Integration Process** (refer to [Figure 5.18](#)), you can set its properties. We've configured it with the parameters shown in [Figure 5.24](#).

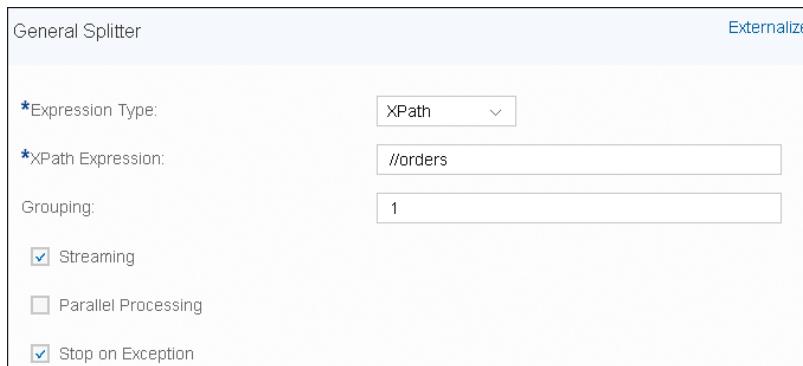


Figure 5.24 Configuration of the Splitter Step

Let's walk through the parameters one by one:

- **Expression Type**

In this dropdown, you select how the split points will be identified in the inbound message—either via **XPath expression** or **Line Break**. As our inbound message is an XML message, we can easily define the split points via XPath expression. The **Line Break** option is useful for non-XML messages that have multiple lines and need to be split into smaller messages. If your inbound message would, for example, be a .csv file containing several lines, the **Line Break** option can be used to split the message into smaller messages containing a defined number of lines or into messages containing only one line. Note that empty lines in the inbound message will be ignored; no empty messages will be created.

- **XPath Expression**

This directs the integration engine during runtime to search for the given tag in the input message and take it as the split argument. In our example, we've used the relative path `//orders`. Relative paths are indicated by the double-slash at the beginning and are quite convenient for allowing the engine to search for the tag's occurrence in your input message. The alternative would have been to use absolute paths starting with a single forward slash. As a consequence, you would need to know the exact path, from the root to the tag that should be used for splitting. If you wanted to extend the message later, for example, by adding tags between the root tag and the splitting tag, the absolute path would no longer be valid, as it doesn't consider the new tag on the way from the root to the splitting tag. Hence, absolute paths are quite static and error-prone when it comes to changes on the message's structure. Conversely, this error would not happen with a relative path.

Let's see how the definition of the XPath expression influences execution during runtime. Take a look at our example input message in [Figure 5.17](#). We have three `<orders>` tags in our message. As such, the splitter will generate three individual messages. Each individual message forwarded to the two **Content Modifier** steps following the **Splitter** looks like [Listing 5.1](#) (with different order number values):

```
<orders>
    <orderNumber>10248</orderNumber>
    <customerName>?</customerName>
    <orderAmount>?</orderAmount>
    <currency>?</currency>
    <taxAmount>?</taxAmount>
</orders>
```

Listing 5.1 Individual Message Containing Exactly One Order

Hence, the single messages will also contain the `<orders>` tags. Consider this while processing each of the individual messages. If you want to understand what the message produced by the splitter looks like in greater detail, read on to [Section 5.2.3](#).

■ Grouping

With the grouping parameter, you define the number of items that should be grouped together into one message for individual processing. As we've chosen **1** as the value, *every occurrence of `<orders>` results in a dedicated individual message for further processing*. If we had selected **2** as our value, the splitter would group the first two items as a single message, the next two items in the second message, and so on. So, if the **Grouping** parameter was set to 2 and assuming an input message contains 10 items in total, the **Splitter** would generate 5 messages, each containing 2 items.

■ Streaming

The **Streaming** parameter is important with regards to memory consumption. Normally, the splitter works on messages by loading them completely into the main memory. However, messages can get quite large. Think about payments that are being sent to a bank once a day. They are collected over the course of the day and forwarded sometimes at night. These collected messages can become huge, and it doesn't make sense to load them entirely into main memory before processing them (sometimes it's even impossible to do so). Therefore, it's useful to let the **Splitter** start working on the incoming streamed data, even though it isn't yet entirely loaded. This is called *streaming*, as it allows you to read a chunk of data, work on it, read another chunk, and so forth, until the entire message is processed.

■ Parallel Processing

As the name indicates, this parameter allows you to run the individual message processing tasks in parallel, leveraging Java's concurrency features using thread pools. By parallelizing tasks, you can get more work done in less time. However, the execution sequence can't be guaranteed, as the threads run independent of each other.

■ Stop on Exception

If this checkbox is set, the route's processing will immediately stop if there is an error, and that error will be propagated back instantly. Otherwise, the splitter continues working on the individual messages and reports the error back at the end, after handling the complete input message.

Now that we have an understanding of how the **Splitter** handles the incoming message, we can continue with the first **Content Modifier**. Its behavior has been described

several times already. As such, it's enough to just take a quick look at its configuration in [Figure 5.25](#).

Action	Name	Type	Data Type	Value	Default
<input type="checkbox"/> Create	OrderNo	XPath	java.lang.String	#/orders/orderNumber	

[Figure 5.25 Configuration of the First Content Modifier in the Route](#)

Note

Because this **Content Modifier** is placed after the **Splitter** step, it's invoked for each individual message created by the **Splitter** representing one order out of the original message's order list. This is crucial to understanding the entire route.

We're setting a variable named `OrderNo` in the message's header area and storing the current order number of the current item for later reference. We also could have chosen `//orderNumber` as an entry for the **Value** field as it's a relative path, and there is only one order number available in each individual message.

The recently stored header value will be retrieved by the second **Content Modifier** and copied into the result message as the configuration in [Figure 5.26](#) shows.

```
<orderDetails>
${header.OrderNo}
</orderDetails>
```

[Figure 5.26 Configuration of the Second Content Modifier in the Route](#)

Again, the second **Content Modifier** will also be invoked for each individual message. It's the **Gather** step that finally collects all the individual single messages created for each invocation of the **Splitter** into one bulk message. [Figure 5.27](#) shows the **Gather** step's configuration.



Figure 5.27 Configuration of the Gather Step

All of the resulting single messages are of the same format (hence, the **Incoming Format** dropdown field is set to **XML (Same Format)**), and the **Gather** step should simply put each of the individual pieces together into one result message. Consequently, the **Aggregation Algorithm** dropdown field is set to **Combine**.

The other configuration options provided by the **Gather** step actually depend all on the entry chosen from the **Incoming Format** dropdown list. The list provides three options:

- **Plain Text**

For the **Aggregation Algorithm**, only **Concatenate** is allowed. All of the plain text messages generated by the single messages will simply be concatenated to one large string, one after another. This is the result being propagated back to the caller.

- **XML (Different Format)**

For the **Aggregation Algorithm**, only **Combine** is allowed, which is similar to the behavior of the **Concatenate** option in the previous case. The individual XML fragments from the single messages will be brought together in one response message using a multimapping from SAP Process Integration's mapping engine.

- **XML (Same Format)**

If you choose this option, you can influence the construction of the response message in two ways:

- First, you can also apply the **Combine** option of the **Different Format** case. Its behavior is identical to the one just described.
- Second, there is one more option that allows you to copy parts from the source message into a user-defined envelope consisting of XML tags. The XML node from which parts of the source message should be copied is formulated using

an XPath expression. The XML envelope is defined using an absolute path definition, such as `/root` or `/level_1/level_2`. The associated configuration screen is depicted in [Figure 5.28](#). The behavior is actually pretty simple: the XPath expression of the **Combine from source (XPath)** field takes the source message (the single message created by the **Splitter**), navigates within that message to the node specified by the XPath expression, and copies the node, including all nested XML subnodes. It takes the copied XML-subtree snippet and pastes it after the tags that the user has defined as the envelope. The idea behind the envelope is that every valid XML document requires one root element, and this is exactly what you can define in the **Combine at target (XPath)** field: the root element (e.g., `/root`) or, if necessary, an absolute root path (e.g., `/level_1/level_2`). During runtime, the copied XML subtree will be pasted after the node(s) given in the **Combine at target (XPath)** field. In other words, the node in the **Combine at target (XPath)** field is the parent of the snippet that needs to be inserted. The corresponding closing tag(s) of the envelope will automatically be added, after the gathering has completed (e.g., `</root>` or `</level_2></level_1>`).

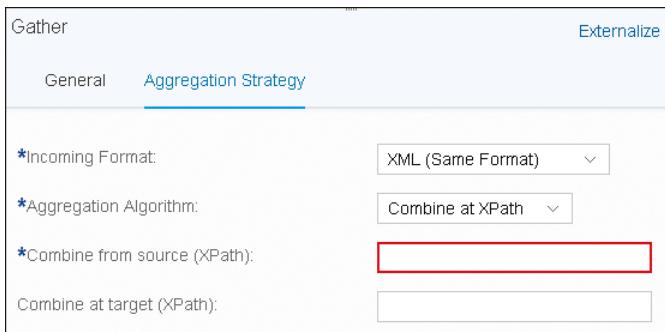


Figure 5.28 Configuration of the Gather Step When the Single Messages Should Be Combined Using XPath Expressions

Let's take a look at a concrete example. We'll assume the **Splitter** generates two messages, as shown in [Listing 5.2](#) and [Listing 5.3](#) (they will later be the source messages in the **Gather** step's configuration).

```
<payload>
<route>
    <multicast>Parallel</multicast>
    <branch>A</branch>
```

```
</route>
</payload>
```

Listing 5.2 First Generated Splitter Message

```
<payload>
  <route>
    <multicast>Parallel</multicast>
    <branch>B</branch>
  </route>
</payload>
```

Listing 5.3 Second Generated Splitter Message

Next, assume the following settings for the configuration of the **Gather** step:

- **Combine from source (XPath)** (only absolute XPath expressions are allowed for this field): /payload/route
- **Combine at target (XPath)**: /xyz/abc

The final resulting message looks like [Listing 5.4](#).

```
<xyz>
  <abc>
    <route>
      <multicast>Parallel</multicast>
      <branch>A</branch>
    </route>
    <route>
      <multicast>Parallel</multicast>
      <branch>B</branch>
    </route>
  </abc>
</xyz>
```

Listing 5.4 Combined Message Generated by the Gather Step

You can now easily understand how this result message was created: the engine copied the parts from the source messages starting at the /payload/route node, including the <route> tag and all nested nodes, and pasted it into the target message, which starts with the <xyz><abc> nodes representing the opening part of the envelope. Remember: the pasting will be done for all messages resulting from the splitter.

Hence, we have two `<route>` tags in between the envelope. Finally, we add the closing tags of the envelope. They can be derived from the definition of the **Combine at target (XPath)** field. As the definition for that field was `/xyz/abc`, the closing tags must be in opposite sequence, resulting in `</abc></xyz>`.

Note

It isn't mandatory to provide an entry for the **Combine at target (XPath)** field. If you leave the field empty, the resulting message will have the same tags that are specified in the **Combine from source (XPath)** field. Referring to the preceding example, the resulting message would start with `<payload><route>`.

5.2.3 Running the Integration Flow

Now that we've configured the scenario completely, we can finally run it. Save your changes, and deploy your integration flow. Don't forget to retrieve the URL for invoking the integration scenario from SAP Cloud Platform Integration's **Monitor** (refer to [Chapter 2, Section 2.3.7](#), for instructions on how to retrieve the URL). Provide the URL in the SOAP tool of your choice (e.g., SoapUI), and invoke the integration flow. The result should look like the one in [Figure 5.29](#).

The flow successfully retrieved the order numbers from the incoming message containing the order list and created an appropriate response message, which is the exact result we hoped to achieve.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
      <multimap:Message1>
        <orderDetails>10248</orderDetails>
        <orderDetails>10249</orderDetails>
        <orderDetails>10250</orderDetails>
      </multimap:Message1>
    </multimap:Messages>
  </soap:Body>
</soap:Envelope>
```

Figure 5.29 Final Response Message after Invoking the Integration Flow with a List of Order Numbers

What Is the Splitter Delivering to the Processing Chain?

In some situations, it might be useful to fully understand what the individual messages produced by the **Splitter** actually look like, and which ones reach the next step

of the integration flow. Maybe you want to pick a concrete field by an absolute XPath expression instead of using a relative XPath due to a potential name conflict. In this case, you need to know exactly what the single message looks like; otherwise, your absolute path won't work. We'll take the process model shown earlier in [Figure 5.18](#) as the basis for this task. Next, configure the first **Content Modifier**, as shown in [Figure 5.30](#).

Action	Name	Type	Data Type	Value	Default
Create	splitterResult	XPath	org.w3c.dom.Document	/	

Figure 5.30 Adding the Complete Single Message into the Header Area

We're using an absolute XPath expression pointing to the root of the received single message. Hence, we really put the complete message into the header variable named `splitterResult`. In the second **Content Modifier**, we now simply pick the variable's content and place it into the body. The result of the second **Content Modifier**'s configuration is shown in [Figure 5.31](#).

```
<splitter_single_message>
${header:splitterResult}
</splitter_single_message>
```

Figure 5.31 Copying the Splitter's Message into the Body

Running the scenario results in the output depicted in [Figure 5.32](#).

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
      <multimap:Message1>
        <splitter_single_message>
          <demo:OrderList_MT xmlns:demo="http://cpi.sap.com/demo">
            <orders>
              <orderNumber>10248</orderNumber>
              <customerName?></customerName>
              <orderAmount?></orderAmount>
              <currency?></currency>
              <taxAmount?></taxAmount>
            </orders>
          </demo:OrderList_MT>
        </splitter_single_message>
        <splitter_single_message>
          <demo:OrderList_MT xmlns:demo="http://cpi.sap.com/demo">
            <orders>
              <orderNumber>10249</orderNumber>
              <customerName?></customerName>
              <orderAmount?></orderAmount>
              <currency?></currency>
              <taxAmount?></taxAmount>
            </orders>
          </demo:OrderList_MT>
        </splitter_single_message>
        <splitter_single_message>
          <demo:OrderList_MT xmlns:demo="http://cpi.sap.com/demo">
            <orders>
              <orderNumber>10250</orderNumber>
              <customerName?></customerName>
              <orderAmount?></orderAmount>
              <currency?></currency>
              <taxAmount?></taxAmount>
            </orders>
          </demo:OrderList_MT>
        </splitter_single_message>
      </multimap:Message1>
    </multimap:Messages>
  </soap:Body>
</soap:Envelope>

```

Figure 5.32 Result Message after Invoking the Integration Flow

In between the `<splitter_single_message>` tags, you can now easily identify the fragment that was produced by the splitter. With this knowledge, you can formulate the absolute path within that single message to the `orderNumber` field: `/demo:OrderList_MT/orders/orderNumber`. Let's quickly verify this by adjusting the two **Content Modifier**'s configurations. You can find the updated properties in [Figure 5.33](#) and [Figure 5.34](#).

The screenshot shows the 'Content Modifier' configuration screen. At the top, there are tabs: 'General', 'Message Header' (which is selected), 'Exchange Property', and 'Message Body'. Below the tabs, under 'Headers:', there is a table with columns: Action, Name, Type, Data Type, Value, and Default. A single row is present: Action is 'Create', Name is 'orderNumber', Type is 'XPath', Data Type is 'java.lang.String', Value is '/demo:OrderList_MT', and Default is empty. There are 'Add' and 'Delete' buttons at the top right of the table.

Figure 5.33 Accessing the orderNumber Field via an Absolute XPath Expression in the First Content Modifier

The screenshot shows the 'Content Modifier' configuration screen with the 'Message Body' tab selected. Under 'Body:', there is a code editor containing the following XML fragment:

```
<orderNumberList>
    ${header.orderNumber}
</orderNumberList>
```

Figure 5.34 Pasting the Copied Order Numbers into the Result Message's Body in the Second Content Modifier

We can't run this scenario quite yet because we have to consider one small, but important, detail. Take a look at the **Value** column in [Figure 5.33](#). The XPath expression begins with `/demo:OrderList_MT`, right? The important detail is the namespace `demo:`. The message handling route isn't aware of this namespace and what it means. Hence, we have to explicitly declare the namespace in the route's configuration. You can define some global settings for an integration flow by clicking somewhere outside of the pools to reach the route's properties beneath the process model. After you can see the properties of the integration flow, select the **Runtime Configuration** tab (see [Figure 5.35](#)).

The **Runtime Configuration** tab allows you to define the namespace mappings in a dedicated field. We've just copied the namespace definition from [Figure 5.32](#). Remember to remove the quotation marks after pasting the string into the **Namespace Mapping** field! Now the route is aware of the namespace and can handle the XML fragment accordingly. Try out the changed integration flow. The result should look like [Figure 5.36](#).

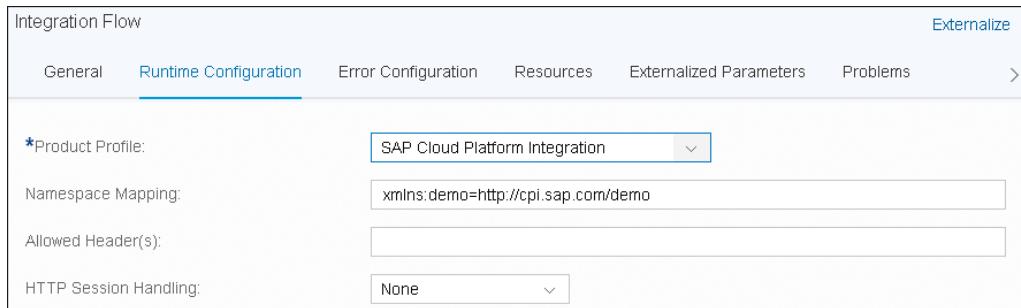


Figure 5.35 Global Configuration Options for an Integration Flow

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
      <multimap:Message1>
        <orderNumberList>10248</orderNumberList>
        <orderNumberList>10249</orderNumberList>
        <orderNumberList>10250</orderNumberList>
      </multimap:Message1>
    </multimap:Messages>
  </soap:Body>
</soap:Envelope>
```

Figure 5.36 Result Message after Picking the Order Number via Absolute XPath Expression

5.2.4 Enriching Individual Messages with Additional Data

We began with a relatively simple example to concentrate on the behavior of the **Splitter** and **Gather**. However, we can now extend this example to something more useful by invoking an external OData data source and enriching our result message. This reflects a typical example where some basic data needs to be enriched by external sources. In our case, we want to retrieve order details for each of the order numbers that we've extracted. In [Chapter 4, Section 4.3.3](#), we configured an OData connection to retrieve detailed data for order numbers. This is exactly what we'll do next: replace the second **Content Modifier** (which actually just sets the body of the single message artificially with the extracted order number) with a **Request-Reply** step invoking the OData service and providing useful data as response for each single message the **Splitter** created. After adding the **Request-Reply** step into the integration flow, the scenario finally looks like the one shown in [Figure 5.37](#).

The configuration of the **Request-Reply** step is identical to the one described in [Chapter 4, Section 4.3](#), and can be found there.

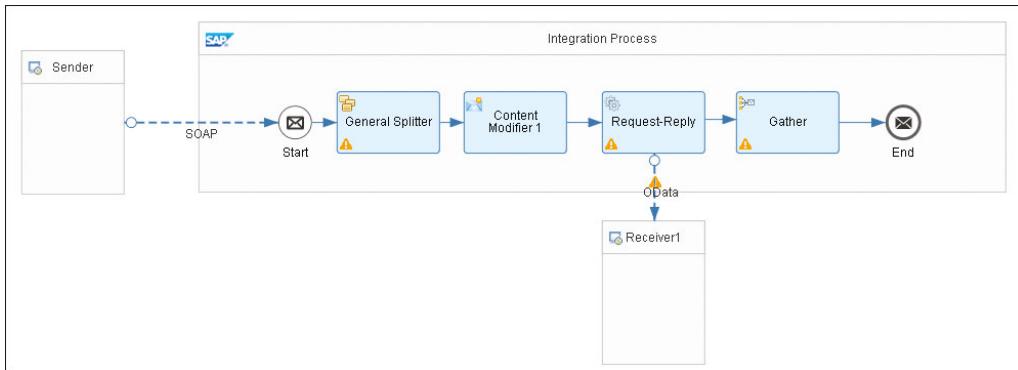


Figure 5.37 Integration Flow with Splitter, Gather, and the Invocation of an External OData Source

Note

The call of the **Request-Reply** step is executed for every order number of the original incoming message. We can't stress enough the importance of this specific behavior of an integration flow making use of the **Splitter** step. All activities following the **Splitter** are invoked for each single message generated by the **Splitter** until either the end of the flow or the **Gather** step is reached! The invocation of the integration flow finally results in the response message shown in [Figure 5.38](#).

To summarize, SAP Cloud Platform Integration solutions frequently have to deal with handling messages containing lists of items. In this section, the **Splitter** step was used to split a message comprising several order numbers into individual single messages, each containing one order number. The individual messages can be treated separately by SAP Cloud Platform Integration. SAP Cloud Platform Integration can then combine the results of each individual message processing chain back into one integrated response message using the **Gather** activity. You've learned how to configure both the **Splitter** and the **Gather** steps correctly to benefit from the message handling behavior just described. In the next section, we'll take a close look at asynchronous message handling scenarios.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
      <multimap:Message1>
        <Orders>
          <Order>
            <ShipPostalCode>51100</ShipPostalCode>
            <ShippedDate>1996-07-16T00:00:00.000</ShippedDate>
            <OrderID>10248</OrderID>
            <ShipCity>Reims</ShipCity>
            <ShipAddress>59 rue de l'Abbaye</ShipAddress>
            <ShipCountry>France</ShipCountry>
            <ShipName>Vins et alcools Chevalier</ShipName>
          </Order>
        </Orders>
        <Orders>
          <Order>
            <ShipPostalCode>44087</ShipPostalCode>
            <ShippedDate>1996-07-10T00:00:00.000</ShippedDate>
            <OrderID>10249</OrderID>
            <ShipCity>Münster</ShipCity>
            <ShipAddress>Luisenstr. 48</ShipAddress>
            <ShipCountry>Germany</ShipCountry>
            <ShipName>Toms Spezialitäten</ShipName>
          </Order>
        </Orders>
        <Orders>
          <Order>
            <ShipPostalCode>05454-876</ShipPostalCode>
            <ShippedDate>1996-07-12T00:00:00.000</ShippedDate>
            <OrderID>10250</OrderID>
            <ShipCity>Rio de Janeiro</ShipCity>
            <ShipAddress>Rua do Paço, 67</ShipAddress>
            <ShipCountry>Brazil</ShipCountry>
            <ShipName>Hanari Carnes</ShipName>
          </Order>
        </Orders>
      </multimap:Message1>
    </multimap:Messages>
  </soap:Body>
</soap:Envelope>

```

Figure 5.38 Result Message, Including Order Details Retrieved from the External OData Source

5.3 Asynchronous Message Handling

The core task of an integration solution is the routing of messages across a company's distributed IT landscape, including connectivity to partners and suppliers. Such integration scenarios can be synchronous or asynchronous in nature. *Synchronous* means, in this regard, the following procedure:

1. A sender opens a connection to SAP Cloud Platform Integration and sends a request message.
2. After sending the request message to SAP Cloud Platform Integration, the sender doesn't close the connection because a reply is expected.
3. SAP Cloud Platform Integration finds the receiver for the respective request message (e.g., by inspecting the message's content), opens a connection to the receiver, and routes the message to the receiver.
4. After sending the message to the receiver, SAP Cloud Platform Integration doesn't close the connection, either.
5. The receiver acts on the request message by creating a response message and returns it to SAP Cloud Platform Integration via the still opened connection.
6. After receiving the message, SAP Cloud Platform Integration will close the connection to the receiver and route the received message as a reply message to the original sender.
7. The sender can now close the connection to SAP Cloud Platform Integration after receiving the final reply message.

The connections from the sender to SAP Cloud Platform Integration, as well as the connection from SAP Cloud Platform Integration to the receiver, are still open, as long as message processing is ongoing. The communication involves a bidirectional message transfer in one session. This message handling procedure significantly differs from *asynchronous* message handling where the procedure looks like the following:

1. A sender opens a connection to SAP Cloud Platform Integration and sends a message.
2. After SAP Cloud Platform Integration receives the message correctly, it acknowledges its reception to the sender.
3. After receiving the acknowledgment from SAP Cloud Platform Integration, the sender closes the connection.
4. SAP Cloud Platform Integration opens a connection to the receiver of the message and forwards it accordingly.
5. After receiving the message completely, the receiver also acknowledges the message's reception to SAP Cloud Platform Integration.
6. After receiving the acknowledgment from the receiver, SAP Cloud Platform Integration closes the connection.

The connections are immediately closed as soon as the messages have been confirmed by the receiving parties. The overall communication only involves a message transfer in one direction. These are, in short, the main differences between synchronous and asynchronous communication. But how does this knowledge affect our discussion of SAP Cloud Platform Integration? First of all, SAP Cloud Platform Integration absolutely must support both communication styles because it is a general-purpose integration infrastructure that is prepared for all kinds of integration requirements. However, in the previous sections, all communications were synchronous. So far, this has been useful because we were able to see the immediate results of our integration flow invocations in the SOAP clients that we used to call the message handling chains. Now the time has come to take a closer look at asynchronous message handling as well, including how to influence the communication style.

5.3.1 Synchronous versus Asynchronous Communication from SAP Cloud Platform Integration's Perspective

Let's repeat some main aspects regarding the synchronous vs. asynchronous discussion from [Chapter 4, Section 4.1.1](#). One of the key terms used in that chapter was *exchange* ([Figure 5.39](#)).

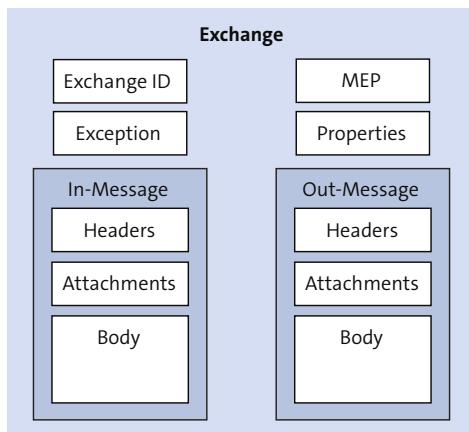


Figure 5.39 Structure of an Exchange

An exchange is a term from Apache Camel terminology that represents a container for a message while it's being processed inside the integration engine. We know so far that the exchange will be filled with an In message only if it's an asynchronous

scenario, whereas the `Out` message within the exchange only plays a role for synchronous scenarios, as discussed in [Chapter 4, Section 4.1.1](#). Additionally, the communication type (synchronous or asynchronous) is determined by the **Message Exchange Pattern** field (MEP) within the exchange. The MEP field can contain two potential values:

- **InOnly**

The route handles a one-way message, and the sender doesn't expect a reply from the respective receiver. Hence, the exchange carries an `In` message only. `InOnly` represents the asynchronous use case.

- **InOut**

The route handles a request-response message. The sender expects a reply from the route which will be stored as an `Out` message in the exchange. `InOut` represents the synchronous communication style.

The component that determines whether a message should be handled synchronously or asynchronously is, in fact, the channel! This might surprise you, but we'll show you how to influence synchronous and asynchronous message handling in SAP Cloud Platform Integration by using the SOAP adapter.

We're reusing the scenario we built in [Section 5.2.4](#). For your convenience, we've added screenshots of the scenario in [Figure 5.40](#) and the associated input message in [Figure 5.41](#).

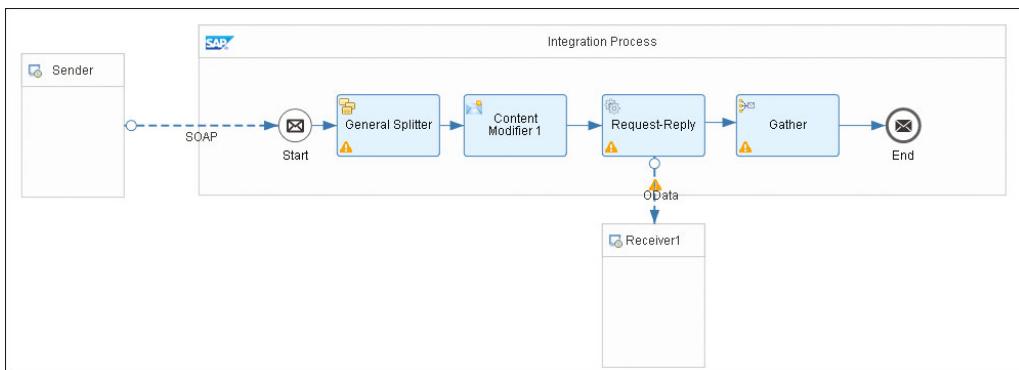


Figure 5.40 Demo Scenario for Splitting and Joining Messages

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
  <soapenv:Body>
    <demo:OrderList_MT>
      <!--Zero or more repetitions:-->
      <orders>
        <orderNumber>10248</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10249</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10250</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
    </demo:OrderList_MT>
  </soapenv:Body>
</soapenv:Envelope>

```

Figure 5.41 Example Input Message for the Demo Scenario

The integration flow, in essence, splits the incoming message into three individual messages. Each of these three single messages contains exactly one order. The two **Content Modifier** steps are executed three times (for each of the single messages). The first **Content Modifier** extracts the order number from the single message and writes it into the message's header area, whereas the second **Content Modifier** retrieves the order number from the header area again and writes the number, embedded inside `orderDetails` tags, into the body area of the message. As this is done three times, the resulting message contains just the three order numbers, each one surrounded by `orderDetails` tags (Figure 5.42).

The screenshots for the input, as well as for the result message, are taken from a SoapUI test client. The scenario is currently a synchronous scenario; otherwise, we wouldn't have received a response back. Mysterious is the fact, that the incoming message was built using a WSDL file containing the description of an *asynchronous* interface. You can verify this by downloading the associated WSDL file called `SendOrderList_Async.wsdl` from www.sap PRESS.com/4650. You'll find the lines in the file shown in Figure 5.43.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <multimap:Messages xmlns:multimap="http://sap.com/xi/XI/SplitAndMerge">
      <multimap:Message1>
        <orderDetails>10248</orderDetails>
        <orderDetails>10249</orderDetails>
        <orderDetails>10250</orderDetails>
      </multimap:Message1>
    </multimap:Messages>
  </soap:Body>
</soap:Envelope>

```

Figure 5.42 Result Message after Invoking the Integration Flow

```

<wsdl:portType name="SendOrderList_Async">
  <wsdl:documentation/>
  <wsdl:operation name="SendOrderList_Async">
    <wsdl:documentation/>
    <wsp:Policy>
      <wsp:PolicyReference URI="#OP_SendOrderList_Async"/>
    </wsp:Policy><wsdl:input message="p1:OrderList_MT"/>
  </wsdl:operation>
</wsdl:portType>

```

Figure 5.43 Definition of the Service's Operation

The operation consists of an input message, but no output message, which would be needed for a synchronous interface. But why, then, is SAP Cloud Platform Integration interpreting it as a synchronous message exchange? Remember the setting for the **Message Exchange Pattern** we've made in the SOAP channel in [Section 5.2.2](#)? Let's take a closer look at the SOAP adapter's configuration ([Figure 5.44](#)).

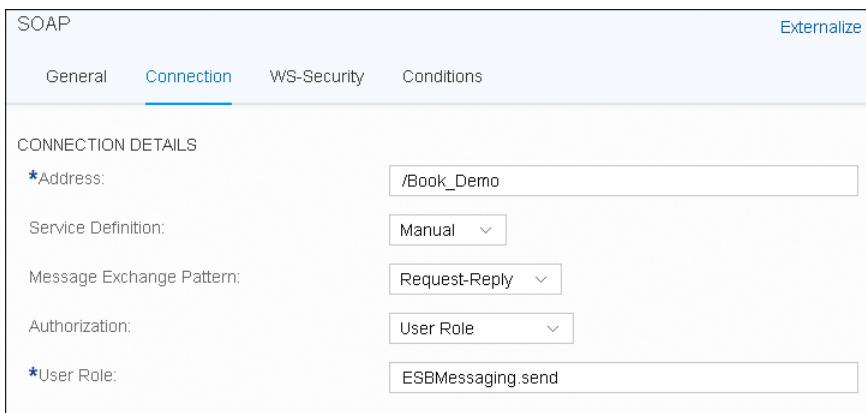


Figure 5.44 Configuration of the SOAP Channel in the Demo Scenario

The **Address** field is set with a string used to create the URL for invoking the integration flow. In addition, we've chosen the default configuration for **Service Definition** and **Message Exchange Pattern**. This is exactly the place where the SAP Cloud Platform Integration's runtime finds the information about synchronous or asynchronous message handling.

When the channel receives a message from a client, it knows nothing about the data that arrives at its address, how the data it receives was constructed, or whether it's based on a synchronous or asynchronous WSDL file. It only knows whether the received message should be treated synchronously or asynchronously because of the configured **Message Exchange Pattern**.

Note

The WSDL file was just used in the SoapUI client to create a proper input message. However, the WSDL file was never used in any of the SAP Cloud Platform Integration configuration steps for that scenario. Hence, SAP Cloud Platform Integration knows nothing about the data it should process. This is the payload-agnostic behavior of Apache Camel we talked about in previous chapters. You can actually push anything to SAP Cloud Platform Integration: it would work as long as you don't have processing steps in your route that rely on a specific format.

We configured **Request-Reply** as **Message Exchange Pattern**; therefore, the SOAP channel sets the **MEP** field in the exchange to `InOut`. This explains the synchronous behavior of our scenario. So, the message walks through the steps of the integration flow, and we have a resulting message created at the end of the chain when we reach the **End** event (see [Figure 5.40](#)). Here, some magic happens: because there is no additional step to process, the last status of the message's body will be copied automatically into the body of the exchange's `Out` message. Remember, synchronous messages in an exchange have both an `In` and an `Out` message, and the reply needs to be in the `Out` message. The `Out` message (including body, headers, and attachments) is finally returned to the caller.

The next issue is how to make the route behave asynchronously. Actually, there are two options to do this: select **One-Way** for the **Message Exchange Pattern** or make the SOAP channel aware of the concrete message it receives.

Let's try out both options. We start with the easy one; we just change the value for the **Message Exchange Pattern** to **One-Way**, as shown in [Figure 5.45](#).

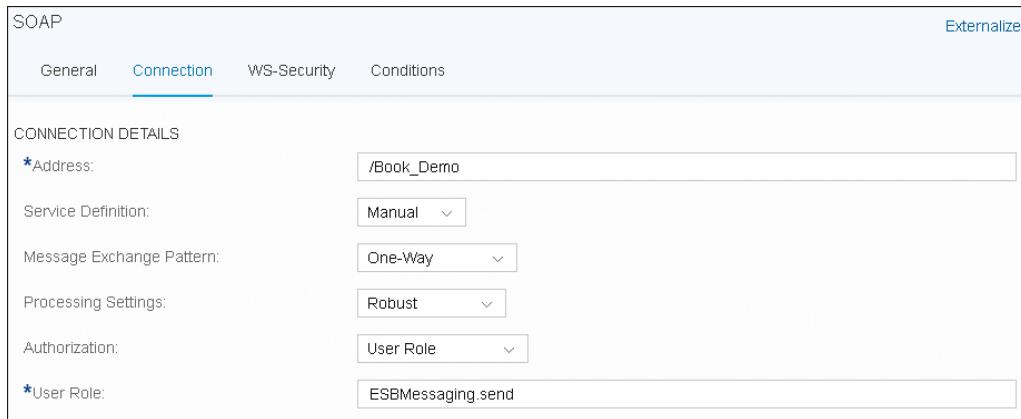


Figure 5.45 Configuration of the SOAP channel after Changing the Message Exchange Pattern Field to One-Way

As soon as we change the setting, a new **Processing Settings** dropdown appears. Two configuration options are available: **WS-Standard** and **Robust**, as described here:

- **WS-Standard**

In the Web Service Standard, one-way processing is also known as the fire-and-forget method. This is useful if the sender of the message just wants to initiate a request but doesn't care if the request is really processed successfully. In SAP Cloud Platform Integration, this behavior can be configured with the **WS-Standard** option. SAP Cloud Platform will process the SOAP request received but won't report back any errors that may occur during processing of the message.

- **Robust**

This is the default option available in SAP Cloud Platform Integration and is also known as Robust In-Only or Robust One-Way. With this configuration, the SOAP adapter synchronously reports back error information to the sender if there is an error during message processing in SAP Cloud Platform Integration or when calling the receiver. This is important for reliable one-way message exchanges, where the sender of a message needs guaranteed delivery by SAP Cloud Platform Integration. This is the option we select because we want to know if the message was processed in SAP Cloud Platform Integration or not.

After the changes are saved and deployed, you can invoke your integration flow again from your SOAP test client of choice. If you're using SoapUI, you'll get nothing back as a reply message. You'll only receive acknowledgment from SAP Cloud Platform Integration about the successful reception of the message as an HTTP response code 202 ([Figure 5.46](#)).

Header	Value
Strict-Transport-Security	max-age=31536000; includeSubDom...
Date	Thu, 19 Nov 2015 17:47:10 GMT
Content-Length	0
#status#	HTTP/1.1 202 Accepted
Set-Cookie	JSESSIONID=D8337A89C2EB478E51AE...
Set-Cookie	JTENANTSESSIONID_a29d0a049=IHQ...
Server	SAP

Headers (7) Attachments (0) SSL Info (3 certs) WSS (0) JMS (0)

Figure 5.46 Returned Header after Invoking an Asynchronous Route via SOAP Channel

Now, let's see if we correctly receive the error information back if there is an error in processing. For this, we need to change the integration flow in a way that an error is raised. This can, for example, be done by setting a wrong **Address** in the **OData** channel retrieving additional information for enriching the message. We set *http://test_error_situation* there, as shown in [Figure 5.47](#), and then save and deploy the integration flow.

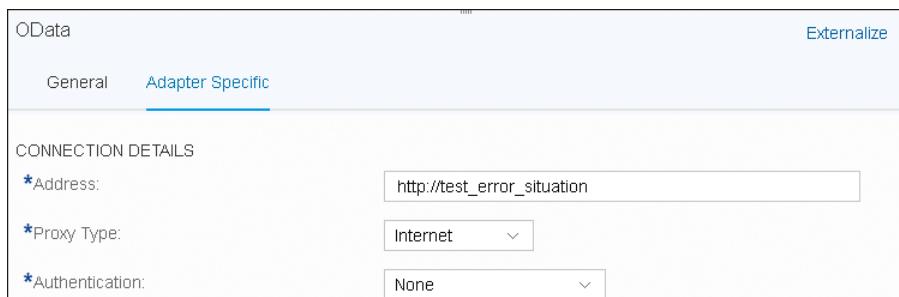


Figure 5.47 Configuration of the OData Channel after Changing the Address

When we now call the integration flow again from our SOAP test client, we receive a SOAP fault error ([Figure 5.48](#)). The SOAP fault contains faultcode and faultstring. Although the faultcode only tells us that an error occurred in the SOAP server, the error details can be found in faultstring.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>An internal error occurred. For error details check MPL ID ... in
      message monitoring or use the URL ... to directly access the error information.</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figure 5.48 SOAP Fault Received in the SOAP Test Client

The real error, that the OData Call wasn't successful, isn't provided; only a generic error containing the message processing log (MPL) ID and the direct link to it in SAP Cloud Platform Integration's message monitoring are provided. We don't see the real error due to security guidelines; no internal details are to be provided externally to avoid attacks. So, we need to call the URL provided to access the MPL in SAP Cloud Platform Integration's message monitoring to investigate the real error (Figure 5.49). Calling the URL is secured by authentication in SAP Cloud Platform Integration so that only authorized users get access to the detailed error description.

```
com.sap.gateway.core.ip.component.odata.exception.OsciException: Host not found : 502 : HTTP/1.1
Request URI          :   GET http://test_error_situation/$metadata HTTP/1.1
Request Headers       :
  x-csrf-token      : fetch

HTTP Status Line     :   HTTP/1.1 502 Host not found
```

Figure 5.49 Error Details Shown in SAP Cloud Platform Integration's Message Monitor

We can now check what happens if **WS-Standard** is set for **Processing Settings** in the SOAP channel. Let's change the configuration (Figure 5.50), and then save and deploy the integration flow.

Invoke the integration flow from your SOAP test client. You now receive the same HTTP response code 202 as in the successful scenario execution (refer to Figure 5.46) because the message was successfully delivered to SAP Cloud Platform Integration, the error isn't given back to the sender—the fire-and-forget behavior explained before. You as sender aren't interested to know whether the message is processed successfully or not.

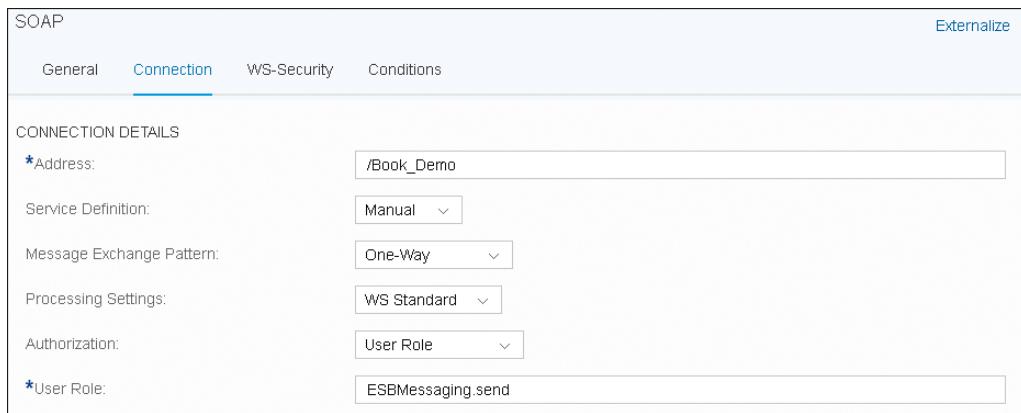


Figure 5.50 Configuration of the SOAP Channel with WS-Standard Processing

Now, let's try out the second option to configure asynchronous processing in the SOAP channel: we make the SOAP channel aware of the concrete message it receives. For this, the configuration of the SOAP channel provides an option to configure the service via WSDL. We switch the configuration for **Service Definition** to **WSDL**. Then, a dedicated field named **URL to WSDL** (see [Figure 5.51](#)) will appear below the **Service Definition** dropdown that allows us to point to our WSDL file.

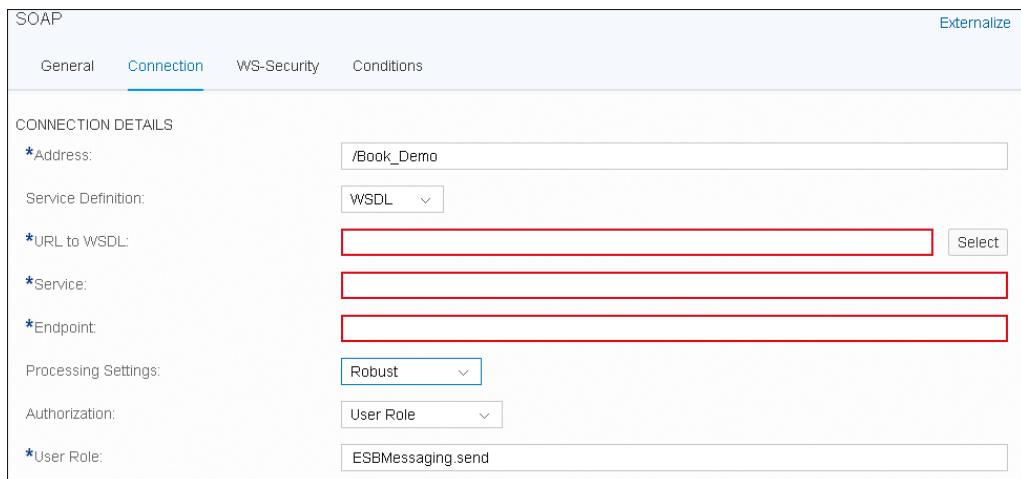


Figure 5.51 Configuration of SOAP Adapter Using Service Definition from WSDL

To assign the WSDL file, simply click on the **Select** button. In the **Select WSDL Resource** dialog box, click on the **Upload from File System** button ([Figure 5.52](#)).

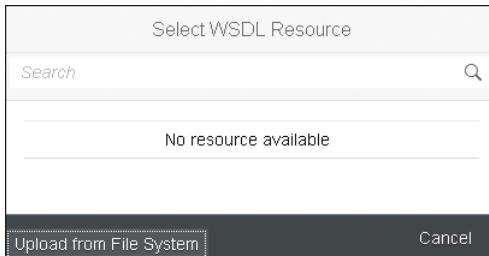


Figure 5.52 Adding a WSDL File to the SOAP Channel

The normal file picker dialog opens. Select the *SendOrderList_Async.wsdl* file from your file system. When uploading the WSDL into the integration flow, SAP Cloud Platform Integration is reading the services defined in the WSDL and automatically sets service and endpoint as defined in the WSDL. In our WSDL file, one service, `SendOrderList_Async`, is defined in the `service` tag, as depicted in [Figure 5.53](#), with the endpoint `SendOrderList_AsyncBinding`.

```
<wsdl:service name="SendOrderList_Async">
  <wsdl:port name="SendOrderList_AsyncBinding" binding="p1:SendOrderList_AsyncBinding">
    <soap:address
      location="https://server:port/cxf/Book\_Demo" />
  </wsdl:port>
</wsdl:service>
```

Figure 5.53 Service Definition in the WSDL

The service and endpoint are taken over into the SOAP adapter configuration, including the link to the WSDL file. The configuration should look like the one shown in [Figure 5.54](#) now. Notice that the namespace `demo:` is added. The namespace definition `xmlns:demo=http://cpi.sap.com/demo` is automatically added into the integration flow's **Runtime Configuration** if not configured already. In our case, the namespace definition was already available because we've configured it earlier.

In addition to the WSDL-specific settings, the **Processing Settings** dropdown is also available for configuration via WSDL. Here, the same options, **Robust** and **WS-Standard**, are available as already described for the configuration without WSDL. We select **Robust** to receive error information back if the message processing in SAP Cloud Platform Integration isn't successful.

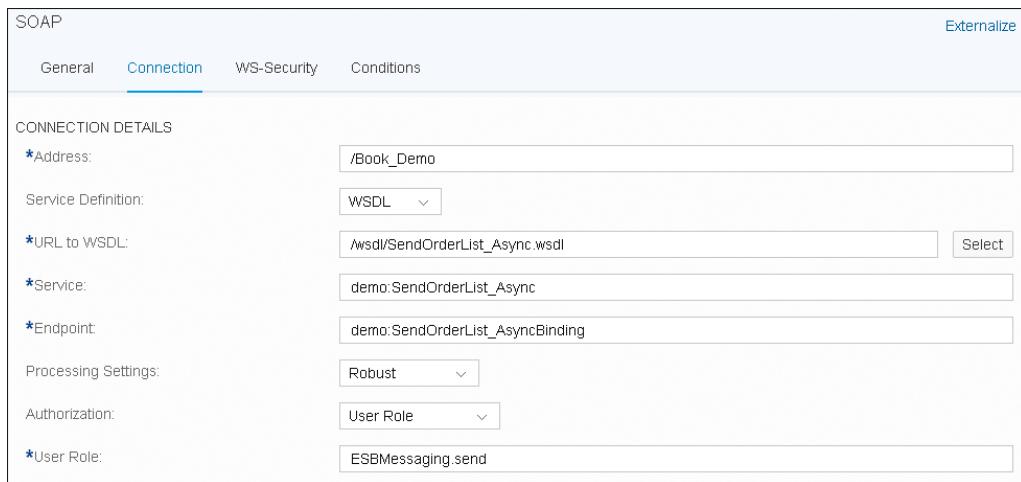


Figure 5.54 Configuration of SOAP Adapter after Assigning the WSDL File

Save and deploy your changes. By adding the WSDL file to the SOAP channel's configuration, correct asynchronous message handling can be ensured. The channel now knows that it receives an asynchronous XML message, compliant to the WSDL, and will set the **MEP** field of the exchange to **InOnly**. However, a validation of the incoming message against the WSDL isn't done for the incoming data. SAP Cloud Platform Integration provides a dedicated processing step for that purpose: the XML validator, which requires the assignment of a WSDL file in its configuration.

Once deployed, you can invoke your integration flow again from your SOAP test client of choice. You'll receive the same HTTP response codes as for the configuration without WSDL. If you still have the wrong address configuration set in the OData channel, you'll get a SOAP fault error because the **Processing Settings** are configured for robust processing (refer to [Figure 5.48](#)). Correct the OData address, and then save and deploy the integration flow. Now you should get the HTTP response code 202 (refer to [Figure 5.46](#)), which means the message processing was successful in SAP Cloud Platform Integration.

But what happened to our message inside SAP Cloud Platform Integration, and where can we track this, now that we don't have SoapUI showing the result? For this, simply navigate to SAP Cloud Platform Integration's monitoring dashboard by clicking on the three horizontal bars in the upper-left corner and choosing **Monitor** from the dropdown menu ([Figure 5.55](#)).

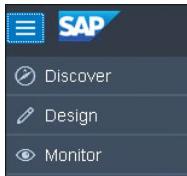


Figure 5.55 Switching to SAP Cloud Platform Integration's Monitoring Environment

The monitoring dashboard opens. Click on the tile for the successfully completed integration flows in the **Monitor Message Processing** area of the screen ([Figure 5.56](#)).

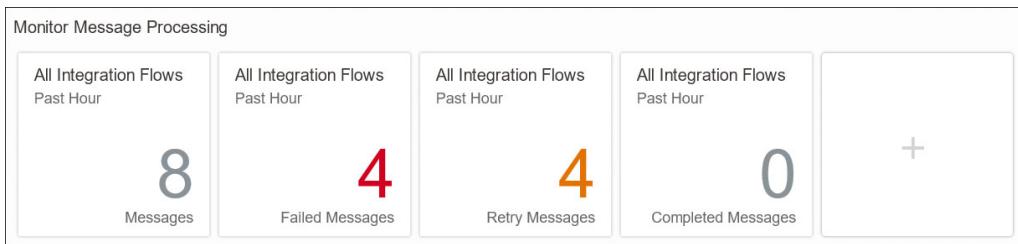


Figure 5.56 Tiles for the Monitors of Completed and Failed Messages

After you've clicked on the tile for completed messages, you'll receive an appropriate list of the successfully completed messages sorted by date and time (see [Figure 5.57](#), left half of the screen). In the right half of the screen, you see the status, logs, and artifact details of the message's processing for the selected message.

If there are errors, you can open the **Failed Messages** tile in [Figure 5.56](#) to see a list of the failed messages. Become familiar with the message processing monitor to use it for root cause analysis in erroneous situations. You'll find additional information about the message monitoring in [Chapter 8](#).

Note

The SOAP channel supports both communication styles: synchronous and asynchronous. You've now seen how to influence its behavior. This isn't the case for all adapters, however. The Secure Shell File Transfer Protocol (SFTP) adapter, for example, supports asynchronous message handling only. Hence, it will merely support `InOnly` as entry in the **MEP** field. Consequently, communication styles are highly adapter dependent and differ from adapter to adapter.

The screenshot shows the Mule Studio interface for managing integration flows. At the top, there are filters for Time (Past Hour), Status (Completed), Artifact (All Integration Flows), and ID (Message or Application ID). Below these, a timeline shows the period from Jan 05, 2018, 13:35:06 to Jan 05, 2018, 14:35:06. A search bar is also present.

The main area displays a list of messages under the heading "Messages (3)". Each message entry includes the artifact name, status, timestamp, and log level. The first message is "Book Integration Flow with Splitter" (Completed at 14:35:01, Log Level: Debug). The second and third messages are also "Book Integration Flow with Splitter" (Completed at 14:34:58 and 14:17:02 respectively, both Log Level: Debug).

To the right of the message list, a specific integration flow is detailed:

- Book Integration Flow with Splitter**
- Last Updated at: Jan 05, 2018, 14:35:01
- Processing Time: 1 sec 103 ms
- STATUS DETAILS: Completed
- LOGS: Message processing completed successfully.
- ARTIFACT DETAILS: Type: Integration Flow, ID: Book_Integration_Flow_with_Splitter, View Integration Flow

Figure 5.57 Details of a Successfully Completed Message

5.3.2 Adding an Asynchronous Receiver

Now that we know how to make a SOAP invocation asynchronous, we probably also want to deliver the message to an asynchronous receiver to really verify the correct content of the received message. To keep administration effort to a minimum, we'll make use of an email receiver. Because the email receiver was already introduced in [Chapter 2](#), won't go into detail here, but we'll show the email adapter's configuration for your reference.

Here are the modeling steps you have to complete to run the scenario with an email receiver:

1. Add a **Receiver** to the model on the right side of the integration flow. You find the receiver besides the participant's node in the palette ([Figure 5.58](#)).
2. Position the **Receiver** on the right side of the **Integration Process**, close to the message **End** event ([Figure 5.59](#)).



Figure 5.58 Selecting a Receiver from the Palette



Figure 5.59 Positioning of the Receiver Pool Close to the End Event

3. Connect the **End** event with the **Receiver** pool ([Figure 5.60](#)).



Figure 5.60 Connecting the End Event with the Receiver Pool

4. In the upcoming **Adapter Type** dialog, select the **Mail** entry. Configure the adapter according to your email account. The values shown in the screenshots ([Figure 5.61](#)) fit a Google email account.

Note

For details on configuration of the fields and creation of the user credentials, refer to [Chapter 2](#).

Mail

Externalize

General Connection Security

CONNECTION DETAILS

*Address: smtp.gmail.com:465

Proxy Type: None

*Timeout (in ms): 30000

*Protection: SMTPS

*Authentication: Plain User/Password

*Credential Name: FirstnameLastname

MAIL ATTRIBUTES

*From: Fn.Ln@gmail.com

*To: Fn.Ln@gmail.com

Cc:

Bcc:

Subject: Book Demo Msg

Mail Body: \${in.body}

*Body Mime-Type: Text/Plain

*Body Encoding: UTF-8

Attachments:

<input type="checkbox"/>	Name	Mime-Type	Source	Header Name
<input type="checkbox"/>	Add Message Attachments			

Add Delete

Figure 5.61 Configuring the Mail Adapter

The final scenario should look similar to the one depicted in Figure 5.62. You can easily identify the sender on the left, the message processing steps in the middle, and the final receiver on the right. That's the advantage of a graphical environment: it clearly and intuitively describes how the message arrives at the server, how it's handled within the SAP Cloud Platform Integration server, and to which systems using which channels it's forwarded.

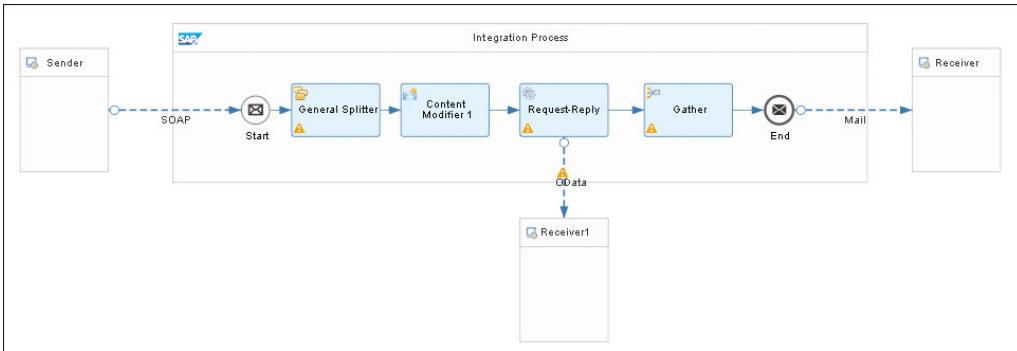


Figure 5.62 Asynchronous Integration Flow with Email Receiver

After you've finished your configurations, you can run the scenario again. This time, the message will be delivered to your email account. You should receive an email containing the three order numbers. A screenshot of the email's content is shown in [Figure 5.63](#).

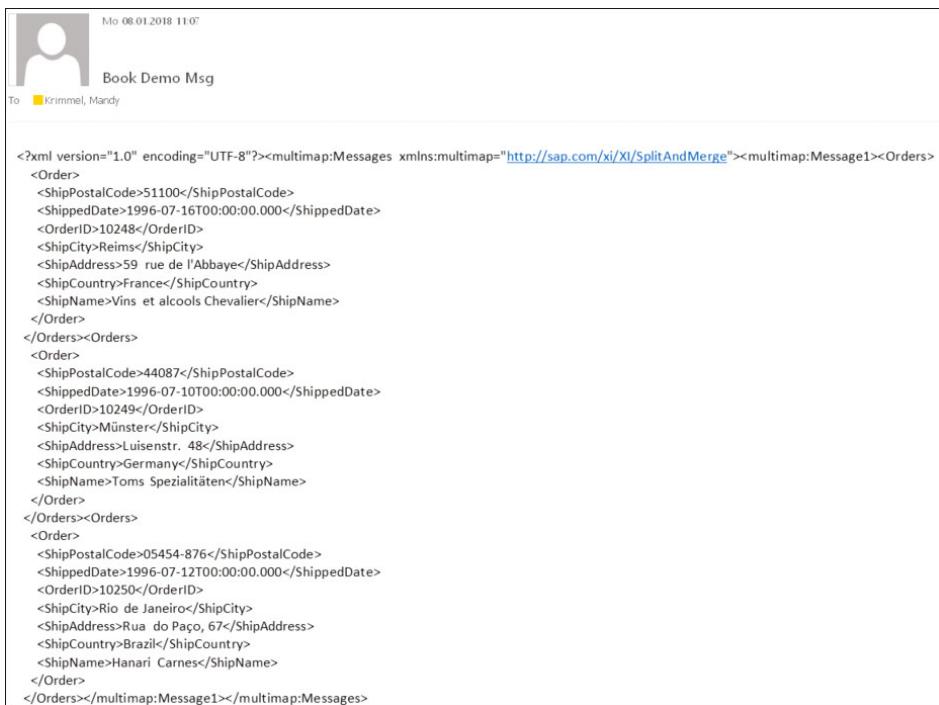


Figure 5.63 Content of Received Email

Note

For analyzing issues connecting to the email server, refer to the troubleshooting information in [Chapter 2, Section 2.3.7](#).

5.3.3 Routing a Message to Multiple Receivers Using the Multicast Pattern

You've now successfully sent the message to one receiver, but some scenarios may require sending the same message to multiple receivers—either in the same or different format—or maybe even only parts of the message. This is also possible in SAP Cloud Platform Integration, so let's configure it now.

We'll use the scenario we set up in the preceding section and add another email receiver, which will just receive an email containing the information that the message was processed successfully.

In [Section 5.1.2](#), we used the **Router** to configure routing messages to multiple receivers. However, we learned that the message will go only to the first receiver, the one for which the routing condition is met, because the **Router** is based on the BPMN-exclusive gateway. Therefore, it doesn't meet the requirements for our use case of sending the message to multiple receivers. To send the message to multiple receivers, we need a flow step based on the BPMN-parallel gateway. For such use cases, SAP Cloud Platform Integration offers the multicast pattern.

Two options are available for the multicast pattern:

- **Parallel Multicast**

In this option, the message is routed to the different branches in parallel. This is the option we want to start with.

- **Sequential Multicast**

In this option, an order of execution can be specified. This is especially important if the second branch won't be executed if the first branch wasn't successful. We'll go into more detail about this option later.

Let's start modeling. To add the **Multicast** option, from the main menu of the palette, select the **Message Routing**  shape. A submenu opens, revealing different routing options, including **Multicast** ([Figure 5.64](#)).

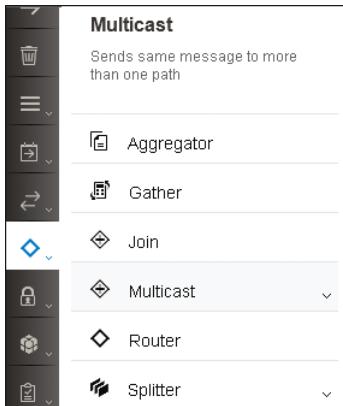


Figure 5.64 Multicast Shape in the Modeling Environment Palette

When you select **Multicast**, another submenu opens providing the two options of **Sequential Multicast** and **Parallel Multicast** (Figure 5.65). Select the **Parallel Multicast** option for now, and place it between **Gather** and the **End** event, as depicted in Figure 5.66.



Figure 5.65 Multicast Options in the Modeling Environment Palette

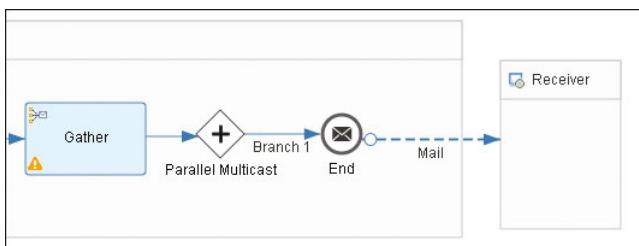


Figure 5.66 Placing a Parallel Multicast between Gather and the End Event

To send the message to a second receiver, we need to add another **End Message** event and one more **Receiver**. To add another branch for the multicast processing, draw a line between **Parallel Multicast** and the **End** event. Then, connect the **End** event with the new **Receiver**, select the **Mail** adapter, and configure it (as shown in [Figure 5.67](#)). We enter “Status of Message Processing” as the **Subject** and add a short status text in the **Mail Body** entry field. Notice, that we add the SAP header for the MPL ID, `SAP_MessageProcessingLogID`, into the email’s body. This value will help later when searching for the message in the SAP Cloud Platform Integration’s message monitoring.



Figure 5.67 Configuring the Mail Adapter for the Second Receiver

The scenario should now look similar to the one depicted in [Figure 5.68](#). Two branches leave from the **Parallel Multicast** shape: **Branch 1** to the first mail receiver and **Branch 2** to the newly added mail receiver.

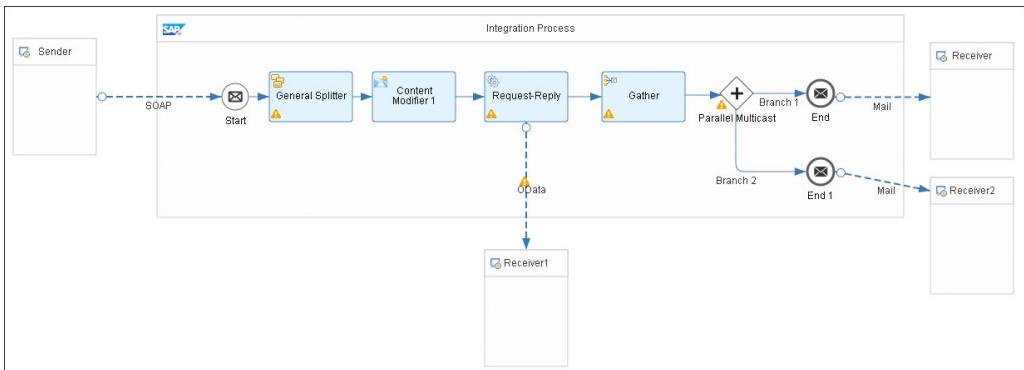


Figure 5.68 Integration Flow with Parallel Multicast and Two Receivers

Note

In each of the branches leaving the multicast shape, you can configure additional flow steps, for example, mappings or converters. With this, you can send different messages or different parts of the inbound message to specific receivers.

After you've finished your configurations, saved, and deployed the integration flow, run the scenario again. From the SOAP test client, the request will be sent successfully to SAP Cloud Platform Integration, and this time the message will be delivered twice to your email account: once with the message created by the **Gather** step (Figure 5.63) and the second time with the status mail text configured in the second **Mail** adapter channel. The received email's content of the status is shown in Figure 5.69.



Figure 5.69 Content of Received Email Containing the Message Processing Status

In the email containing the processing status, we get the **MPL ID** for the message's processing in SAP Cloud Platform. With this information, we can easily check the processing and status in the message monitor by searching for the MPL ID, if required. To do that, open the **Monitoring** page as described in the previous section, and select the **All Messages** tile. In the search criteria, paste the MPL ID from your email into the **ID** field, as shown in Figure 5.70. Press **Enter**, or click the magnifier icon next to the added MPL ID to start the search. The search opens the **Message Processing** details of your message.

Artifact Name	Status
Book Integration Flow with Splitter	Completed
Jan 08, 2018, 13:35:30	Log Level: Debug

Book Integration Flow with Splitter
Last Updated at: Jan 08, 2018, 13:35:30
Processing Time: 3 sec 330 ms

Status Details Logs Artifact Details

Message processing completed successfully.

Figure 5.70 Searching for a Specific MPL ID in the Message Monitor

But what happens if Branch 1 could not be completed because of an error? Let's try it out. To produce an error in Branch 1, we change the **Address** field in the first **Mail** adapter channel to some configuration that doesn't work, for example, to `smtp.gmail.com:444`. Save and deploy the integration flow, and run the scenario again.

Now we get an HTTP 500 error in the SOAP test client because the message could not be processed successfully in SAP Cloud Platform Integration. This is what you probably expected, but we still get the email that the message processing was successful. Let's take a detailed look at the message processing of this message in SAP Cloud Platform Integration. We use the MPL ID received in the email as a search criteria in the **ID** field in message monitor, as shown in [Figure 5.71](#).

We see that the message processing of the message has the status **Failed**, and we also see the reason: connection to `smtp.gmail.com:444` wasn't possible. But why do we still get the status email, if the message processing is failing?

We need to understand the processing of a message in the multicast pattern in SAP Cloud Platform Integration. We already discussed the inbound processing in the SOAP adapter, in which the message is processed by each of the configured flow steps one after the other. But as soon as the message reaches the **Multicast** step, it's multiplied as many times as outbound branches leave the **Multicast**. Each of these messages is processed independently in its own branch until it reaches an **End** event or a **Join** step.

The screenshot shows the SAP Cloud Platform Integration Message Monitor interface. At the top, there are filters for Time (Past 24 Hours), Status (All), Artifact (All Integration Flows), and an ID search bar containing `AFpTowFETIu7q4a8gAjUDSihedvu`. Below the filters, a table lists a single message entry:

Artifact Name	Status
Book Integration Flow with Splitter	Failed

Details for the failed message:

- Last Updated at: Jan 08, 2018, 17:58:09
- Log Level: Info
- Processing Time: 31 sec 963 ms

Below the message table, there are tabs for STATUS DETAILS, LOGS, and ARTIFACT DETAILS. The STATUS DETAILS tab is active, showing a red box with the message "Message processing failed." Under the ARTIFACT DETAILS tab, there is an "Error Details" section with the following text:

```
com.sun.mail.util.MailConnectException: Couldn't connect to host, port:  
smtp.gmail.com, 444; timeout 30000;
```

Figure 5.71 Searching for the MPL ID of the Failed Message in the Message Monitor

For our scenario, this tells us that our two branches leaving the parallel multicast are executed in parallel and are completely independent of each other. So, **Branch 2** is executed even if **Branch 1** fails. But this isn't what we want to have in our scenario; instead, we want to get the successful status email only when the first call was successful. We need to make sure the branches are executed one after the other and ensure that the second branch is executed after the first branch. In SAP Cloud Platform Integration, we can use the second multicast option for this: the **Sequential Multicast**.

Combination of Multicast with Join and Gather

In this chapter, we use the **Multicast** pattern to send messages to multiple receivers. However, in SAP Cloud Platform Integration, there is another use case for it; that is, in combination with **Join** and **Gather**, the **Multicast** step can be used to process the message in different branches; later bring the branches together using the **Join**, and combine the messages into a single message using **Gather**. This would look similar to the processing depicted in [Figure 5.72](#). For more details about the detailed configuration, refer to the documentation for SAP Cloud Platform Integration (found at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud) and search for “Defining Join and Gather.”

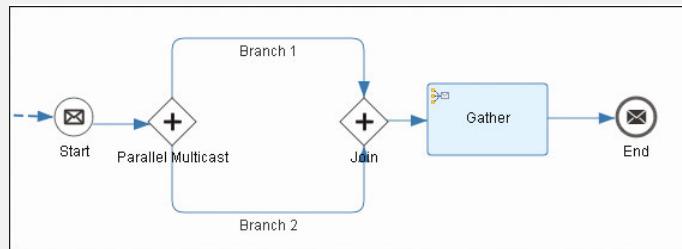


Figure 5.72 Using Multicast with Join and Gather

Now let's change the integration flow by removing the **Parallel Multicast** and adding the **Sequential Multicast** instead. Afterwards, we connect **Gather** with **Sequential Multicast** and **Sequential Multicast** with the two **End** events connected to the two **Receivers** ([Figure 5.73](#)).

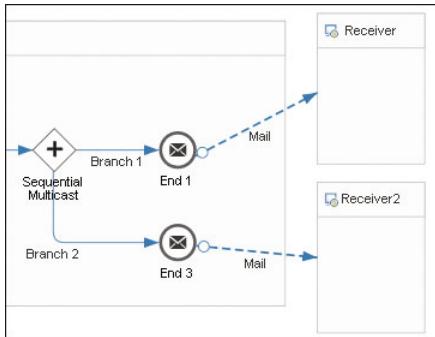


Figure 5.73 Sequential Multicast Connected with two Receivers

We need to make sure that the two branches, **Branch 1** and **Branch 2**, are executed in the correct order. First, **Branch 1** will be processed by sending out the message produced by the **Gather** step, and, if this is executed successfully, **Branch 2** will be processed. To check and, if necessary, change the order, we select the **Sequential Multicast** step to get its details in the **Properties** section. In the **Routing Sequence** tab, a table with the configured branches is shown (Figure 5.74). Make sure the branch to the first **Receiver** (in our integration flow, it's **Branch 1**) is at the top of the table to get executed first. If this isn't the case, the order can be changed with the **Move Up** and **Move Down** buttons.

If the configuration is done, save and deploy the integration flow. Trigger the scenario again from your SOAP test client. You'll still get the HTTP 500 error, but you won't receive the status email anymore.

Sequential Multicast		Externalize
General	Routing Sequence	
Specify the sequence of execution of branches:		Move Up Move Down
Sequence Order	Sequence Name	
1	Branch 1	
2	Branch 2	

Figure 5.74 Configure Order in Sequential Multicast

To check the successful execution, correct the **Address** in the first **Mail** adapter channel, save, and deploy the change. The execution now correctly sends the message produced by **Gather** and the status email to your email account.

Now we know two different configuration options for the parallel gateway processing in SAP Cloud Platform Integration. But there is even a third, very special option to configure this specific multicast scenario. If, like in our case, no additional configuration step is to be executed in the first multicast branch, we can use the **Send** step instead.

Let's configure the scenario with a **Send** step by adding the **Send** step from the palette in your integration flow after the **Gather** step. You find the **Send** step under the **Call**  shape. Select the shape, and a submenu with the two options, **External CALL** and **Local Call**, opens ([Figure 5.75](#)). Select the **External Call** to see three options for doing external calls from SAP Cloud Platform Integration: **Request-Reply**, **Send**, and **Content Enricher** ([Figure 5.76](#)). **Request-Reply** and **Content Enricher** were already explained in some detail, so we focus now only on the **Send** step.

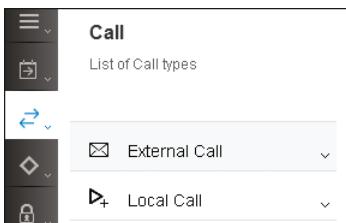


Figure 5.75 Call Shape in the Modeling Environment Palette

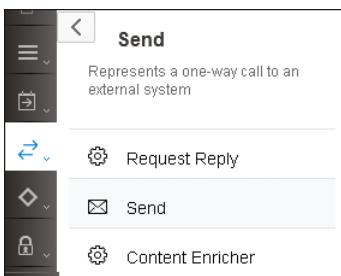


Figure 5.76 External Call Shape in the Modeling Environment Palette

Connect the **Gather** step with the **Send** step, and draw a line from the **Send** step to the first **Receiver**, which is expecting the message coming from **Gather**. Select the **Mail**

adapter, and reconfigure the **Mail** channel like it was configured in the integration flow using **Multicast** (refer to [Figure 5.61](#)). Connect the **Send** step with the **End** event. Your integration flow should now look similar to [Figure 5.77](#).

Save the configurations, and deploy the integration flow. Triggering the flow from your SOAP test client will produce the same result as when calling the integration flow with **Sequential Multicast**: the message will be sent successfully, and the two emails, one created by the **Gather** step and the status email, will be received in your mailbox.

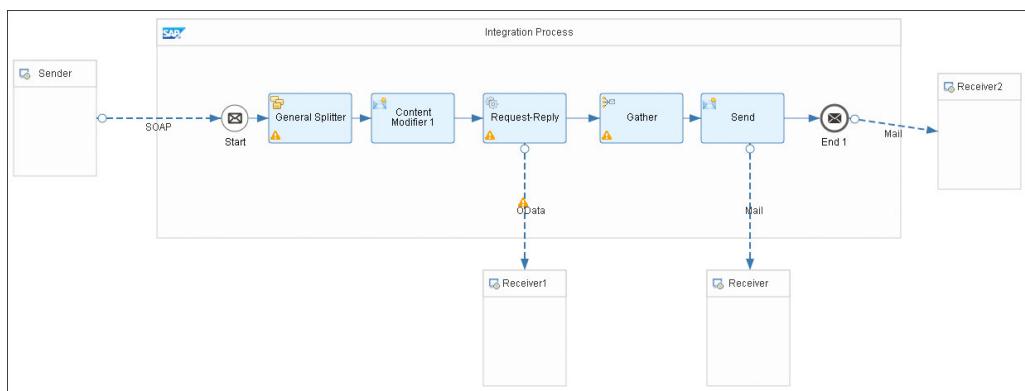


Figure 5.77 Integration Flow with Send Step and Additional Receiver

If during runtime, the sending of the email via the **Send** step fails, the processing will stop, and the status email won't be sent. Therefore, in the runtime, the **Send** step behaves like the **Sequential Multicast**. In fact, the **Send** step is a sequential multicast with exactly two branches: the first branch goes to the connected **Receiver**, and the second branch goes to the next processing step.

Synchronous and asynchronous message handling procedures are at the core of every integration solution, and SAP Cloud Platform Integration is no exception from this rule. In this section, you learned more about the internal message processing details of SAP Cloud Platform Integration. You've seen how synchronous and asynchronous message handling can be controlled for the SOAP channel and how multiple asynchronous receivers can be added to your scenario using the multicast pattern. The message monitor was finally used to help you track the message processing within SAP Cloud Platform Integration. In the next section, we'll go one step further and decouple inbound and outbound processing of the integration flow to make the scenario more robust for potential errors during message processing.

5.4 Reliable Messaging Using the JMS Adapter

In the previous section, you learned how asynchronous messaging can be configured in SAP Cloud Platform Integration, the basic concept behind it, and the different modeling techniques to send the message to multiple receivers. You've used robust one-way communication to make the asynchronous scenario reliable to ensure the message delivery is guaranteed and the sender is notified about errors to be able to resend the erroneous messages.

Following are the characteristics of such robust one-way scenarios using SAP Cloud Platform Integration:

- This pattern relies on a retry mechanism in the sender system. If there is an error during processing in SAP Cloud Platform Integration, for example, if one of the receivers isn't available temporarily, the error will be reported back to the sender, and the whole message processing needs to start from the beginning. Meaning it needs to be triggered by the sender system.
- If the parallel multicast is used, temporary connection issues associated with one of the receiver systems trigger retries from the sender system and may lead to duplicate messages in the other receiver systems.
- The overall execution time of an asynchronous reliable messaging scenario includes the whole message's processing in SAP Cloud Platform Integration until the message finally reaches the receiver. If there is a complex integration flow, the execution time may become undesirably long.

This isn't the desired behavior in many scenarios; often the sender wants to deliver the message and expects an asynchronous response later—and doesn't want to wait or care about retries. The sender usually wants to send the message and expects the middleware to handle temporary connection problems to receiver systems using built-in retry mechanisms. In this section, we'll extend the scenario developed in the preceding section in such a way that a retry is triggered from SAP Cloud Platform Integration automatically if there are execution errors.

5.4.1 Asynchronous Decoupling of Inbound Communication

To trigger the retries from SAP Cloud Platform Integration if there is an error during message processing, we need to decouple the inbound message processing in SAP Cloud Platform Integration from outbound processing, and we need to temporarily persist the message in SAP Cloud Platform Integration to be able to restart the

message from SAP Cloud Platform Integration's persistency if an error happens during processing.

The next section discusses which SAP Cloud Platform Integration options exist to temporarily persist messages, how to configure such scenarios, and what monitoring options we can use.

Data Store and Java Message Service Queues

SAP Cloud Platform Integration provides two options—data store and JMS queues—to temporarily store messages during message processing:

- **Data store**

Based on the database SAP Cloud Platform Integration is running on, data store is used to store messages using the **Write** step and subsequently read from data store using the **Select** or **Get** step. Because the data store isn't designed to execute high-performance messaging, it isn't the optimal solution for our scenario. The main use cases for the data store are as follows:

- You need to temporarily store a message during message processing because the original payload is required later in the processing of the message. With the **Write** step, the initial message is saved in the data store. Afterwards, changes to the payload are made, for example, using mappings, converters, or scripts. Later in the processing, the initial message is fetched using the **Get** step.
- The data store is often used in the push-pull pattern. The sender is sending messages to SAP Cloud Platform Integration, where the messages are temporarily stored in the data store. In a synchronous call, the receiver actively polls the messages from the data store using the **Select** step and acknowledges the receipt of each message. Afterwards, these acknowledged messages are deleted from the data store using the **Delete** step. We don't go into a detailed description of the push-pull pattern here, but a small sample integration flow is shown for your reference in [Figure 5.78](#).

- **JMS queues**

JMS queues are the second option to temporarily store messages in SAP Cloud Platform Integration. Messages are stored in JMS queues using the **JMS Receiver** adapter and read from JMS queues using the **JMS sender** adapter. Because JMS supports high-speed messaging with high throughput, it offers the optimal solution for reliable messaging using asynchronous decoupling. So, that is the option we go for in our scenario.

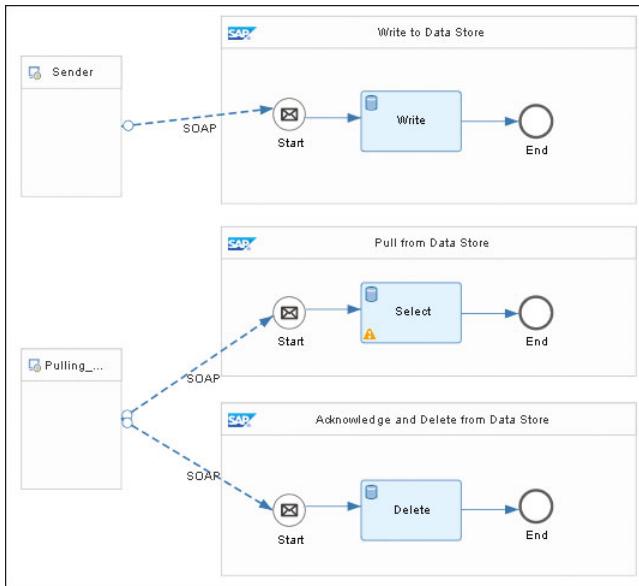


Figure 5.78 Sample Integration Flow for the Push-Pull Pattern

JMS

JMS is a Java-based standard application programming interface (API) for sending and receiving messages. It allows a reliable asynchronous communication between different components based on a JMS Message Broker. The JMS Message Broker is a separate runtime component that ensures the handling of the messages in JMS queues in the JMS Message Broker. The JMS adapter available in SAP Cloud Platform Integration is used to store messages in the JMS queue and to consume messages from the JMS queue in the JMS Message Broker.

The processing sequence used by the JMS adapter is first-in, first-out (FIFO), which means messages stored in the JMS queue latest are also consumed latest. But be aware that this doesn't mean that the messages are processed in a guaranteed order. This is because SAP Cloud Platform Integration uses several runtime nodes consuming messages from the JMS queue in parallel. Furthermore, messages that cause errors are taken out of the processing and are retried later according to the retry interval defined in the JMS channel.

To decouple the inbound processing of the integration scenario from the outbound processing, we need to split the scenario into two processes: one for receiving the

message and storing it in a JMS queue and a second process for consuming the message from the JMS queue and further processing the message and sending it to the receiver.

Prerequisites for Using the JMS Adapter

JMS messaging is available only with the SAP Cloud Platform Integration, Enterprise Edition, or if a JMS messaging license is purchased separately. If your SAP Cloud Platform Integration system isn't running with the Enterprise license and no JMS messaging license is purchased, the JMS adapter doesn't appear in the list of available adapters.

Furthermore, you need to get a JMS Message Broker provisioned for your SAP Cloud Platform Integration tenant. The provisioning is triggered using a self-service in the account cockpit. Details about the provisioning of a JMS Message Broker can be found in the documentation for SAP Cloud Platform Integration (https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud) and in the “Provision Message Broker” blog in the SAP Community (www.sap.com/community.html).

Extending the Integration Scenario

Let's start extending the scenario we configured in the previous section. As a first step, to decouple the inbound processing from all the other processing steps, we create and configure a new integration process for the inbound processing by making the following changes to the integration flow:

1. Add a new **Integration Process** below the existing integration process by selecting it from the modeling palette under the **Process**  shape. In the submenu, the **Integration Process** is one of the process elements available (Figure 5.79).

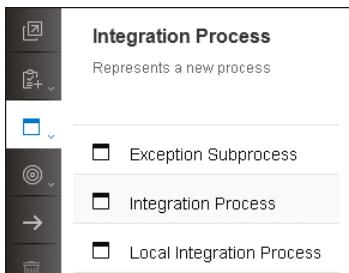


Figure 5.79 Process Shape in the Modeling Environment Palette

2. In the new integration process, add a **Start Message** event and an **End Message** event from the **Events**  shape (Figure 5.80). Connect the **Start Message** event with the **End Message** event.

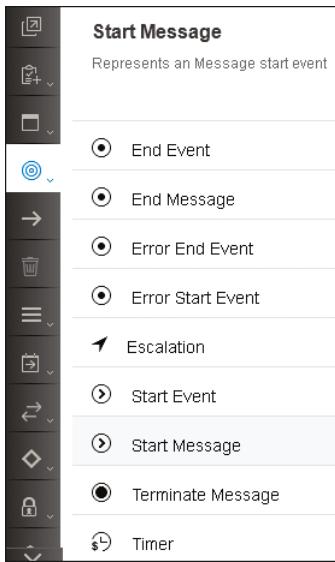


Figure 5.80 Events Shape in the Modeling Environment Palette

3. Select the dotted line representing the message flow from the **Sender** to the first integration process's **Start** event. The line gets orange with small blue dots at the start and the end of the line. Select the blue dot appearing at the **Start** event in the first **Integration Process**, and move it to the **Start** event in the new **Integration Process** (Figure 5.81). Now the existing **SOAP** channel points to the new integration process; you don't have to remodel the channel.
4. Add a **Receiver** to the model on the right side of new the integration process. Remember, you find the **Receiver** under the **Participant** shape  in the palette.
5. Connect the **End** event of the new **Integration Process** to the new **Receiver**. In the adapter selection screen choose the **JMS** adapter. (Note that the JMS adapter entry will only appear in the list if you're using SAP Cloud Platform Integration, Enterprise Edition.)

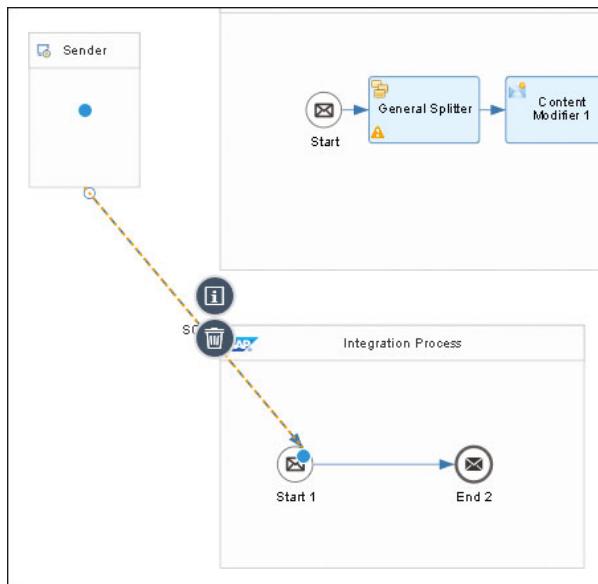


Figure 5.81 Connecting the SOAP Sender Channel to the New Integration Process

6. In the **JMS** channel's **Processing** tab, configure the following properties, as shown in [Figure 5.82](#):

- **Queue Name:** As the most important configuration setting, we specify a unique name for the JMS queue to be used for this scenario, that is, `Inbound_Queue`. The JMS queue is used by the **JMS** receiver adapter to store the message in and by the **JMS** sender adapter to consume the messages from. You'll see this when we configure the **JMS** sender later.
- **Retention Threshold for Alerting:** You set the number of days the message should be picked up from the JMS queue and further processed. If the message stays in the JMS queue for a longer time than configured, the message appears as **Overdue** in the Monitor to indicate that there is some problem during processing: maybe the consuming integration flow doesn't run anymore.
- **Expiration Period:** There are only limited resources available on the JMS instance configured for a SAP Cloud Platform Integration system. Therefore, JMS queues are only allowed for temporary storage. In the **Expiration Period** field, you define after how many days the message is automatically deleted from the JMS queue. The default value is 90 days, and the maximum value allowed is 180 days, but keep in mind that there is a maximum capacity allowed

in the JMS instance for all messages in all JMS queues in your SAP Cloud Platform Integration system. This maximum capacity is based on the number of messaging licenses purchased. The available and the used capacity are shown in the queue monitor. We'll see this later, when we run the scenario.

- **Encrypt Stored Message:** This checkbox should be enabled for security reasons to encrypt the messages in the JMS queue. For a scenario processing messages that aren't security relevant, the checkbox can be disabled to improve performance.
- **Transfer Exchange Properties:** Exchange properties are usually not transferred using JMS queues. If, for any scenario-specific reason, transferring them is required, the **Transfer Exchange Properties** checkbox can be enabled. But take into consideration that there is a hard limit of 4 MB for storing headers and properties in JMS queues. If the limit is hit during runtime, the message will go into **Failed** status with an error stating that the limit for headers and properties is reached.

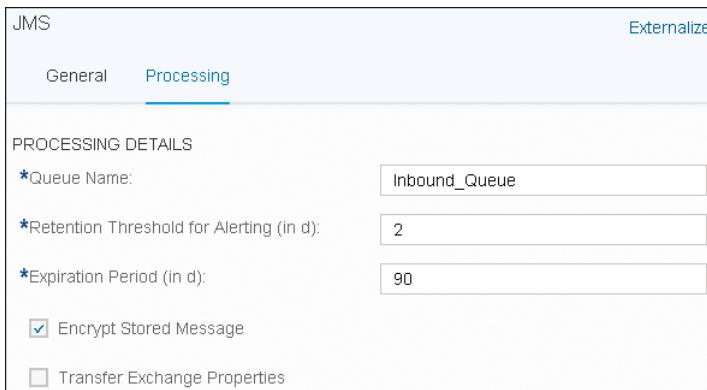


Figure 5.82 JMS Receiver Channel Configuration

Limits for JMS Resources

There are limited resources available on the JMS instance connected to the SAP Cloud Platform Integration tenant. The JMS resources available depend on the licenses purchased.

One hard limit is the overall size of 4 MB for storing headers and properties in a JMS queue. This limit can neither be configured nor increased, not even by purchasing

additional licenses for JMS messaging. Our recommendation is to not transfer properties via JMS queues and to restrict the usage of headers.

More details about the JMS resources can be found in the “JMS Resource and Size Limits in CPI Enterprise Edition” blog in the SAP Community (www.sap.com/community.html).

Now the new integration process looks similar to the one in [Figure 5.83](#).

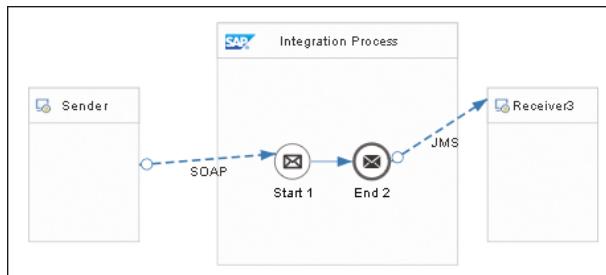


Figure 5.83 Integration Process with JMS Receiver

We've now changed the inbound processing in a way that the message isn't processed by SAP Cloud Platform Integration and sent to the receiver. Instead of this, after being received, the message is directly stored in a JMS queue. Let's run the scenario before we configure the main integration process that will consume the messages from the JMS queue.

Save and deploy the integration flow. Now trigger the integration flow from the SOAP test client. You'll notice that the HTTP response code 202 notifying you about successful execution is shown faster than in the executions triggered in the previous section. This is because with the change made in the integration flow, the message is just stored in the JMS queue, and no further processing is executed in SAP Cloud Platform Integration.

The message is now stored in the JMS queue waiting for consumption and further processing. You can monitor the JMS queues in the SAP Cloud Platform Integration's monitoring dashboard to locate it. Open the dashboard, in the **Manage Stores** area of the screen, you find the **Message Queues** monitor ([Figure 5.84](#)).

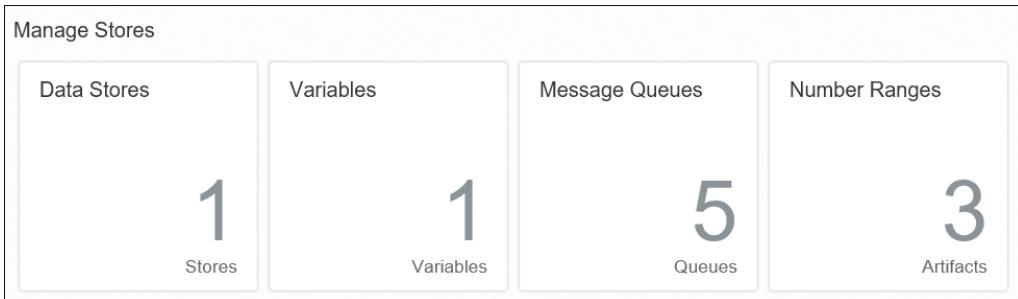


Figure 5.84 Manage Stores Section in the Monitoring Dashboard

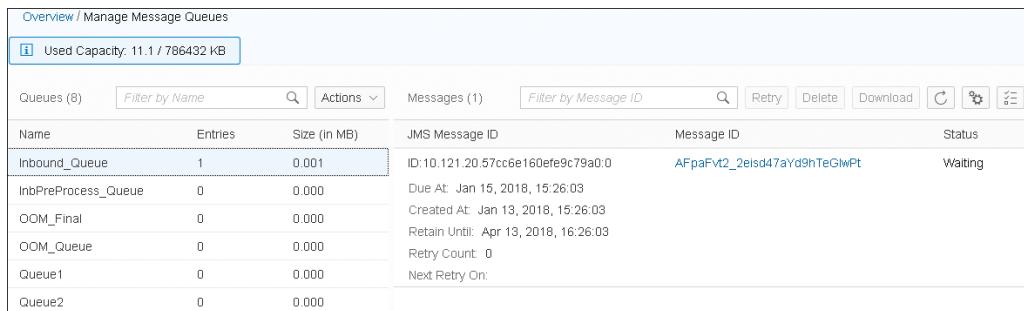
Queue Creation during Deployment

The JMS queues are created automatically in the JMS messaging instance during deployment of the first integration flow using a new JMS queue name. Be aware of the following two aspects:

- If the same **Queue Name** is used in multiple integrations flows or integration processes, the same JMS queue is used in runtime. This means multiple processes may write to the same JMS queue, or even multiple JMS senders may consume from the same JMS queue. The integration developer may want this to happen if the same queue will be reused by different scenarios, but it may also happen by accident.
- JMS queues aren't deleted automatically during undeployment of the integration flow because there may still be messages in the JMS queue, and deleting the queue would delete them as well, which would lead to data loss. Only the owner of the scenario knows if the JMS queue can be deleted with all its content or if it's still required. A JMS queue can be deleted in the **Manage Message Queues** monitor using the **Delete** action.

Select the **Message Queues** monitor to get the list of JMS queues created in the JMS messaging instance of your SAP Cloud Platform Integration system. Search for the JMS queue with the name **Inbound_Queue**, which was the queue name we defined in the JMS receiver channel. Select the queue to get the messages in this specific queue displayed ([Figure 5.85](#)).

You notice that the message sent to SAP Cloud Platform Integration is displayed with the status **Waiting**. This status indicates that the message is waiting to be consumed by another process.



The screenshot shows a table titled 'Overview / Manage Message Queues'. At the top, it displays 'Used Capacity: 11.1 / 786432 KB'. Below this, there are two tabs: 'Queues (8)' and 'Messages (1)'. The 'Messages (1)' tab is selected, showing a single message entry. The message details are as follows:

Name	Entries	Size (in MB)	JMS Message ID	Message ID	Status
Inbound_Queue	1	0.001	ID: 10.121.20.57cc6e160efef9c79a0:0	AFpaFvQ_2eisd47aYd9hTeGhPf	Waiting
InbPreProcess_Queue	0	0.000		Due At: Jan 15, 2018, 15:26:03	
OOM_Final	0	0.000		Created At: Jan 13, 2018, 15:26:03	
OOM_Queue	0	0.000		Retain Until: Apr 13, 2018, 16:26:03	
Queue1	0	0.000		Retry Count: 0	
Queue2	0	0.000		Next Retry On:	

Figure 5.85 Message in Waiting Status in the Manage Message Queues Monitor

Configure the Integration Process

Now, let's configure the integration process, which consumes and finally sends the message to the receiver:

1. Add a **Sender** to the model on the left side of the integration process configuring the main processing. Remember, you find the **Sender** under the **Participant** shape  in the palette.
2. Connect the **Sender** to the **Start** event of the integration process. In the adapter selection screen, choose the **JMS** adapter.
3. In the **JMS** channel's **Processing** tab, configure the properties as shown in [Figure 5.86](#):
 - **Queue Name:** Define the same value as specified in the JMS receiver (**Inbound_Queue**) to consume the messages from the JMS queue that the inbound processing stores the messages in.
 - **Number of Concurrent Processes:** Messages from the JMS queue can be consumed in parallel if required for high message throughput. In the **Number of Concurrent Processes** field, you specify the number of parallel processes consuming messages from the JMS queue.
 - **RETRY DETAILS:** The retry configuration is specified in this section, as follows:
 - **Retry Interval (in min):** Define after how many minutes the first retry is executed.
 - **Exponential Backoff:** Set this checkbox to avoid lots of retries in a short time. When **Exponential Backoff** is set the retry interval is doubled after each unsuccessful retry. For example, if a receiver system isn't available because of maintenance, it doesn't make sense to retry the message every second.

- **Maximum Retry Interval (in min):** Define the maximum interval between two tries to avoid an endless increase of the retry interval if **Exponential Backoff** is used.
- **Dead-Letter Queue:** Take those messages out of processing that cause outages of the runtime node, for example, by an out-of-memory error. Messages where processing stopped unexpectedly are retried only twice when the **Dead-Letter Queue** checkbox is enabled. Further details about this feature, its usage, and its monitoring can be found in the “Configure Dead Letter Handling in JMS Adapter” blog in the SAP Community (www.sap.com/community.html).

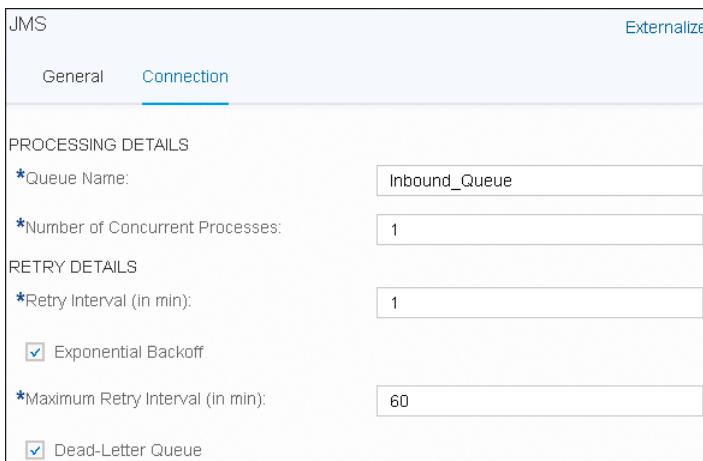


Figure 5.86 JMS Sender Channel Configuration

The integration flow with the two integration processes now looks similar to [Figure 5.87](#).

Retry from JMS Queue

Messages in JMS queues are retried as long as the messages aren't successfully processed or the expiration period is reached. As soon as the message goes to **Completed** status, the message is removed from the JMS queue. The message can either get **Completed** after successful execution of the scenario or if the message processing error is caught in an **Exception Subprocess**. The second option will be discussed later in this chapter.

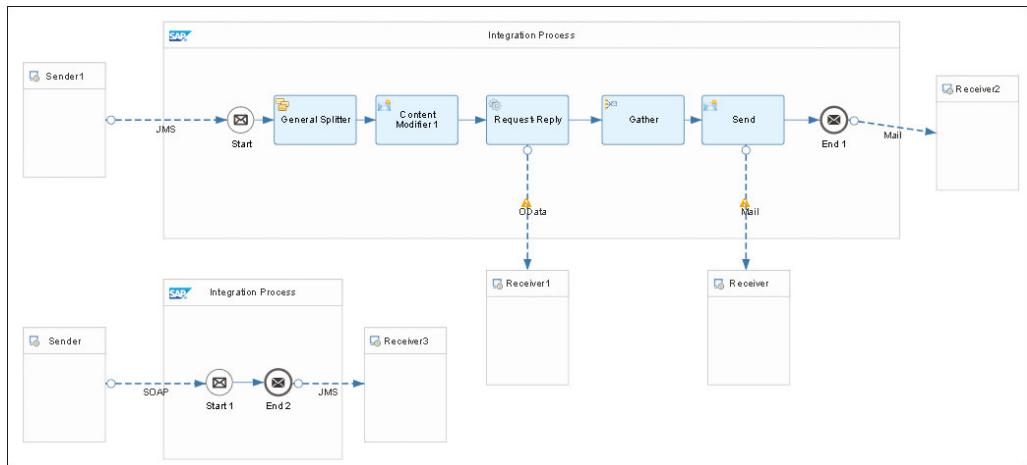


Figure 5.87 Integration Flow with JMS Sender and JMS Receiver Adapter

Save your changes and deploy the integration flow. The JMS sender channel now consumes the message from the JMS queue, and you get the missing emails. Triggering the integration flow again from your SOAP test client, you'll receive the emails directly because storing the message to the JMS queue and consuming it from there doesn't require much time in SAP Cloud Platform Integration.

You've seen now that for successfully delivered messages, not much changes from an end-to-end perspective. But how does SAP Cloud Platform Integration handle the retry if errors occur in SAP Cloud Platform Integration's runtime? Let's try it out. We change the **Address** field in the **Mail** channel connected to the **Receiver** from the **Send** step to an address to some configuration that doesn't work, for example, to `smtp.gmail.com:444`. Save and deploy the integration flow, and run the scenario again.

In the SOAP test client, we get an HTTP response code 202 because the message was successfully delivered to SAP Cloud Platform Integration, but we don't get an email. We've expected this, because of the wrong **Address** configuration in the **Mail** channel. But what happens in SAP Cloud Platform Integration? How is the retry executed, and where do we find the processing details for the message? Let's have a look in the SAP Cloud Platform Integration's monitoring.

Open the monitoring dashboard, and select the **Message Queues** monitor. Select the JMS queue **Inbound_Queue** that is used in the scenario. As shown in [Figure 5.88](#) and [Figure 5.89](#), the message in the JMS queue has the status **Failed**, and the retry count

indicates the number of retries that were already executed. In addition, the time of the next retry is displayed.

The screenshot shows a table titled 'Queues (8)'. The columns are 'Name', 'Entries', and 'Size (in MB)'. The rows list several queues: 'Inbound_Queue' (1 entry, 0.002 MB), 'InbPreProcess_Queue' (0 entries, 0.000 MB), 'OOM_Final' (0 entries, 0.000 MB), 'OOM_Queue' (0 entries, 0.000 MB), 'Queue1' (0 entries, 0.000 MB), and 'Queue2' (0 entries, 0.000 MB). A blue box highlights the header 'Used Capacity: 11.7 / 786432 KB'.

Queues (8)		
Name	Entries	Size (in MB)
Inbound_Queue	1	0.002
InbPreProcess_Queue	0	0.000
OOM_Final	0	0.000
OOM_Queue	0	0.000
Queue1	0	0.000
Queue2	0	0.000

Figure 5.88 Message in Failed Status in Manage Message Queues Monitor A

The screenshot shows a table titled 'Messages (1)'. The columns are 'JMS Message ID', 'Message ID', and 'Status'. The row contains 'ID: 10.120.41.21d635160f517f4960:2', 'AFpbcmuRfdqJa4V9Josz8z1soS2k', and 'Failed'. Below the table, message details are listed: 'Due At: Jan 16, 2018, 16:08:27', 'Created At: Jan 14, 2018, 16:08:27', 'Retain Until: Apr 14, 2018, 17:08:27', 'Retry Count: 2', and 'Next Retry On: Jan 14, 2018, 16:11:47'. Action buttons include 'Filter by Message ID', 'Retry', 'Delete', 'Download', and 'Edit'.

Messages (1)		
JMS Message ID	Message ID	Status
ID: 10.120.41.21d635160f517f4960:2	AFpbcmuRfdqJa4V9Josz8z1soS2k	Failed

Due At: Jan 16, 2018, 16:08:27
Created At: Jan 14, 2018, 16:08:27
Retain Until: Apr 14, 2018, 17:08:27
Retry Count: 2
Next Retry On: Jan 14, 2018, 16:11:47

Figure 5.89 Message in Failed Status in Manage Message Queues Monitor B

Message Processing Details

To get more details for the processing of this message and the error, we need to check the message in the message processing monitor. We can do this either by searching for the message ID shown in the message processing monitor or directly selecting the link for the **Message ID** that opens the **Message Processing** monitor for this specific message (Figure 5.90). There we find the details of the message processing and the

error information. Notice, that the status of the message in message processing monitor is **Retry**, indicating that this isn't a final status. In the **Status Details** for the message, the error details of the last retry are shown.

The screenshot shows the 'Monitor Message Processing' interface. At the top, there are filters for 'Time' (Past Hour), 'Status' (Retry), 'Artifact' (All Integration Flows), and an 'ID' search bar. Below the filters, a timestamp range is displayed: Jan 14, 2018, 15:10:11 - Jan 14, 2018, 16:10:11. The main area shows a list of 'Messages (1)'. The single message listed is 'Book Integration Flow with Splitter', which has a status of 'Retry'. The message details show it was created on Jan 14, 2018, 16:10:04 and has a log level of 'Info'. To the right of the message list, there is a summary for 'Book Integration Flow with Splitter': Last Updated at Jan 14, 2018, 16:10:04, Processing Time: 1 min 36 sec 677 ms. Below the message list, there are tabs for 'Status Details' (which is selected), 'Logs', and 'Artifact Details'. A callout box highlights an error message: 'An error occurred during message processing and a retry has been started automatically.' Under the 'Error Details' section, the error stack trace is shown: com.sun.mail.util.MailConnectException: Couldn't connect to host, port: smtp.gmail.com, 444; timeout 30000;

Figure 5.90 Message Processing Details for a Message in Retry Status

Correct the wrong address configuration in the mail channel, save, and deploy the integration flow. The next retry will consume the message and process it with the new configuration resulting in a successful execution. The emails finally arrive in your email account.

Trigger a Retry in Manage Message Queues Monitor

If you don't want to wait for the time of the next retry, you can initiate the retry in the **Manage Message Queues** monitor by selecting the **Retry** action for this message. Then the next retry is executed immediately, and the message gets processed.

Now you're able to configure reliable messaging scenarios with decoupled inbound processing in SAP Cloud Platform Integration, ensuring that retries of the message are executed in SAP Cloud Platform Integration if there are errors during message processing. With this, you've speeded up the processing time for the sender and ensured reliable messaging by using the retry functionality based on JMS queues offered by SAP Cloud Platform Integration.

5.4.2 Configure Retry for Multiple Receivers

For the scenario configured in the previous section, the retry initiated directly after the inbound processing is sufficient because the message isn't sent to multiple receivers in parallel. But for a parallel multicast scenario, where the message is to be delivered to multiple receivers in parallel, we still have the problem that some receivers get duplicate messages if there is a temporary communication error sending the message to one of the receivers. To address this, we use separate outbound queues for each of the receivers to execute the retry only for the message going to the receiver that is temporarily not available.

To showcase this, we change the scenario slightly by simply sending the same message out to two receivers using the mail adapter. For the ease of testing, we send both to the same email account using different subjects. Follow these steps:

1. Add another **End Message** event to the integration process configuring the main processing.
2. Select the dotted line representing the message flow from the **Send** step to the **Receiver**. The line gets orange with small blue dots at the start and the end of the line. Select the blue dot appearing at the **Send** step, and move it to the new **End** event. Now the existing **Mail** channel is connected from the **End** event.
3. Reconfigure the **Mail** channel going to **Receiver2** to send the message payload instead of the status email: in the **Subject** field, set **Message to Receiver2**, and as **Mail Body**, include the message's payload ([Figure 5.91](#)).



Figure 5.91 Configuring Subject and Mail Body in the Mail Adapter for Receiver2

4. Remove the **Send** step and add a **Parallel Multicast** instead. Draw a second line from the **Parallel Multicast** step to the new **End** event to configure the second multicast branch.

The changed part of the integration flow now looks similar to [Figure 5.92](#).

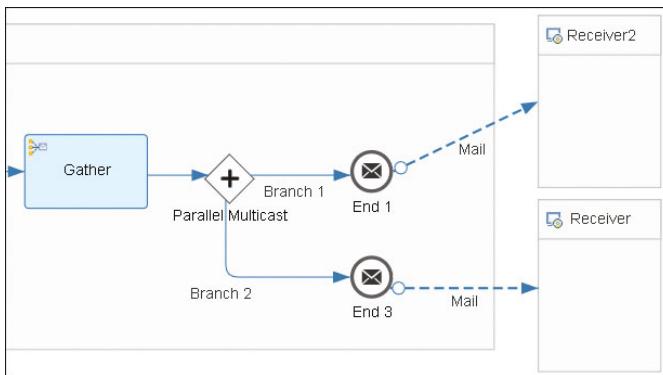


Figure 5.92 Configuration of a Parallel Multicast to Two Receivers

Save the changes, and deploy the integration flow. When triggering the integration flow from your SOAP test client, two emails are sent to your email account—each one representing the message flow in one of the multicast branches and both containing the same message in the email’s body.

To test the retry behavior triggered from the inbound JMS queue for the parallel multicast scenario, we change the **Address** for one of the **Mail** channels to a configuration that doesn’t work, for example, `smtp.gmail.com:444`. Save and deploy the integration flow. Call the integration flow from the SOAP test client. You now get the same email again and again, one for each retry triggered from the inbound JMS queue. Because the second **Mail** receiver can’t be reached, the overall processing status of the message is **Retry**, so the message stays in the JMS queue and is retried according to the **Retry Details** configured in the **JMS** sender adapter.

The behavior to send the message again and again in error situations may cause problems in the receiver system if the receiver can’t handle duplicates. To make sure the receiver doesn’t get the same message multiple times, we’ll now change the scenario in a way that separate outbound queues are used for each of the receivers:

1. Add a new **Integration Process** below the existing integration process. Remember, you find the **Integration Process** in the modeling palette under the **Process** shape.
2. In the new **Integration Process**, add a **Start Message** event and an **End Message** event from the **Events** shape. Connect the **Start** event with the **End** event.
3. Select the dotted line representing the message flow from the **End** event to **Receiver2** in the integration process doing the main processing. The line turns orange

with small blue dots at the start and the end of the line. Select the blue dot appearing at the **End** event, and move it to the **End** event in the new integration process. Now the existing **Mail** channel is connected from the **End** event of the new integration process.

4. Add a **Receiver** from the **Participant**  shape to the model on the right side of the integration process doing the main processing.
5. Connect the **End** event of the integration process doing the main processing to the new **Receiver**. In the adapter selection screen, choose the **JMS** adapter.
6. In the **JMS** channel's **Processing** tab, configure **Outbound_Queue2** as **Queue Name**, and keep the defaults for the other configuration settings ([Figure 5.93](#)).

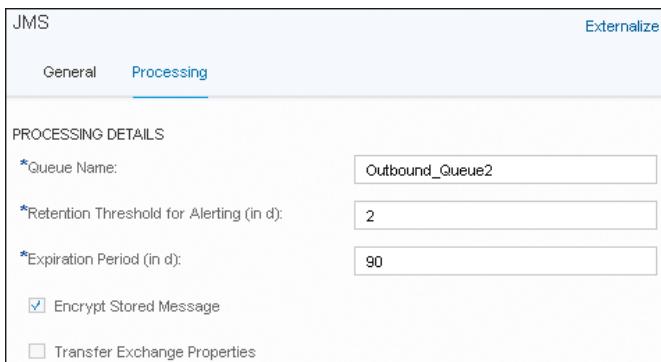


Figure 5.93 Configuration of the JMS Receiver Channel for Outbound Queue

7. Add a **Sender** from the **Participant**  shape to the model on the left side of the new integration process.
8. Connect the **Sender** to the **Start** event of the new integration process. In the adapter selection screen, choose the **JMS** adapter.
9. In the **JMS** channel's **Processing** tab, in the **Queue Name** field, configure the same value as in the **JMS** receiver channel; **Outbound_Queue2**. With the JMS outbound queue configured for the first receiver, the extended parts of the integration flow now look similar to [Figure 5.94](#).
10. Create another **Integration Process** configuring a JMS outbound queue for **Receiver**. Execute the same configuration steps, move the existing **Mail** channel to the new integration process, and use **Outbound_Queue1** as the **Queue Name** in the **JMS** adapter. The configuration of the JMS outbound queues now looks similar to [Figure 5.95](#).

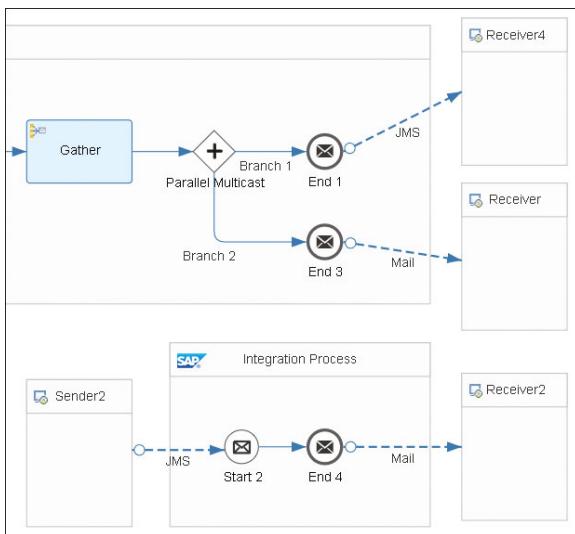


Figure 5.94 Configuration of JMS Outbound Queue for Receiver2

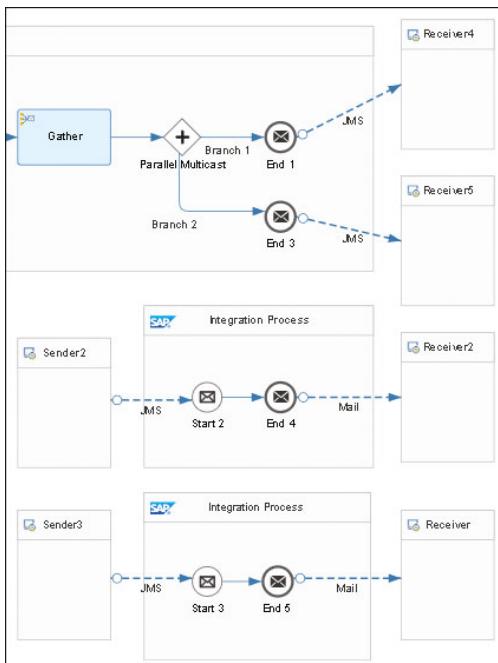


Figure 5.95 Configuration of JMS Outbound Queues for Both Receivers

11. Because we're using two **JMS** receiver channels in the same integration process, we need to configure a JMS transaction handler. The transaction handler makes sure the message is either stored in both JMS queues consistently or in none of them; if there is an error, the whole transaction is rolled back. Select the integration process configuring the parallel multicast, and the properties for the integration process are shown in the **Properties** section. In the **Processing** tab, the configuration options for transaction management are available. In the **Transaction Handling** dropdown, you configure the transaction handler for the integration process ([Figure 5.96](#)) with three available options:
- **Required for JDBC:** Choose this option if Java Database Connectivity (JDBC) transacted resources such as **Data Store** steps are used in the scenario and it's necessary to ensure end-to-end transactional processing in all steps accessing the database.
 - **Required for JMS:** Choose this option if, like in our scenario, JMS transacted resources are used, and end-to-end transactional processing is required. This is the setting we configure for our integration process ([Figure 5.96](#)).
 - **Not Required:** Choose this option for most of the scenarios that don't use transaction resources and therefore don't require a transaction handler.



Figure 5.96 Configuration of Transaction Handling

If a transaction handler, JMS or JDBC, is configured, a **Timeout** needs to be defined. Because transactions are limited resources in the database and in the JMS instance, this setting is required to make sure the transaction is stopped after some time and doesn't run forever. You have to configure a timeout sufficient for your scenario but not too high, keeping the limited transactions in mind. If the available numbers of transactions are reached in runtime, no new messages can be processed.

Configuration of Transaction Handling

In integration scenarios, it's usually required to ensure consistent end-to-end processing and to roll back all actions in the database or in JMS queues consistently. This is ensured by a transaction handler. SAP Cloud Platform Integration offers two transaction handlers: one for JMS transactions and one for JDBC transactions. Distributed transactions between JMS and JDBC aren't supported in SAP Cloud Platform Integration. Recommendations and limitations using transaction handlers, including several sample scenarios, are explained in detail in the "How to Configure Transaction Handling in Integration Flow" blog in the SAP Community (www.sap.com/community.html).

Save the changes made in the integration flow to trigger the check's execution for the integration flow. We notice that a check error is now shown for the main integration process; the error tells us that the JMS transaction handling isn't allowed for parallel multicast (Figure 5.97).

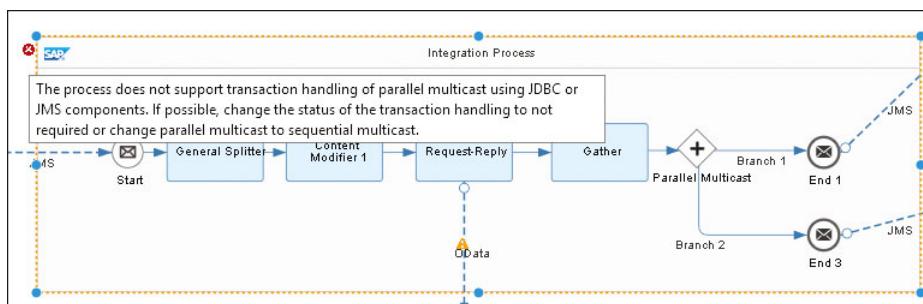


Figure 5.97 Error When Using JMS Transactions with Parallel Multicast

What is the background of this error, and what do we need to do? We already learned that the message is copied in the SAP Cloud Platform Integration's runtime in the parallel multicast pattern, and the different messages are processed independently in all branches leaving the multicast. The problem is that such independent requests can't be rolled back consistently. For our scenario, we need to use sequential multicast and store the messages in the different JMS outbound queues one after the other. If an error happens, the whole process is rolled back, no messages are stored in the JMS outbound queues, and the processing starts again from the JMS inbound queue or the sender system if no JMS inbound queues are used. After successfully

storing the messages to all the JMS outbound queues, the consumption of the messages from the JMS outbound queues is executed in parallel for each of the receivers.

Make the necessary changes by following these steps:

1. Remove the **Parallel Multicast** step, and add a **Sequential Multicast** step instead.
2. Reconnect the **Gather** to the **Parallel Multicast** and the **Parallel Multicast** to the two **End** events.

The final integration flow now looks similar to [Figure 5.98](#).

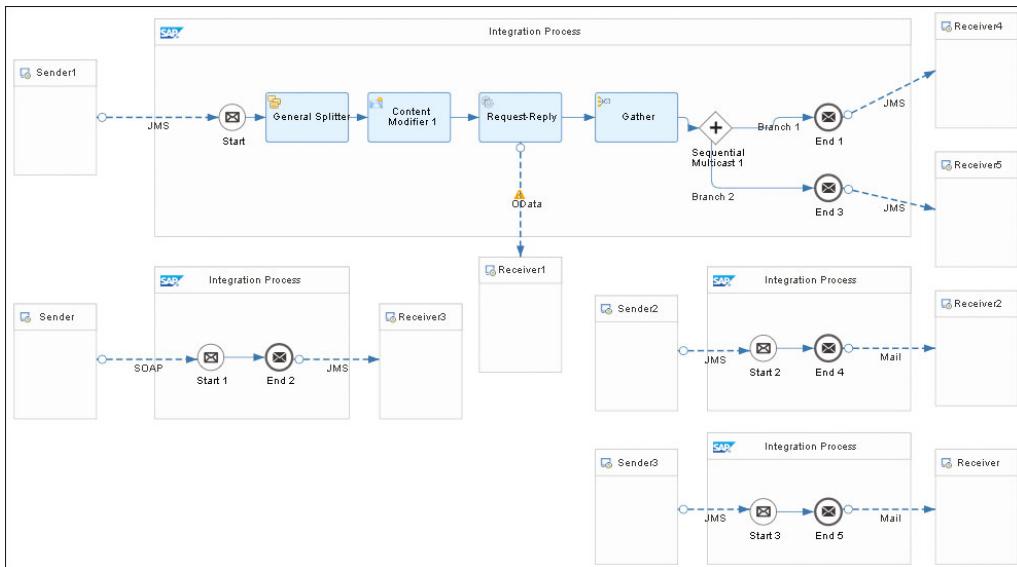


Figure 5.98 Integration Flow with Sequential Multicast and JMS Outbound Queues

Save and deploy the integration flow, and trigger the integration process from your SOAP test client. Now you get only one email, the one processed by the mail adapter that is configured correctly. The message for the second receiver, the one with the wrong configuration, is now available in the JMS outbound queue for this receiver ([Figure 5.99](#) and [Figure 5.100](#)) and is retried from this queue. If the message in **Failed** status from the last scenario execution is still available in the JMS inbound queue, it will be retried at the next retry time, or you can trigger an immediate retry in the **Manage Message Queues** monitor.

Overview / Manage Message Queues		
■ Used Capacity: 12.3 / 786432 KB		
Queues (10)		Filter by Name <input type="text"/> Actions ▼
Name	Entries	Size (in MB)
Inbound_Queue	0	0.000
InbPreProcess_Queue	0	0.000
OOM_Final	0	0.000
OOM_Queue	0	0.000
Outbound_Queue1	1	0.002
Outbound_Queue2	0	0.000

Figure 5.99 Message in JMS Outbound Queue in Failed Status in the Manage Message Queues Monitor A

Messages (1) Filter by Message ID <input type="text"/> Retry Delete Download [C] [D] [E]		
JMS Message ID	Message ID	Status
ID:10.120.41.21d635160f517f4960:19	AFpc6v45dWnDfrsYB40ugN68IC8A	Failed
Due At: Jan 17, 2018, 18:55:10		
Created At: Jan 15, 2018, 18:55:10		
Retain Until: Apr 15, 2018, 19:55:10		
Retry Count: 1		
Next Retry On: Jan 15, 2018, 18:56:00		

Figure 5.100 Message in JMS Outbound Queue in Failed Status in the Manage Message Queues Monitor B

Configuration Option: Use Separate Integration Flows

We configured the different integration processes for storing the messages in a JMS queue and consuming from the JMS queue in one single integration flow, but they can also be configured in different integration flows instead. When separate integration flows are used, you're more flexible when changes and redeployments of the integration flow are necessary.

You've now successfully set up a reliable messaging scenario using JMS inbound and JMS outbound queues, the inbound processing of your scenario is decoupled from

further processing of the message in SAP Cloud Platform Integration, and the sending to multiple receivers is executed asynchronously using a separate JMS outbound queue for each receiver. You've significantly shortened the processing time for the sender and ensured reliable messaging to multiple receivers using the retry capabilities based on JMS queues offered by SAP Cloud Platform Integration.

5.4.3 Configure Explicit Retry with Alternative Processing

Depending on the scenario or on the receiver system, you may want to retry the message only a couple of times and then do an alternative processing of the message. This isn't possible with the default retry configuration offered in the JMS sender channel: the JMS sender adapter retries the message forever. But SAP Cloud Platform Integration offers the option to configure explicit retry handling by using the exception subprocess. In this section, we'll extend the scenario and configure explicit retry handling in the process, sending the message to one of the receivers.

To extend the integration flow to support explicit retry configuration, we need to create a local integration process that contains the configuration for the retry. More details about using local integration processes will be shared in [Chapter 6](#). Execute the following steps to extend the integration flow:

1. Add a **Local Integration Process** below the existing integration processes by choosing it in the modeling palette under the **Process**  shape. In the **General** tab, enter the name “Retry Configuration” to give the local process a meaningful name.
2. In the new **Local Integration Process**, add a **Router** from the **Message Routing**  shape between the **Start** and **End** events. Add an **Error End** event from the **Events**  shape below the newly added **Router** step. Connect the **Router** step to the **Error End** event, as depicted in [Figure 5.101](#).

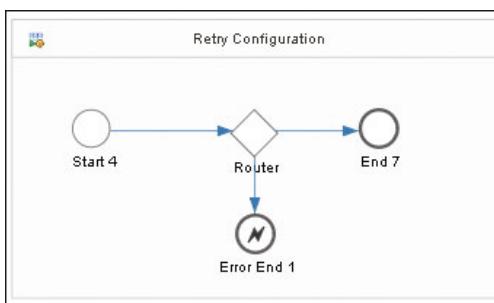


Figure 5.101 Configuration of a Local Integration Process

3. The JMS sender adapter sets the header SAPJMSRetries at runtime. This header allows you to configure a specific retry behavior: it contains the number of retries that already are executed. Based on the value of this header, we can configure whether the message processing continues or ends. If we configure it to end, we can also configure that the message is processed in an alternative way. This is done in the **Router** step. As a simple example, we want to configure that after five retries, the message is sent via email to an administrator, and no further retries are executed from the JMS queue. We configure the two routes leaving the **Router** step per [Figure 5.102](#):

- In the route going to the **End** event, we define a **Non-XML** condition based on the SAPJMSRetries header: \${header.SAPJMSRetries} > '5'. With this configuration, the route to the **End** event is executed as soon as the value of the SAPJMSRetries header is larger than 5. This means that with the sixth retry (SAPJMSRetries = 6) the message is routed to the **End** event, which ends the processing. The message goes to the status **Completed** and is removed from the JMS queue.
- The route to the **Error End** event we define as the default route so that during runtime, the message is routed to the **Error End** event as long as the value of SAPJMSRetries is smaller than 6. The **Error End** event raises an error and makes sure the message stays in the JMS queue, and the message gets the status **Retry**.

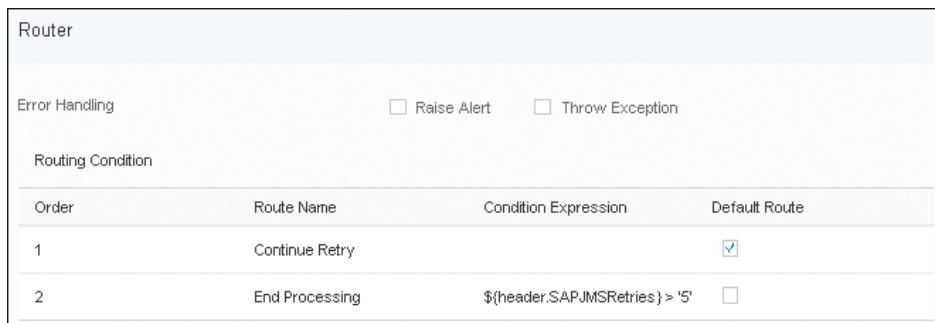


Figure 5.102 Configuration of Router Step in Local Integration Subprocess

4. Add an **Exception Subprocess** from the **Process** shape to the outbound integration process connected to the **Receiver** for which we want to configure the alternative processing. This **Exception Subprocess** is supposed to catch the error in the outbound processing.

5. Add a **Process Call** (from the **Call ↗** shape, submenu **Local Call**), and place it between the **Error Start** event and the **End** event of the newly added **Exception Subprocess**. In the properties area for the **Process Call**, select the local integration process **Retry Configuration** created before (Figure 5.103).

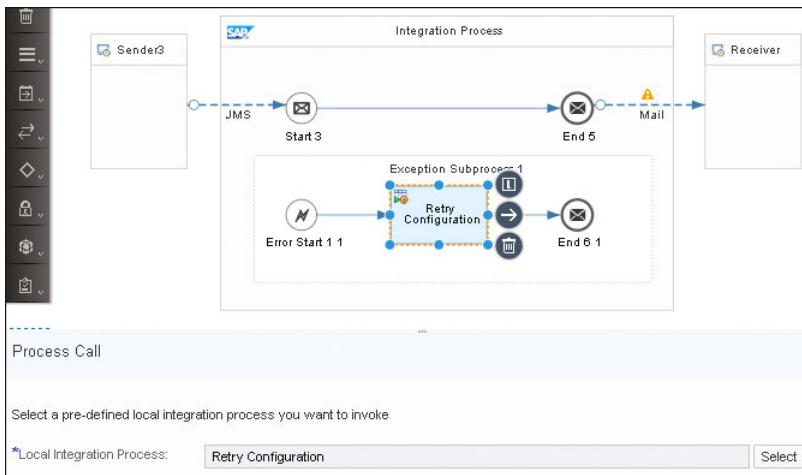


Figure 5.103 Configuration of the Process Call in the Exception Subprocess

6. Add a **Receiver** from the **Participant** shape, and place it on the right side of the integration process that contains the **Exception Subprocess**. In the properties for the **Receiver**, define **Administrator** as the **Name**. Draw a line from the **End** event of the **Exception Subprocess** to the newly added **Receiver**, and select the **Mail** adapter in the adapter selection screen.
7. In the **Connection** tab of the **Mail** adapter, configure the **Connection Details** according to your email account and the **Mail Attributes**, as shown in Figure 5.104. For the ease of testing, you can use the same email account to send the email created for the administrator too, but use a different **Subject** to identify that this email is meant for the administrator.

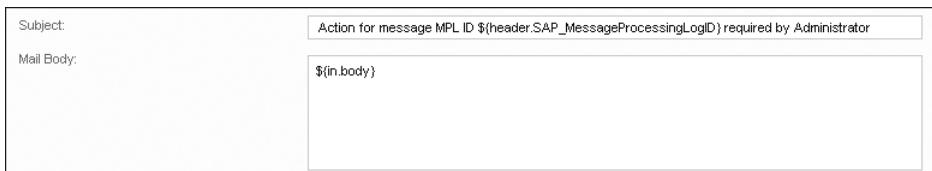


Figure 5.104 Configuring the Mail Adapter for the Administrator

The integration flow containing all of its integration processes, the local process, and the exception process now looks similar to [Figure 5.105](#).

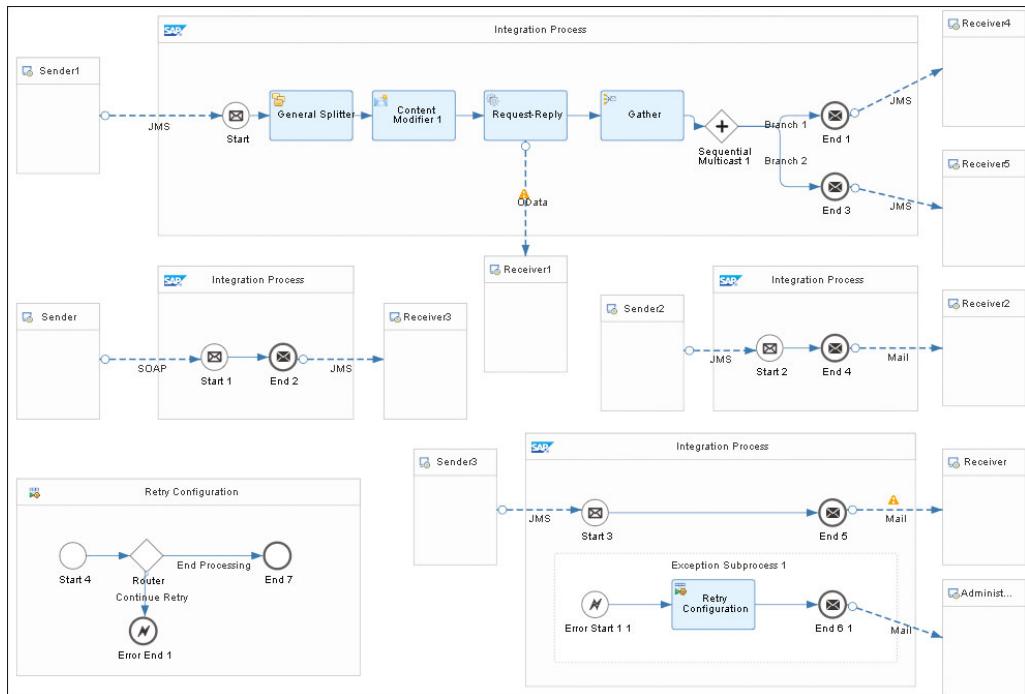


Figure 5.105 Integration Flow with Explicit Retry Configuration in Exception Subprocess

Save and deploy the integration flow. Now we use the integration flow to test the runtime and the retry behavior by triggering the processing from the SOAP test client. We adjust the **Address** in the different **Mail** adapter channels to raise an error during runtime to trigger the retries configured.

Successful End-to-End Processing

First, we need to test the successful scenario execution. Make sure the **Address** field is correct in both **Mail** adapter channels, save and deploy the integration flow, and trigger the integration flow from your SOAP test client.

The messages are sent successfully to both receivers, and you receive two emails after a short time, both containing the same message in the email's body. However, they have different subjects: **Book Demo Msg** (for **Receiver**) and **Message to Receiver2** (for

Receiver2) as configured in the **Mail** adapter channels for the two receivers. This indicates that the end-to-end processing of the message was executed without errors, and the messages were sent asynchronously to both receivers.

Let's have a look at the SAP Cloud Platform Integration's monitoring dashboard to understand what happens during runtime. Open the **Message Processing** monitor. As shown in [Figure 5.106](#), four entries appear for this single scenario execution. This may be surprising, but it can be understood if you consider the end-to-end scenario configuration and the fact that SAP Cloud Platform Integration writes a separate message processing log for each integration process. The scenario consists of four integration processes: one integration process for inbound processing and storing the message in the JMS inbound queue, one integration process for the main processing and storing the message in the two JMS outbound queues for the two receivers, and one integration process for each of the two receivers. For each of these processes, a separate log entry is created in message monitoring.

To make the end-to-end monitoring easier the four MPLs are correlated by a **Correlation ID**. The **Correlation ID** is shown below the **Message ID** in the **Properties** of each MPL. As depicted in [Figure 5.106](#), it's possible to search for all MPLs having the same correlation ID using the **ID** field in the search criteria.

The screenshot shows the SAP Cloud Platform Integration Message Processing monitor interface. At the top, there are filters for Time (Past 24 Hours), Status (All), Artifact (All Integration Flows), and an ID search field containing 'AFsWyDE9YBYbLI9A3v8o6ileuQhc'. Below this, a table lists 'Messages (4)' with columns for Artifact Name and Status. All four entries are 'Completed'. The table data is as follows:

Artifact Name	Status
Book Integration Flow with Splitter	Completed
Jun 05, 2018, 20:31:27	1 h 3 min
Book Integration Flow with Splitter	Completed
Jun 05, 2018, 19:28:29	4 sec 560 ms
Book Integration Flow with Splitter	Completed
Jun 05, 2018, 19:28:24	2 sec 667 ms
Book Integration Flow with Splitter	Completed
Jun 05, 2018, 19:28:21	4 sec 144 ms

On the right, a detailed view for the first completed flow is shown. It includes a summary message: 'Book Integration Flow with Splitter' last updated at 'Jun 05, 2018, 20:31:27'. Below this, tabs for Status, Properties, and Logs are present. The Status tab shows a green box stating 'Message processing completed successfully.' The Properties tab lists the Message ID as 'AFsWyDhYCBVt1h7t4Ab5zqXuZ391' and the Correlation ID as 'AFsWyDE9YBYbLI9A3v8o6ileuQhc'.

Figure 5.106 Message Processing Monitor for Successful Execution

All four MPLs have the status **Completed**, indicating that the message was executed successfully in all four integration processes.

In the **Manage Message Queues** monitor, there will be no message in any of the involved JMS queues because the messages were consumed and successfully processed.

Error Sending Message to Receiver2

We now change the **Address** field in the **Mail** adapter channel connected to the **Receiver** with name **Receiver2** to simulate that the processing to **Receiver2** is broken. Save and deploy the integration flow, and trigger the integration flow from your SOAP test client.

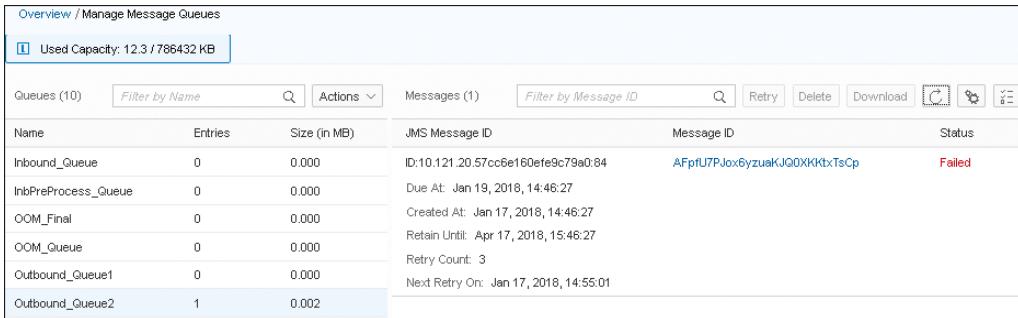
Because the message is successfully sent to **Receiver**, you get one email with the subject **Book Demo Msg**. The other message which is expected to be sent to **Receiver2** remains in the JMS outbound queue **Outbound_Queue2** configured for **Receiver2** and is retried according to the retry configuration in the JMS sender channel.

In the SAP Cloud Platform Integration's monitoring for this execution, we again find four entries in the **Message Processing** monitor, one for each process execution. The first three MPLs have the status **Completed**, but the most recent entry has the status **Retry** (Figure 5.107). This is what we expect because the message to **Receiver2** is being retried from the JMS queue.

Messages (4)	
Artifact Name	Status
Book Integration Flow with Splitter	Retry
Jan 17, 2018, 14:46:57	Log Level: Debug
Book Integration Flow with Splitter	Completed
Jan 17, 2018, 14:46:27	Log Level: Debug
Book Integration Flow with Splitter	Completed
Jan 17, 2018, 14:46:27	Log Level: Debug
Book Integration Flow with Splitter	Completed
Jan 17, 2018, 14:46:25	Log Level: Debug

Figure 5.107 Message Processing Monitor for Erroneous Execution

As depicted in Figure 5.108, the message can be found in the **Manage Message Queues** monitor in the queue **Outbound_Queue2** with the status **Failed**, providing information about the retries already executed and the time of the next retry.



The screenshot shows a table with two sections: 'Queues (10)' and 'Messages (1)'. The 'Queues' section has columns for Name, Entries, and Size (in MB). The 'Messages' section has columns for JMS Message ID, Message ID, and Status. A single message is listed with its details.

Name	Entries	Size (in MB)	JMS Message ID	Message ID	Status
Inbound_Queue	0	0.000	ID:10.121.20.57cc6e160ef9c9a0:84	AFprU7Pjox6yzuaKJQ0XKtxTsCp	Failed
InbPreProcess__Queue	0	0.000		Due At: Jan 19, 2018, 14:46:27	
OOM_Final	0	0.000		Created At: Jan 17, 2018, 14:46:27	
OOM_Queue	0	0.000		Retain Until: Apr 17, 2018, 15:46:27	
Outbound_Queue1	0	0.000		Retry Count: 3	
Outbound_Queue2	1	0.002		Next Retry On: Jan 17, 2018, 14:55:01	

Figure 5.108 Message in Error in Outbound_Queue2 in Manage Message Queues Monitor

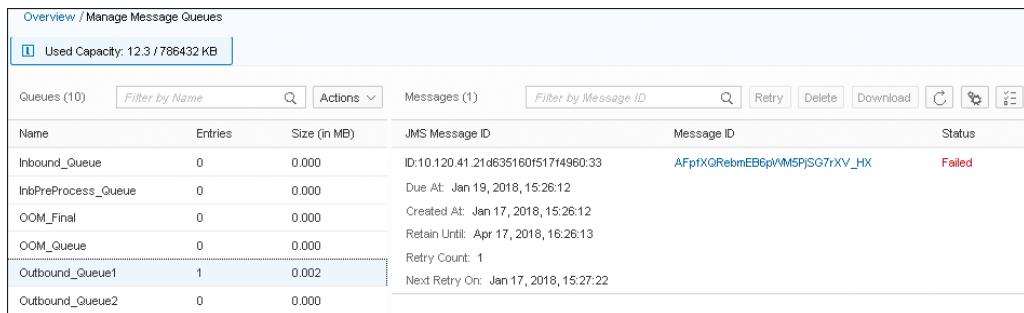
To clean up the **Manage Message Queues** monitor in preparation for the next test, you can either correct the **Mail** adapter's **Address** configuration, redeploy the integration flow, and retry the message by using the **Retry** action in the **Manage Message Queues** monitor, or you can delete the message from the JMS queue using the **Delete** action in the **Manage Message Queues** monitor. Otherwise, the message is retried again and again until it's finally deleted after it exceeds the **Retention Threshold** defined in the JMS receiver channel.

Error Sending Message to Receiver

In the next test, we change the **Address** field in the **Mail** adapter channel connected to the **Receiver** with name **Receiver** to a configuration that doesn't work. With this configuration, we simulate that the processing to **Receiver** is broken. If not already done, correct the address in the **Mail** adapter channel for **Receiver2** to make sure the message to this receiver can be sent successfully. Save and deploy the integration flow, and trigger the integration flow from your SOAP test client.

Because the message is successfully sent to **Receiver2**, you get one email with the subject **Message to Receiver2**. About one hour later (because of the configured six retries with **Exponential Backoff**), you get the email with the subject indicating that it's to be processed by the administrator.

If we check the **Message Processing** monitor directly after scenario execution, we see exactly the same message processing status as in the last test: three entries with the status **Completed** and the most recent entry with the status **Retry** (Figure 5.107). This time, the message that is being retried is the message to **Receiver**. In the **Manage Message Queues** monitor, you find the message in the **Outbound_Queue1** with the status **Failed** (Figure 5.109).



The screenshot shows the 'Manage Message Queues' monitor interface. At the top, it displays 'Used Capacity: 12.3 / 786432 KB'. Below this, there are two tabs: 'Queues (10)' and 'Messages (1)'. The 'Queues' tab is selected, showing a table with columns: Name, Entries, Size (in MB), JMS Message ID, Message ID, and Status. The table contains the following data:

Name	Entries	Size (in MB)	JMS Message ID	Message ID	Status
Inbound_Queue	0	0.000	ID:10.120.41.21:ds35160!51714960:33	AFptXQRbmEB6pWMS5PSG7rXV_HX	Failed
InbPreProcess_Queue	0	0.000		Due At: Jan 19, 2018, 15:26:12	
OOM_Final	0	0.000		Created At: Jan 17, 2018, 15:26:12	
OOM_Queue	0	0.000		Retain Until: Apr 17, 2018, 16:26:13	
Outbound_Queue1	1	0.002		Retry Count: 1	
Outbound_Queue2	0	0.000		Next Retry On: Jan 17, 2018, 15:27:22	

The 'Messages' tab is also visible, showing one message entry with a status of 'Failed'.

Figure 5.109 Message in Error in Outbound_Queue1 in Manage Message Queues Monitor

If you check the **Message Processing** monitor again after you received the second email that is targeted for the administrator, you notice that all four entries in the monitor have the status **Completed** now. This is because all retries are executed in the same MPL entry; no new log is created to make the monitoring easier. The sixth retry, still running in an error, triggered the alternative route and completed the message.

Error Sending to the OData Receiver

In a scenario, not only can the final connection to the receivers be broken, but errors can happen during the other message processing steps as well. In our scenario, for example, the request-reply call to the OData receiver could fail. To see how the integration flow copes with this, we change the **Address** field in the **OData** adapter channel to a configuration that doesn't work. Save and deploy the integration flow and trigger the integration flow from your SOAP test client.

For this execution, you don't get any email because the message processing in SAP Cloud Platform Integration didn't even get to the outbound processing. This time, the main processing and the enrichment of the message with the additional data from the OData service could not be executed.

In the **Message Processing** monitor, we find only two entries this time (Figure 5.110), one with status **Completed** for the integration process storing the message in the JMS inbound queue, and one with status **Retry** for the integration process doing the main processing. So, the message in the **Manage Message Queues** monitor in the **Inbound_Queue** has the status **Failed** (Figure 5.111), and the retries are triggered from this JMS queue.

Messages (2)	
Artifact Name	Status
Book Integration Flow with Splitter	Retry
Jan 17, 2018, 18:33:30	Log Level: Info
Book Integration Flow with Splitter	Completed
Jan 17, 2018, 18:33:30	Log Level: Info

Figure 5.110 Message Processing Monitor for Errors in OData Connection

Overview / Manage Message Queues					
Queues (10)			Messages (1)		
Name	Entries	Size (in MB)	JMS Message ID	Message ID	Status
Inbound_Queue	1	0.002	ID:10.121.20.57cc6e160efe9c79a:95	AFppfiOrtD8wqLe33ANVE2vN9ykCR	Failed
InbPreProcess_Queue	0	0.000	Due At: Jan 19, 2018, 18:33:30		
OOM_Final	0	0.000	Created At: Jan 17, 2018, 18:33:30		
OOM_Queue	0	0.000	Retain Until: Apr 17, 2018, 19:33:29		
Outbound_Queue1	0	0.000	Retry Count: 3		
Outbound_Queue2	0	0.000	Next Retry On: Jan 17, 2018, 18:39:54		

Figure 5.111 Message in Error in Inbound_Queue in Manage Message Queues Monitor

Delete the message from the JMS queue to stop the retries; otherwise, the message is retried again and again until it's finally deleted after exceeding the **Retention Threshold** defined in the JMS receiver channel.

Now you've successfully set up a reliable messaging scenario using JMS inbound and JMS outbound queues. You've also understood the different options for configuring the retries from JMS queues: either define the retry configuration in the JMS sender channel or model explicit retry configuration in an exception subprocess.

5.5 Summary

Congratulations! You've mastered another important chapter on your journey through the world of integration using SAP Cloud Platform Integration. We implemented more sophisticated integration scenarios, comprising steps for content-based message routing and managing messages containing lists of entries. You applied the splitter pattern to create individual messages out of the list and used the gather pattern to aggregate the individual messages back into one reply message.

You also learned how to influence synchronous and asynchronous message handling for the SOAP adapter and how to add asynchronous receivers to your integration flow. You used the multicast pattern to distribute the message to multiple receivers. At the end of the chapter, you used the JMS adapter to decouple inbound and outbound processing and to ensure retries of the message processing in SAP Cloud Platform Integration. This knowledge allows you to play around with SAP Cloud Platform Integration's more advanced features. However, our journey continues: the next chapter will reveal even more secrets about the timer-based start of integration flows, the structuring of large flows using modularization, and, finally, developing new adapters for SAP Cloud Platform Integration.

Chapter 6

Special Topics in Integration Development

In the previous chapters, you learned the basic concepts of integration development with SAP Cloud Platform Integration, but our discussion wouldn't be complete without covering a few special topics. This chapter discusses timer-based message handling processes, structuring large integration scenarios, using dynamic parameters, integration flow component versioning, transporting integration packages, and developing custom adapters, which may be required to connect to some external systems.

This chapter addresses typical integration scenarios that will require special attention from you. This chapter will provide answers if you have questions about the time-based triggering of integration flows, including the invocation of a Simple Object Access Protocol (SOAP) data source, the dynamic configuration of integration flows; the slicing and dicing of complex integration logic; the intrinsic versioning of integration flows and how to handle it. Furthermore, you'll learn how to transport integration packages from one tenant to another and how to develop your own adapters that can be integrated into the SAP Cloud Platform infrastructure. After you finish this chapter, you'll be well equipped to handle a variety of integration challenges.

6.1 Timer-Based Message Transfer

In previous chapters, we used SAP Cloud Platform Integration as the cloud-based solution for reliable message transfer between on-premise and on-demand enterprise applications. So far, the integration logic executed on SAP Cloud Platform Integration was mainly triggered by incoming messages, such as an order request originating from a sender customer relationship management (CRM) system, which

needed to be routed to various backend enterprise resource planning (ERP) systems, depending on the message's content. However, not all integration scenarios require an incoming message to trigger their execution. Sometimes, you want to check for existing data on a regular basis, for example, retrieving the status of a machine or related machine data in the Internet of Things and forwarding the information to a business intelligence system for further analysis. Scenarios like this require a timely initiation of a message transfer. This section of the book will dive into the details on how this can be achieved using SAP Cloud Platform Integration.

6.1.1 The Scenario

To show how to design a timer-base integration flow, we'll modify the scenario from [Chapter 4, Section 4.3](#) (Figure 4.44). You remember that in this scenario, we invoked an external OData service by sending a SOAP message with a dedicated order number. The SOAP service retrieved details for that specific order.

Let's assume now that you're interested in getting regular updates about all orders placed for a specific shipping country, for example, France. We can design a timer-based integration flow in such a way that we get a list of all such orders into your email account in regular intervals. With just a few steps, we can now modify the scenario without introducing any new steps except for the **Start Timer** event. [Figure 6.1](#) shows the target scenario.

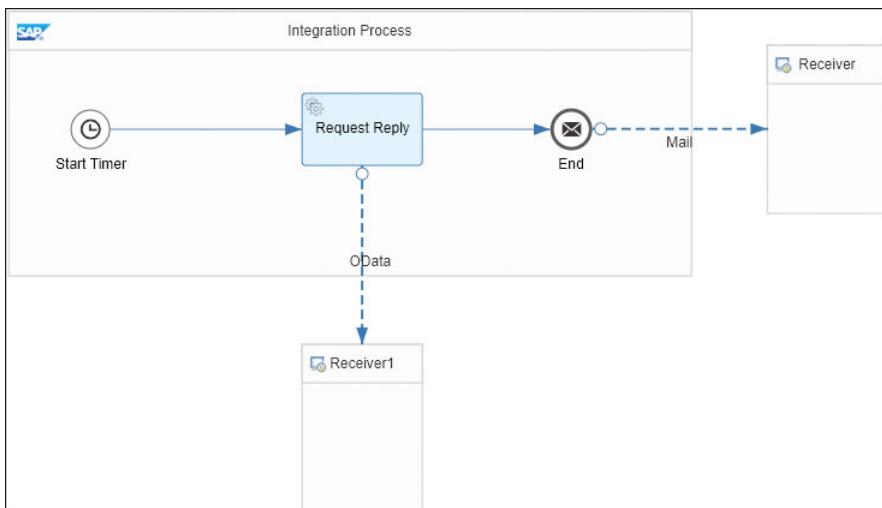


Figure 6.1 Timer-Based Invocation of an OData Service

At the start of the flow on the left side of the figure, you'll see the first difference compared to the other integration flows we've discussed so far. This integration flow begins with a new event called **Start Timer**, indicated by the clock . We'll take a closer look at its configuration in a moment. The **Start Timer** event is followed by a **Request-Reply** step and connected via OData to the external service (the same service we've used in [Chapter 4, Section 4.3](#)). After we receive the result from the service, it's sent to the receiver via a **Mail** adapter. It shouldn't be a problem for you to model the scenario based on the knowledge you've acquired so far. The only new shape in this integration flow is the **Start Timer** event. You'll find it in the palette beneath the event's main entry  (see [Figure 6.2](#)).

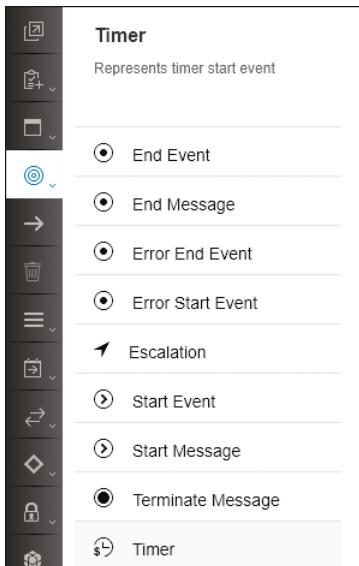


Figure 6.2 The Timer Start Event in the Modeling Palette

6.1.2 Configuring a Timer-Based Integration Flow

We'll begin by configuring the most important shape of the flow: the **Start Timer** event. This event is responsible for initiating the flow's execution without requiring a dedicated start message. The configuration options are depicted in [Figure 6.3](#) and [Figure 6.4](#).

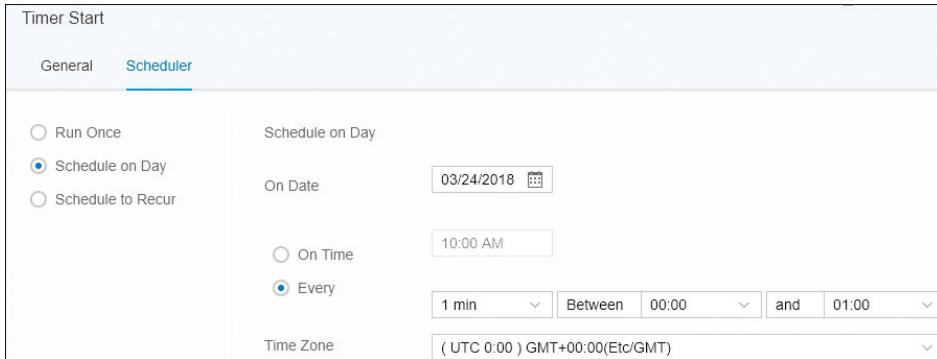


Figure 6.3 Configuration of the Start Timer Event for a Scheduled Start on a Certain Day

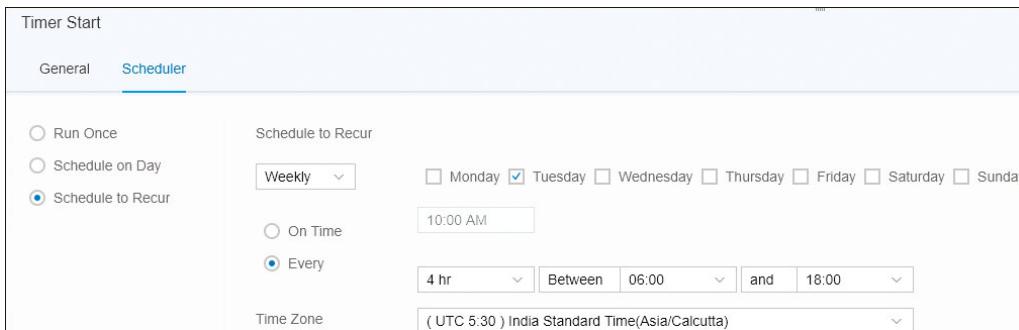


Figure 6.4 Configuration of the Start Timer Event for a Weekly Recurring Schedule

You have three options at your disposal for starting the integration flow:

- **Run Once**

Run only once immediately after deployment of the flow.

- **Schedule on Day**

Scheduled execution on a concrete day at a well-defined time (or well-defined times).

- **Schedule to Recur**

Recurring schedule on a daily, weekly, or monthly basis at a well-defined time (or well-defined times).

The **Run Once** option allows you to trigger the flow's execution immediately after a successful deployment. It can also be used for testing purposes, if you want to try out certain integration functionalities without sending incoming messages all the time,

just to get the flow started. Obviously, you don't have to configure additional settings for this option, as the exact point of time is given by the finalization of the flow's deployment.

The **Schedule on Day** option (shown in [Figure 6.3](#)) defines a concrete day on which the integration flow should run. In the **On Date** field, you enter the respective execution date. However, you can also specify either the one-time execution or a recurring execution on that particular date. If you want to run the flow only once, you have to set the **On Time** checkbox, accompanied by a concrete time. For a recurring invocation of the flow's logic, select the **Every** radio button. In addition, you have to define two intervals:

- **The interval during which the flow should be activated**

Every minute? Every hour? There are even other intervals possible: every second minute, every four hours, and so on. You select the respective interval from the dropdown list to the right of the **Every** radio button.

- **Between which times of the day the flow should run**

For this, you can define the start and end time. The respective dropdown lists are positioned to the right of the **Between** label in [Figure 6.3](#), the first representing the start time, and the second representing the end time.

The settings in [Figure 6.3](#) represent a recurring invocation of the integration flow on March 24, 2018. The flow will be called every minute in the time between midnight and 1:00 a.m. (Greenwich Mean Time zone).

With these options, you can quite flexibly schedule the flow's execution for a certain day. And even for that particular day, you can define a recurring execution. However, the repetition of the invocation is limited to one day. To overcome this limitation, you want to make use of the third option, which allows you to define a more flexible period for the flow's repetition. For this purpose, select the **Schedule to Recur** radio button ([Figure 6.4](#)). The dropdown list beneath the **Schedule to Recur** label allows you to select the period: currently **Daily**, **Weekly**, and **Monthly** repetitions are supported. Depending on your choice, you can define additional attributes:

- **Daily**

Define whether the flow should run only once on that day or on a recurring basis. You have exactly the same options described for the **Schedule on Day** selection.

- **Weekly**

Define the days per week on which the flow should be invoked. The chosen days

will activate the flow every week. You can even pick several days, such as Monday, Wednesday, and Friday.

■ Monthly

Specify the date on which the integration flow should run every month. If a chosen date isn't applicable for a certain month (e.g., 31 isn't a valid date for February, April, June, September, and November), the flow isn't executed. Only one day can be selected.

With the configuration shown in [Figure 6.4](#), the integration flow will be activated every week on Tuesday. The flow runs on this day every four hours, between 6:00 a.m. and 6:00 p.m. The time zone has been set to India Standard Time.

Of course, many configuration options are available to schedule the execution of your integration flows. The **Start Timer** event is the element of choice for start, end, and recurring times, periods, and time spans. Returning to our initial goal of receiving order information in regular intervals, we assume that we want to get this information into our inbox every morning at 7:00 a.m.

Therefore, a reasonable configuration is the selection of the **Schedule to Recur** radio button with the identically named dropdown list set to **Daily**. In addition, select the **On Time** radio button, and set the according time to **7:00 AM** ([Figure 6.5](#)).

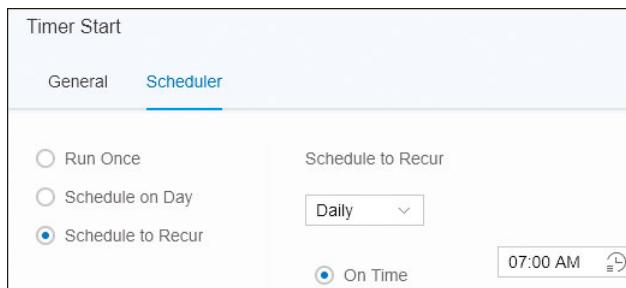


Figure 6.5 Setting the Recurrence to Every Day at 7:00 a.m.

However, there is an alternative approach possible, especially if you want to retrieve the information only for workdays. In this case, choose the **Weekly** entry from the **Schedule to Recur** dropdown list, and set the checkboxes for the required workdays (Monday to Friday in our example; see [Figure 6.6](#)).

It's quite impressive what can be configured for the **Start Timer** event. Use the many configuration options available to define your own schedule.

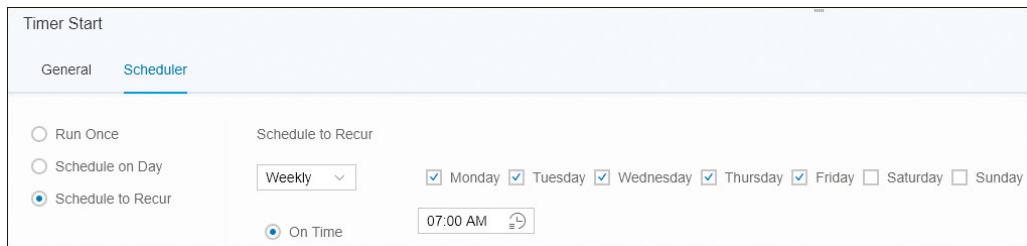


Figure 6.6 Setting the Recurrence to Workdays Only

To continue, we'll now configure the OData connection. To find out how to configure an OData connection in general, refer to [Chapter 4, Section 4.3](#). We just modify the OData adapter by configuring the following settings (Figure 6.7).

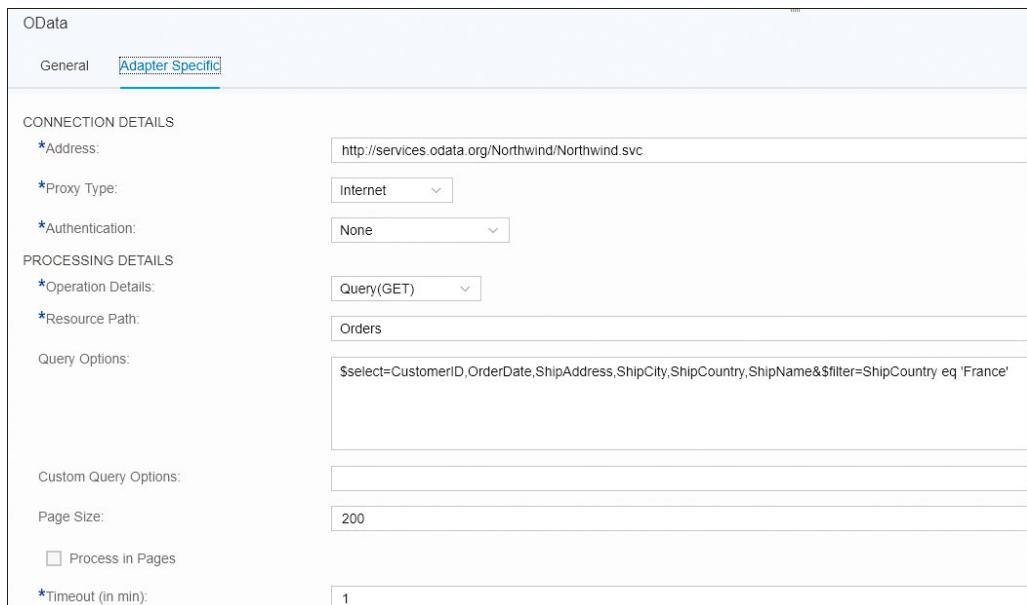


Figure 6.7 OData Adapter Settings to Retrieve All French Orders

Note that in the **Query Options** field, we've entered the expression `$filter=ShipCountry eq 'France'` at the end of the string, which makes sure that we retrieve French orders as expected.

Before we can test the integration flow, we must define the connection to the email server. This has already been explained in the previous chapter about asynchronous

message handling. Refer to [Chapter 5](#), [Section 5.3.2](#), and [Figure 6.1](#) in this chapter to configure the connection to the email server correctly.

6.1.3 Running the Integration Flow

Now that our integration flow has been configured, it's time to try it out. It's recommended to set the **Start Timer** event to **Run Once** for the very first time, just to avoid waiting for the timer to expire on the next day at 7:00 a.m. With the timer set to **Run Once**, the integration flow will begin automatically after deployment, which is exactly what we need for test purposes. Once deployed, connect to your email provider to check the email.

When the integration flow executes successfully, you should find an email with a list of orders (and details as configured in the OData adapter), as shown in [Figure 6.8](#).

```
<Orders>
  <Order>
    <OrderDate>1996-07-04T00:00:00.000</OrderDate>
    <ShipCity>Reims</ShipCity>
    <CustomerID>VINET</CustomerID>
    <ShipAddress>59 rue de l'Abbaye</ShipAddress>
    <ShipCountry>France</ShipCountry>
    <ShipName>Vins et alcools Chevalier</ShipName>
  </Order>
  <Order>
    <OrderDate>1996-07-08T00:00:00.000</OrderDate>
    <ShipCity>Lyon</ShipCity>
    <CustomerID>VICTE</CustomerID>
    <ShipAddress>2, rue du Commerce</ShipAddress>
    <ShipCountry>France</ShipCountry>
    <ShipName>Virtuailles en stock</ShipName>
  </Order>
  <Order>
    <OrderDate>1996-07-25T00:00:00.000</OrderDate>
    <ShipCity>Strasbourg</ShipCity>
    <CustomerID>BLONP</CustomerID>
    <ShipAddress>24, place Kléber</ShipAddress>
    <ShipCountry>France</ShipCountry>
    <ShipName>Blondel père et fils</ShipName>
  </Order>
  <Order>
    <OrderDate>1996-08-06T00:00:00.000</OrderDate>
    <ShipCity>Reims</ShipCity>
    <CustomerID>VINET</CustomerID>
    <ShipAddress>59 rue de l'Abbaye</ShipAddress>
```

Figure 6.8 Excerpt from the Received Email

To summarize, regularly fetching data and forwarding messages to systems are typical requirements for integration solutions such as SAP Cloud Platform Integration. In this part of the book, we've shown you how to prepare integration flows for this purpose. You've seen how to apply the **Start Timer** event for triggering the integration

flow's execution on a regular basis. It doesn't matter whether you want to run the flow only once or in a recurring fashion: the **Start Timer** event is prepared for all kinds of invocation scenarios.

6.2 Using Dynamic Configuration via Headers or Properties

All integration flows you've learned about so far in this book have been designed under the tacit assumption that when modeling the integration flow (during design time), you already know the values of all adapter and integration flow step attributes. However, you can easily think of scenarios where this assumption doesn't apply and where certain attribute values are only known during runtime—when the integration flow is being executed.

You might, for example, like to configure an integration flow with a mail receiver adapter, and the **Subject** field of the outbound email isn't known yet during design time. It should, instead of this, depend on the content of the inbound message (to anticipate an example, we'll show you how to configure this in a minute).

SAP Cloud Platform Integration also has a solution for use cases like this: various integration flow step and adapter attributes that can be configured *dynamically*. To understand how this works, think again about Camel's data model as introduced in [Chapter 4](#). There, you learned that SAP Cloud Platform Integration provides the option to pass along certain information during the processing of a message, namely, first in the message header and, secondly, in the exchange (as an exchange property). During message processing, this data can be used in various ways. For example, with the content modifier integration flow step, you can write data into the message header or (as a property) into the exchange container in a well-defined way so that it can be accessed at a later step during the message processing (e.g., in a routing condition). We've shown how to use the content modifier several times within this book.

Dynamic configuration of an attribute now means that instead of a concrete value, you enter a reference to a certain header or property for this attribute. At runtime, SAP Cloud Platform Integration resolves this reference by looking up the actual value of this header or property in the incoming message. The term *incoming message* in this case relates to the actual instance of the message that exists when message processing reaches the adapter or step that contains the dynamically configured attribute. [Figure 6.9](#) shows the general principle.

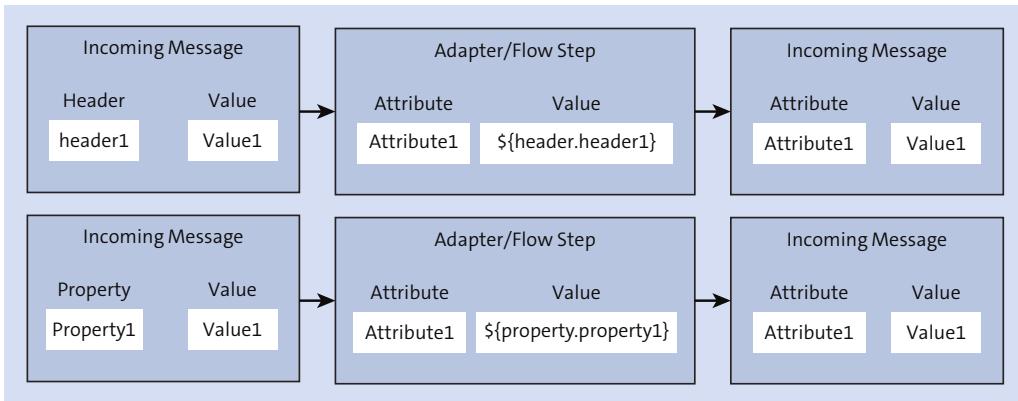


Figure 6.9 How Dynamically Configured Attributes Are Processed during Runtime

The upper part shows how a dynamically configured attribute is handled when a message header is used, and the lower part of the figure shows the same for when a property is used. To refer to a message header or property, you need to apply a certain syntax, which you already have seen in this book:

- The expression `${header.header1}` points to the field `header1` in the incoming message.
- The expression `${property.property1}` points to the property `property1` in the message exchange container.

You remember that expressions such as `${header.header1}` are expressions from Camel Simple Expression Language (described at <http://camel.apache.org/simple.html>), as has been already introduced in [Chapter 4, Section 4.1.4](#).

You've seen already specific usage of dynamically configured attributes in the following sections:

- In [Chapter 4, Section 4.3.4](#), in the context of the OData adapter, we showed you how to dynamically configure a filter condition for an OData request by referring to a specific header.
- In [Chapter 5, Section 5.1.2](#), a specific header was used in a routing condition to allow you to configure a content-based routing (CBR) scenario.
- Furthermore, in [Chapter 5, Section 5.4.3](#), we showed you how to use a specific header (`SAPJMSRetries`) to access the number of already executed retries in a scenario when Java Message Service (JMS) queues are used.

Therefore, the topic of dynamic configuration isn't completely new. In this section, we show you some more simple examples of how to use dynamic parameters, and we provide more background information.

Before going into the details, we would like to mention that many, but not all, integration flow attributes support dynamic configuration. To get a list of integration flow attributes that support dynamic configuration, check out the documentation for SAP Cloud Platform Integration (https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud) and search for the topic **Dynamic Parameters**.

6.2.1 An Integration Flow with a Dynamically Configured Attribute

To show how to use dynamic attributes step by step, we provide a simple example. We won't create a new demo example from scratch, but we'll show you how to extend an already introduced integration flow (from [Chapter 4, Section 4.3](#)).

This is how the integration scenario works: your integration flow invokes an OData service to retrieve information about a certain order and sends this information (using the mail receiver adapter) to an email account. The first part of the scenario (invoking the OData service) is identical to the scenario we've explained already in [Chapter 4, Section 4.3](#). In this example, you'll configure the **Subject** field of the outbound email dynamically to contain the name of the country associated with the order. [Figure 6.10](#) illustrates how the dynamically configured **Subject** field is handled during runtime.

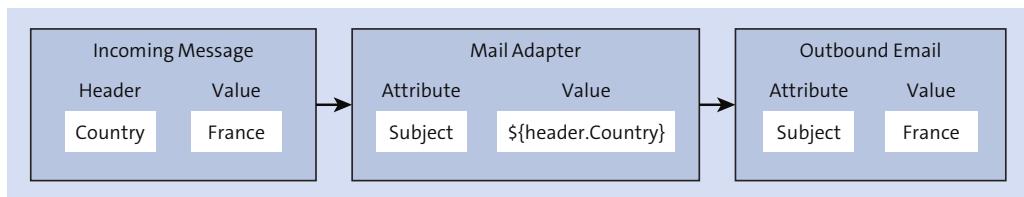


Figure 6.10 Dynamically Configuring the Subject Field of the Receiver Mail Adapter

[Figure 6.11](#) shows the integration flow.

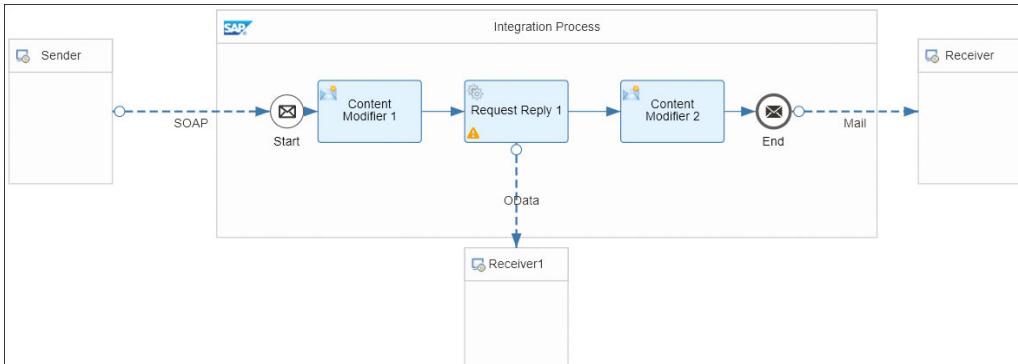


Figure 6.11 Target Integration Flow

This is how the integration scenario is supposed to work at runtime:

1. With a SOAP client (implemented by the SoapUI), you send a SOAP message containing an order number (e.g., 10249).
2. Using the first content modifier, the order number is retrieved from the incoming message and stored in the message's header field for later reference.
3. The **Request-Reply** step invokes an external OData service through the OData channel to retrieve the details of that particular order. In essence, the scenario implements a request-reply pattern scenario. The OData adapter is configured in such a way that certain details of the order are retrieved, among them also the country related to the order, which is contained in the `ShipCountry` field of the order.
4. The second content modifier writes the value of the `ShipCountry` field into the message header field `Country`.
5. The mail receiver adapter is configured so that the value of the **Subject** attribute is dynamically set based on the header `Country`.
6. The outbound email contains the **Subject** with the actual name of the country for the specified order number (e.g., **Germany** for order number 10249).

How to configure the first part of this integration flow was already shown in [Chapter 4, Section 4.3](#). Therefore, we refer to this section for the details. We'll now explain each step for configuring the rest of the integration flow, starting with the second content modifier step.

Let's assume you've already configured the integration flow as shown in [Chapter 4, Section 4.3](#). For your convenience, we again show the attributes of the OData adapter in [Figure 6.12](#).

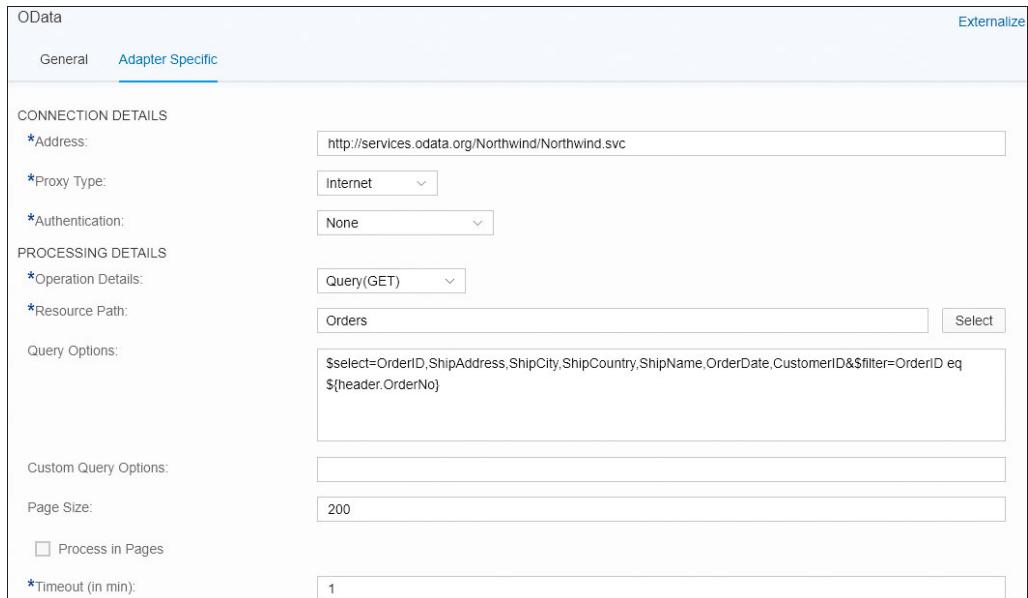


Figure 6.12 OData Adapter Configuration (Example)

Note

Consider the following regarding the configuration of the OData adapter:

- You can, of course, configure the OData adapter differently than explained in [Chapter 4, Section 4.3](#). However, make sure that you also specify the field `Ship-Country` (to be retrieved during the OData request), as shown in [Figure 6.12](#).
- The expression `${header.OrderNo}` in the field **Query Options** makes sure that the order number from the header of the incoming message is retrieved at run-time (and the OData request gets the order for exactly this order number). As mentioned, this expression indicates that we've already intrinsically used a dynamic parameter at this step.

To enhance the integration flow from [Chapter 4, Section 4.3](#), proceed as follows:

1. Edit the integration flow and add a second **Content Modifier** between the **Request-Reply** step and the **Message End** event (as shown in earlier [Figure 6.11](#)).
2. Configure the second **Content Modifier** so that the value of the ShipCountry field of the OData message is written into the newly defined message header Country ([Figure 6.13](#)).



Figure 6.13 Setting the Header in the Second Content Modifier

3. Connect the **Content Modifier** with the message **End** event.
 4. Add a second **Receiver** (refer to [Figure 6.11](#)).
 5. Connect the message **End** event with the second **Receiver**, and specify **Mail** as the adapter type.
 6. Configure the mail adapter as shown in [Chapter 2, Section 2.3.4](#). [Figure 6.14](#) shows the **Connection** settings of the adapter.
You can use the same mail adapter configuration as shown in [Chapter 2, Section 2.3.4](#), except you should enter “\${header.Country}” for the **Subject**.
 7. Save and deploy the integration flow.
 8. Run the integration flow (specifying a certain order number, e.g., 10249).
 9. Check in your mail Inbox for the received message. It should look similar to that shown in [Figure 6.15](#).
- Notice that the email contains the subject **Germany** (as order number 10249 identifies a German order).

Mail

General **Connection** Security

CONNECTION DETAILS

*Address: smtp.gmail.com:465

Proxy Type: None

*Timeout (in ms): 30000

*Protection: SMTPS

*Authentication: Plain User/Password

*Credential Name: FirstnameLastname

MAIL ATTRIBUTES

*From: Fn.Ln@gmail.com

*To: Fn.Ln@gmail.com

Cc:

Bcc:

Subject: \${header.Country}

Mail Body: \${in.body}

Figure 6.14 Mail Adapter with Dynamically Configured Subject Attribute

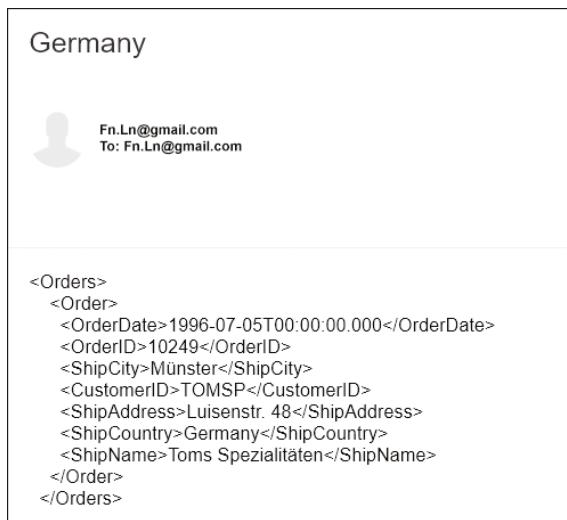


Figure 6.15 Email with Subject Germany

10. Run the integration flow again, this time entering order number “10250”.
11. You should receive an email that looks similar to the one shown in [Figure 6.16](#), this time with the Subject **Brazil** (as the order number 10250 identifies a Brazilian order).

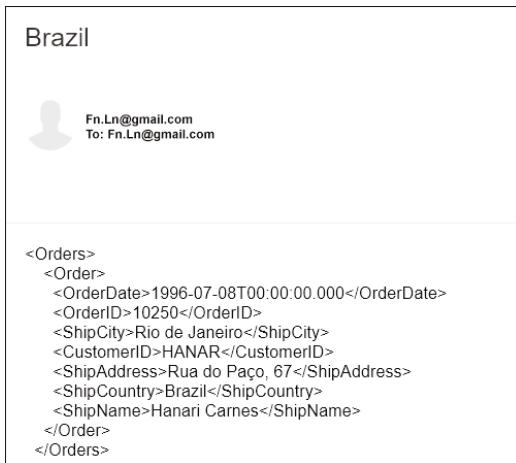


Figure 6.16 Email with Subject Brazil

This shows that you've correctly configured your dynamic parameter for the mail **Subject**.

Note that the mail adapter isn't the only part of the integration flow where a value is dynamically retrieved at runtime. As already mentioned, the OData adapter uses the expression \${header.OrderNo} in the filter condition (refer to [Figure 6.12](#)), which has the effect that the OData request retrieves the order for exactly that order number value provided in the incoming message. Strictly speaking, the integration flow developed in this section contains two dynamically configured attributes.

6.2.2 Monitoring Dynamically Configured Attributes at Runtime

Next we'll show how you can monitor what happens with dynamically configured attributes during message processing. For this purpose, you need to look into the message content during message processing. As we'll show in detail in [Chapter 8, Section 8.2.2](#), the monitoring application provides the option to configure different log levels to specify in which detail processing-related information is to be displayed during monitoring. For the following exercise, it's required that you set the log level

Trace that makes sure payloads and headers are logged after each processing step. This log level is only activated for a short period.

In addition, you need to make sure that your user has the right permissions to display the message content (as provided by this log level). In particular, the authorization group `AuthGroup.BusinessExpert` (or the role `esbmessagestorage.read`) has to be assigned to your user. How authorization groups or roles are assigned to a user is shown in detail in [Chapter 10, Section 10.3](#).

For the following steps, we assume that you've activated the log level **Trace** for your tenant (see [Chapter 8, Section 8.2.2](#)). Proceed as follows:

1. Go to the monitoring application of the Web UI, and under **Manage Integration Content**, select your integration flow.
2. As **Log Level**, select **Trace**. Note that after this activity, this log level setting is effective only for 10 minutes to save resources.
3. Run the integration flow again, and choose a tile under **Monitor Message Processing**.
4. Select your message flow, and click the **Trace** link in the **Logs** section.

An overview of the message processing steps and the integration flow model appears below the list of steps ([Figure 6.17](#)).

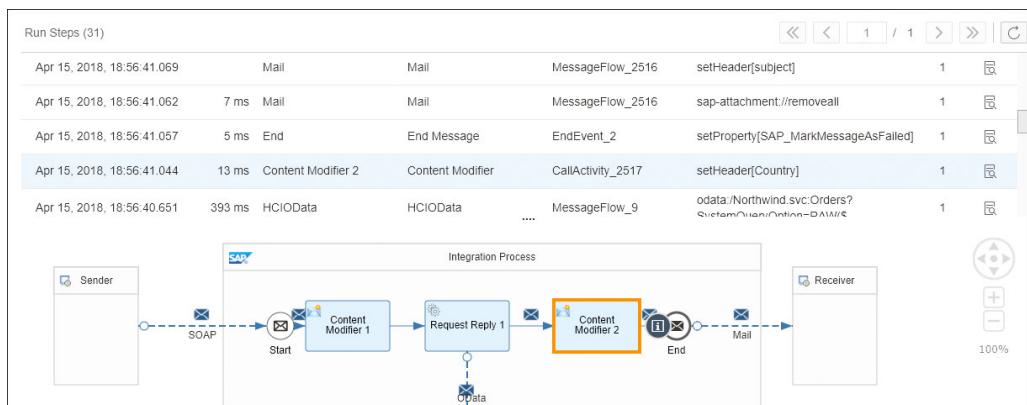


Figure 6.17 The Monitoring Application Shows the Individual Steps during Message Processing

5. Click on the entry for the second **Content Modifier**. The corresponding shape is highlighted in the integration flow model, as shown in [Figure 6.17](#).

6. Click the **View Step Details**  icon under **Actions** at the right end of the table row.
7. Select **Message Content** (the **Message Content** tab is shown only when you've specified a log level **Trace**,).

The message header fields (on top) and the message payload (at the bottom) are shown. [Figure 6.18](#) shows the section with the message header for the selected step shown in [Figure 6.17](#).

Log	Configuration	Message Content
Message before Step		
Header		
Name	Value	
Address	http://services.odata.org/Northwind	
AuthenticationType	None	
clientSidePageSize	200	
ComponentContentType	xml	
Content-Type	xml	
destinationAlias	IGNORE	
destinationManagerImpl	com.sap.it.rt.adapter.odata.destination.HCIDestinationManagerImpl@5f36c0ed	
HttpStatusCodes	OK	
ODataMethod	GET_FEED	
ODataResponseType	Properties	
OrderNo	10250	
retrieveAllPages	true	
SAP_MessageProcessingLogID	AFrU38njcRE-QgeBw7OterV1eNMF	

Figure 6.18 Actual Message Header at the Second Content Modifier

Below the header, you find the message payload with the data of the actual order (not shown in [Figure 6.18](#)), but we'll keep our focus on the header. It contains several fields related to the OData request, such as the address of the OData source, the request method, and other fields we won't elaborate on. But what is important for the actual discussion is that at this processing step, the header **Country** isn't shown yet, which is as expected if you remember that this header has just been created by the second **Content Modifier** step.

Now, let's see how the message has changed with the following processing step:

1. Navigate back from the step details to the message processing overview. To do that, click the **Message Processing Run** link in the breadcrumb link at the top of the screen.
- Notice the overview with the integration flow model shown earlier in [Figure 6.17](#).
2. Select the **End Message** step (one row on top of the second **Content Modifier** in [Figure 6.18](#)).
3. Click the **View Step Details**  icon under **Actions** at the right end of the table row.

The **Header** section again shows various header fields. Notice that now the header **Country** has been added ([Figure 6.19](#)) as you would also expect from how the integration flow has been designed.

Log	Configuration	Message Content
Message before Step		
Header		
Name	Value	
Address	http://services.odata.org/Northwind	
AuthenticationType	None	
clientSidePageSize	200	
ComponentContentType	xml	
Content-Type	xml	
Country	Brazil	
destinationAlias	IGNORE	
destinationManagerImpl	com.sap.it.rt.adapter.odata.destination.HCIDestinationManagerImpl@5f36c0ed	
HttpStatusCodes	OK	
ODataMethod	GET_FEED	
ODataResponseType	Properties	
OrderNo	10250	
retrieveAllPages	true	
SAP_MessageProcessingLogID	AFrU38njcRE-QgeBw7OterV1eNMf	

Figure 6.19 Message Headers during Message Processing before the Message End Event

Note

In addition, for the actual message flow, the header **Country** has been set now and has the value **Brazil**. This value is given over to the mail adapter.

We've shown you with a simple example how you can easily dynamically configure a certain integration flow attribute (by referring to a certain header) and how you can monitor the usage of this header during message processing.

At runtime, the actual value for this header is retrieved from the incoming message and used to (dynamically) set an attribute of a subsequent step (for our example, of the mail adapter). Note that in this example, we explicitly had to create the header during message processing (with the second **Content Modifier** step) just to create a data container for the data to be used to dynamically set the mail adapter attribute.

There's more to tell about dynamic parameters. In addition to using a specially created header to dynamically configure a certain integration flow attribute, you can use a number of *predefined* headers and exchange properties that are provided by the SAP Cloud Platform Integration framework to retrieve specific data during the processing of a message. In the same way as shown in this section, you can also use these special headers and properties to dynamically configure integration flow attributes.

6.2.3 Using Predefined Headers and Properties to Retrieve Specific Data Provided by the Integration Framework

In [Chapter 5, Section 5.4.3](#), you already came across an example for a header predefined by the integration framework when using the JMS adapter. The header field SAPJMSRetries is automatically created by the integration framework when you use JMS queues. It records the number of retries of a JMS message that are already executed. In [Chapter 5, Section 5.4.3](#), we showed you how to use the content of this header to dynamically define a routing condition (by entering the expression \${header.SAPJMSRetries}).

This header is only set when using the JMS adapter. Similarly, other adapters and integration flow steps set other, specific headers and properties to store data that is specific to this adapter or integration flow step.

To get an overview of the predefined headers and properties provided by the SAP Cloud Platform Integration framework, check out the documentation for SAP Cloud Platform Integration at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/

Cloud, and search for the topic **Headers and Exchange Properties Provided by the Integration Framework**.

To round out this topic, we'll show you how to use another header. As you can see from the documentation of SAP Cloud Platform Integration, the splitter step also creates the exchange property `CamelSplitSize`, which provides the total number of split items of an exchange. To show how to use this property to dynamically configure a certain integration flow attribute, refer to the integration flow explained in [Chapter 5, Section 5.2](#) (see Figure 5.18).

To briefly recap, the integration flow splits a single message that contains multiple order items (as depicted in [Chapter 5](#) in Figure 5.17) into individual messages with one order item per message (by applying the **General Splitter**). Each split message is enriched by additional details (by an OData request to an external source, as depicted in [Chapter 5](#), Figure 5.37). Finally, the split messages are joined to a single one (with multiple items) by the **Gather** step. However, we'll skip the message enrichment and only use the very simple, first variant of the integration flow as the basis for our integration scenario, as depicted in [Chapter 5](#), Figure 5.18.

We'll now modify the integration scenario shown in [Chapter 5](#), Figure 5.18, in the following way: After having split the larger message into individual messages and joining it back into a single one (as depicted in [Chapter 5](#), Figure 5.29), we introduce a **Routing** step that sends the final message to a dedicated email receiver (with a specific email **Subject**) in case the message contains exactly three items. In all other cases (when the joined message contains a different number of items than three), the message is forwarded to another email receiver (to keep it simple, we in fact use the same email account for both receivers, but we only specify another email **Subject** in the other mail adapter). The integration flow is shown in [Figure 6.20](#).

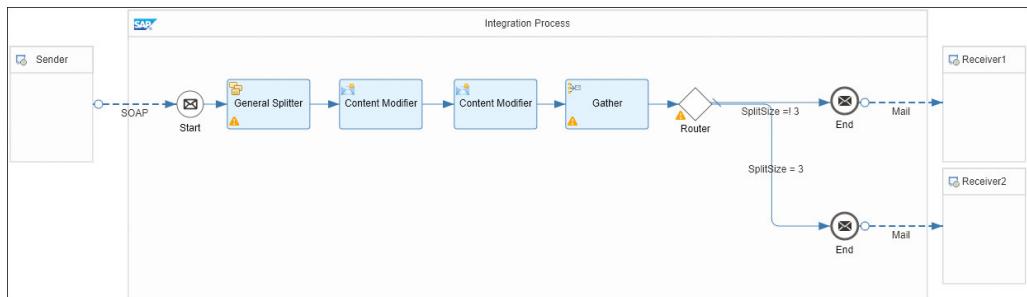


Figure 6.20 Message Split Integration Flow with an Additional Routing Step and Two Email Receivers

We won't explain in detail how to modify the various integration flow steps, as you're already an expert in these topics. We'll just make you aware of the following aspects:

- In the **General Splitter** dialog box, deselect the **Streaming** checkbox ([Figure 6.21](#)).

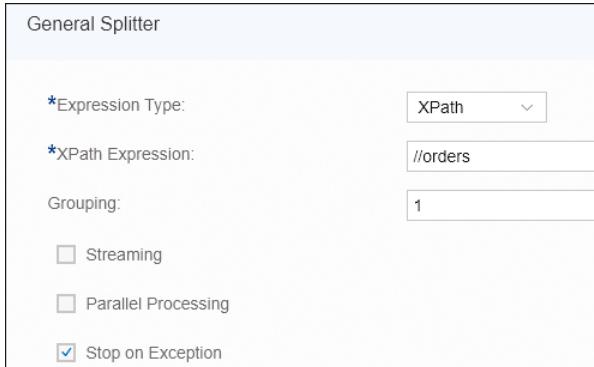
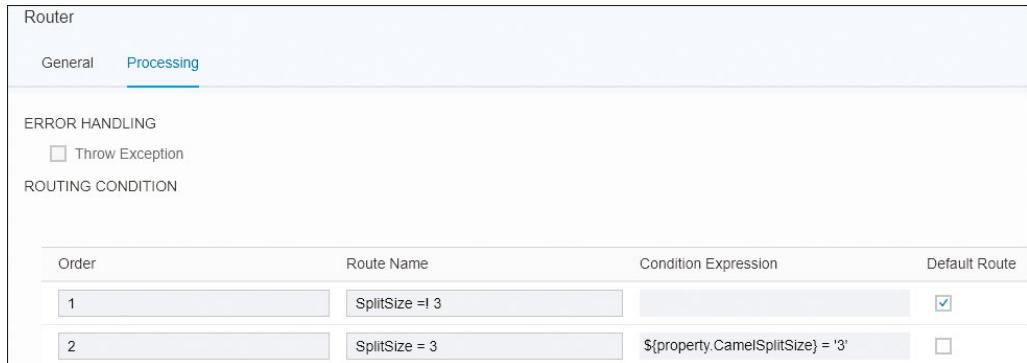


Figure 6.21 Deselecting the Streaming Checkbox in the General Splitter Dialog Box

In [Chapter 5, Section 5.2.1](#) ([Figure 5.24](#)), we kept this option selected. However, when using the property `CamelSplitSize`, we propose to disable the **Streaming** option because the property `CamelSplitSize` is only applied on the complete exchange if **Streaming** is enabled.

- You can use the same mail adapter settings as already shown at various times within this book (e.g. [Chapter 2](#), in [Figure 2.31](#) and [Figure 2.32](#)). Note, however, that you specify two different email subjects in the two different mail adapters:
 - In the upper mail adapter to which messages with the condition `SplitSize = 3` (refer to [Figure 6.20](#)) are routed, enter, for example, “No of items =! 3” as the **Subject**.
 - In the lower mail adapter to which messages with the condition `SplitSize = 3` are routed, enter, for example, “No of items = 3” as the **Subject**.
- We also explicitly show how the routing conditions should be defined in [Figure 6.22](#).

Notice that for the lower route, we enter the routing condition “ `${property.CamelSplitSize} = '3'`”. The property `CamelSplitSize` wasn't defined at any previous step in the integration flow (e.g., in a **Content Modifier**). When running the scenario, this property is set automatically and provides the specific information related to the processed message (in this case, the total number of split items).

**Figure 6.22** Routing Conditions

Now, run the scenario with the following input message (to be edited in SoapUI), as shown in [Figure 6.23](#).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
<soapenv:Header/>
<soapenv:Body>
  <demo:OrderList_MT>
    <orders>
      <orderNumber>10251</orderNumber>
      <customerName>?</customerName>
      <orderAmount>?</orderAmount>
      <currency>?</currency>
      <taxAmount>?</taxAmount>
    </orders>
    <orders>
      <orderNumber>10252</orderNumber>
      <customerName>?</customerName>
      <orderAmount>?</orderAmount>
      <currency>?</currency>
      <taxAmount>?</taxAmount>
    </orders>
  </demo:OrderList_MT><demo:OrderList_MT>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 6.23 Input Message with Two Items

The input message contains two items. The result (as received in your email account) should look like [Figure 6.24](#).

No of items != 3

 Fn.Ln@gmail.com
To: Fn.Ln@gmail.com

```
<?xml version="1.0" encoding="UTF-8"?><mymap:Messages xm
mymap:Message1><orderDetails>
10251
</orderDetails><orderDetails>
10252
</orderDetails></mymap:Message1></mymap:Messages>
```

Figure 6.24 Email with Two Split Items and Subject “No of Items != 3”

The email subject is **No of items != 3**, which proves that the message has taken the upper route in [Figure 6.20](#), shown previously.

Repeat the same with the input message shown in [Figure 6.25](#).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderList_MT>
      <orders>
        <orderNumber>10250</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10251</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
      <orders>
        <orderNumber>10252</orderNumber>
        <customerName>?</customerName>
        <orderAmount>?</orderAmount>
        <currency>?</currency>
        <taxAmount>?</taxAmount>
      </orders>
    </demo:OrderList_MT><demo:OrderList_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 6.25 Input Message with Three Items

As the input message now has three items (processed by the **General Splitter**), the result (as received in your email account) should look like [Figure 6.26](#).



Figure 6.26 Email with Three Split Items and Subject “No of Items = 3”

The email subject is **No of items = 3**, which proves that the message has taken the lower route in [Figure 6.20](#), shown earlier.

That's it! You've now seen how to dynamically configure an integration flow attribute by using data provided by the integration framework by one of the various pre-defined headers.

You're now well equipped with knowledge to design more sophisticated scenarios. The more creative you are with modeling even more complicated message flows, the higher the risk is that you'll easily lose the overview of the overall integration flow. In the subsequent sections, we show you some options to manage complexity by modularizing your integration logics into smaller chunks by using subprocesses. Thereafter, we introduce the ProcessDirect adapter as an option to implement direct communication between different integration flows (on the same tenant), which also can be a good approach to modularizing a comprehensive integration scenario into smaller pieces.

6.3 Structuring Large Integration Flows Using Local Processes

With SAP Cloud Platform Integration, you can model fairly large integration scenarios. Due to the flexible pipeline of the underlying Apache Camel integration engine,

you could potentially add as many processing steps to your route as are necessary to fulfill your integration needs. However, as we're using a graphical modeler, those large models can easily become confusing, and with that, you lose all the benefits of a graphical notation. In this section of the book, which is about modeling and running integration flows on SAP Platform Cloud Integration, you'll learn how to structure large process models using subprocesses.

6.3.1 Taking Hold of Complexity by Modularization

It's never a good idea to put too much logic in one module. This holds true for classical programming languages, such as Java, as well as for graphical environments, such as the Web UI being used in SAP Cloud Platform Integration. In your informatics classes, you've certainly learned how to slice large programs into manageable logical units and treat them separately (*separation of concerns*). The same can be applied to graphically modeled integration flows. To achieve this in SAP Cloud Platform Integration, we need to use subprocesses, or *local processes*, as they are called in SAP Cloud Platform Integration. (The terms subprocess and local process (integration) will be used interchangeably throughout this book.)

As you'll see, it isn't that difficult to work with subprocesses. We encourage you to make use of them, simply to keep your individual processes and subprocesses at a reasonable size. As a rule of thumb, process models shouldn't contain more than 10 to 15 elements. If your models become larger, it's recommended to refactor them and reduce their size by moving parts of your model into newly created subprocesses. To apply this rule, you need to know how subprocesses can be modeled and how parameters are exchanged between parent and child processes. As an example, we'll use the process model shown in [Figure 6.27](#).

Let's see how this process works. We'll take a look at the execution sequence first, before diving into the detailed configuration of each step. The main process, modeled at the top of the diagram, is triggered by an incoming SOAP message. A **Content Modifier** step will set some header and exchange properties, as we've done several times before in this book. The exchange is the central container carrying all necessary data, including the message's payload and header information from step to step inside of the integration flow.

Returning to our model of [Figure 6.27](#), the main process invokes the subprocess entitled **Execute Business Logic**. Within the subprocess, a **Content Modifier** step is used to work with the variables being set in the parent process, simply to highlight the availability of those variables in the child process, although they have been set in the

parent process. Additionally, the first **Content Modifier** step in the local process will add a new header variable to the exchange. The goal is to demonstrate how variables created in a child process will also be available in the parent process after the subprocess has finished. To add some more logic, a gateway (the diamond shape in the **Execute Business Logic** subprocess of Figure 6.27) representing a CBR is used to distinguish between different order number ranges (see the respective labels at the two gates).

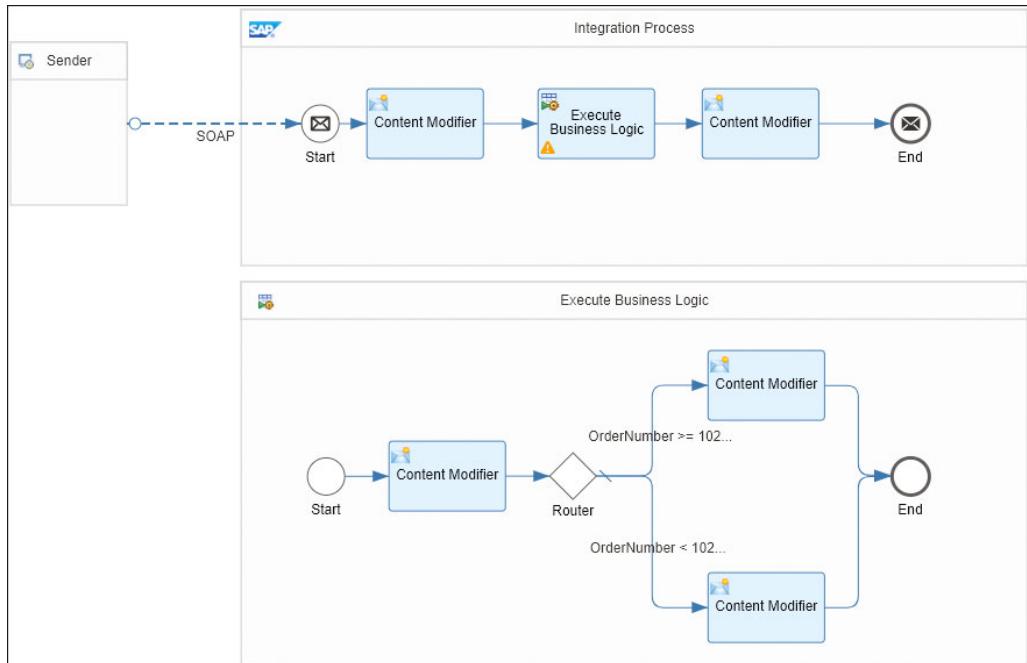


Figure 6.27 Process Model Calling the “Execute Business Logic” Local Process

The two **Content Modifier** steps following the gateway set the content of the reply message. Once executed, the subprocess is finished, and the process execution continues in the main process by calling the last **Content Modifier** step. Here we'll access the variable set in the local process. It verifies its availability although the subprocess has already been finished.

We take the well-known structure we've used in several scenarios before, which is shown again in Figure 6.28 for your convenience, as the input message.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <demo:OrderNumber_MT>
      <orderNumber>10249</orderNumber>
    </demo:OrderNumber_MT>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 6.28 Input Message

We're really focusing on the cooperation between parent and child processes, as well as how the parameter transfer between the two works. Let's see how to configure the individual steps to make the collaboration executable.

6.3.2 Configuring the Collaboration between Parent and Child Processes

We begin with the configuration of the first **Content Modifier** step in the main process. The settings are shown in [Figure 6.29](#) and [Figure 6.30](#).

The screenshot shows the 'Content Modifier' configuration dialog with the 'Message Header' tab selected. The 'Headers:' section contains a table with one row. The table has columns for Action, Name, Type, Data Type, Value, and Default. The row shows 'Create' in the Action column, 'orderNumber' in the Name column, 'XPath' in the Type column, 'java.lang.String' in the Data Type column, and '/orderNumber' in the Value column. There are 'Add' and 'Delete' buttons at the top right of the table.

Action	Name	Type	Data Type	Value	Default
Create	orderNumber	XPath	java.lang.String	/orderNumber	

Figure 6.29 Setting a Message Header

The screenshot shows the 'Content Modifier' configuration dialog with the 'Exchange Property' tab selected. The 'Properties:' section contains a table with one row. The table has columns for Action, Name, Type, Data Type, Value, and Default. The row shows 'Create' in the Action column, 'msg' in the Name column, 'Expression' in the Type column, and '\${in.body}' in the Value column. There are 'Add' and 'Delete' buttons at the top right of the table.

Action	Name	Type	Data Type	Value	Default
Create	msg	Expression		\${in.body}	

Figure 6.30 Setting an Exchange Property

Two variables are being set here: one with the name **orderNumber** in the **Name** field of the **Message Header** area ([Figure 6.29](#)), and the other with the **msg** in the **Name** field of the **Exchange Property** area ([Figure 6.30](#)). This should remind you of the very first scenario that we built back in [Chapter 4](#), where we did exactly the same thing. As a result, our exchange will contain the two variables in their respective locations. Now comes the interesting part: the integration flow invokes the subprocess. So, how

do you model the subprocess and its invocation? We need to begin with the subprocess first. This is important because the parent process will have to reference the subprocess later. Hence, the subprocess must be in place or such a reference can't be established. Follow these steps:

1. Model the subprocess alongside the main process by picking the **Local Integration Process** entry from the palette, which can be found beneath the **Process** main menu entry ([Figure 6.31](#)).

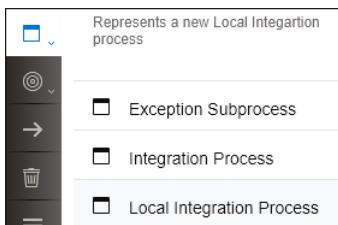


Figure 6.31 Modeling a Local Integration Process

2. After you've positioned the subprocess beneath the main process, you'll get a new pool containing an empty flow ([Figure 6.32](#)).



Figure 6.32 Newly Positioned Local Integration Flow

Note the new  icon in the upper-left corner of the local process, signifying it as a subprocess, which can't be started by an incoming message or by a **Timer Start** event. It can only be invoked from a parent process by a respective **Process Call** shape, which we'll explain soon. Because of this invocation relationship to a parent process, the subprocess needs to start with an empty **Start** event . The only attribute you can change when selecting the subprocess is its name. You should adjust it and give it a self-explanatory name. Within the subprocess, you can model any integration logic, as you would for the main process.

Limitations of Local Integration Processes

Note that there are some limitations when using local integration processes. You can't use the following integration flow components within a local integration process:

- Aggregator
- Process call
- End message event

Furthermore, using the **Splitter** step in a local integration process also requires some specific considerations because it behaves differently compared to when it's used in the main process. For more information, see the "Cloud Integration – Usage of Splitter Flow Steps in Local Process" blog in the SAP Community (www.sap.com/community.html).

Next, we need to model the invocation of the local integration process from the main process—the referencing of the child process from the parent process we were talking about previously. This is done in the following way:

1. In the palette, click the **Call** icon  to open the **Call** submenu (Figure 6.33).

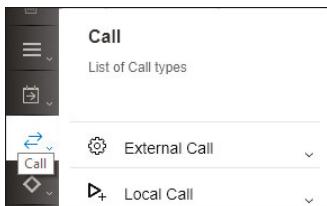


Figure 6.33 The Call Submenu from the Palette

2. Click **Local Call** to open another submenu (Figure 6.34).

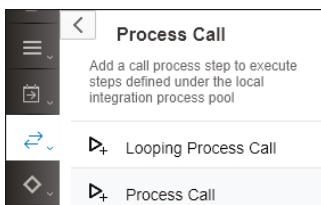


Figure 6.34 The Process Call Shape Located Under Local Call

3. Position the **Process Call** shape shown in Figure 6.34 inside the main process.

The last step is to connect the newly positioned **Process Call** shape with the subprocess itself. This is done by selecting the **Process Call** rounded rectangle in the main process and adjusting the **Local Integration Process** field in the associated properties area beneath the process model in the **Processing** tab. Click on the **Select** button to open another dialog box listing all modeled local integration processes. Pick the one you want to invoke (as we've modeled only one local process, there should only be one entry). The dialog box closes automatically after you've chosen one entry from the list ([Figure 6.35](#)).



Figure 6.35 Connecting the Process Call Step with the Subprocess

That's all you need to do to model a subprocess invocation from the main process. However, you may wonder whether there is a need to define an interface for your subprocess that describes which data the subprocess expects from its parent process and which data it will return after it finishes its execution. The answer is that you don't have to define such an interface because the called subprocess also relies on the same exchange the main process is working on, which is automatically handed over from step to step within the main process as well as within the subprocess. This again stresses the importance of the exchange as *the central data container* within integration flows while working with SAP Cloud Platform Integration and its underlying Camel framework.

It should be clear now how data transfer between parent and child processes works. There is no need for local or global variables, as you might typically find in programming languages. The only container carrying variables and their values is the exchange, which is being transferred back and forth between parent and child.

We continue with the configuration of the first **Content Modifier** inside of the local integration flow **Execute Business Logic** (refer to [Figure 6.27](#)). Its configuration is shown in [Figure 6.36](#).



Figure 6.36 Setting the Message's Payload in the Subprocess

We set the message's payload by filling its body with two XML tags and the contents of the two variables, which were set previously in the parent process. By doing this, we demonstrate the availability of data in the child process, which has been set before by the parent process. To display the data transfer in the reverse direction (from child to parent process), we create a new variable named **VarFromSubprocess** within the same **Content Modifier** (Figure 6.37 and Figure 6.38).

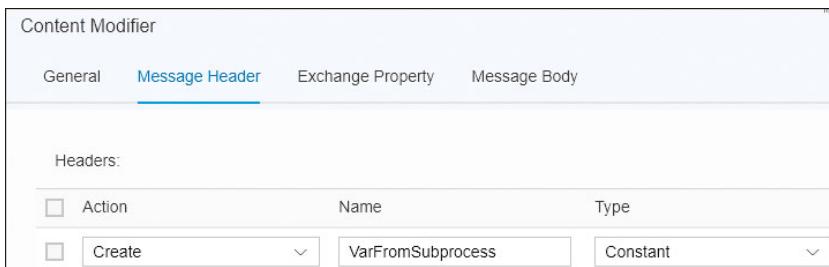


Figure 6.37 Subprocess Setting a Variable in the Message Header Area of the Exchange A

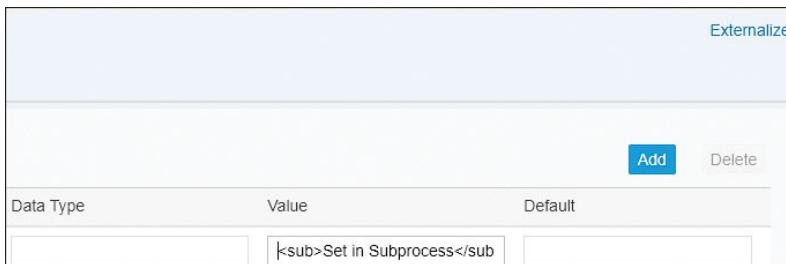


Figure 6.38 Subprocess Setting a Variable in the Message Header Area of the Exchange B

The new variable just contains a String constant, which will later be added to the response message by the parent process.

Following the first **Content Modifier** in the subprocess, a CBR, represented by the diamond-shaped **Exclusive Gateway**, takes care of adding more information to the response message. The gateway's configuration is depicted in [Figure 6.39](#).

The screenshot shows the configuration of an Exclusive Gateway in a Router. It includes sections for Error Handling (Raise Alert and Throw Exception), Routing Condition, and a table for defining routes based on Order Number.

Order	Route Name	Condition Expression	Default Route
1	OrderNumber < 10250		<input checked="" type="checkbox"/>
2	OrderNumber >= 10250	#{header.orderNumber} >= '10250'	<input type="checkbox"/>

Figure 6.39 Configuration of the Gateway

Note the **Condition Expression** column in the second row of the **Routing Condition** table: for the decision of which route to follow, the condition again relies on the header variable set by the parent process.

We continue with the two **Content Modifier** steps following the gateway. They just add some static text to the already existing payload. The configuration of the upper **Content Modifier** is shown in [Figure 6.40](#).

The screenshot shows the configuration of a Content Modifier. The 'Message Body' tab is selected. The 'Body:' field contains an XML snippet that includes a conditional expression and a reference to the input body.

```
<result2>
    OrderNumber greater equal 10250
    ${in.body}
</result2>
```

Figure 6.40 Setting the Body's Content Using the Content Modifier

Note the two new surrounding XML tags labeled `result2`. They wrap the current payload referenced by the Camel variable `#{in.body}`. In addition, the constant text will later reveal whether the right path was chosen. The lower **Content Modifier** is similarly configured. The text has just been changed to `OrderNumber lower 10250`.

This concludes the explanation of the subprocess. We can continue with the last **Content Modifier** in the main process following the step, which caused the subprocess's invocation. Its settings are straightforward and are shown in [Figure 6.41](#).

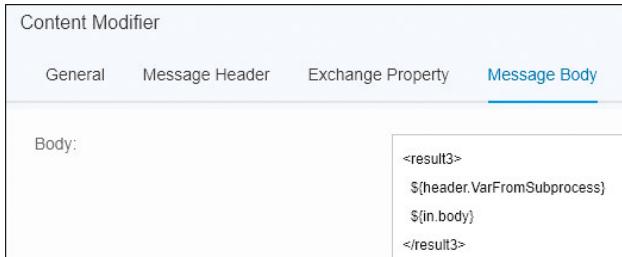


Figure 6.41 Configuration of the Last Content Modifier in the Main Process

What is missing now is proof that the variables being set in the subprocess can be accessed by the parent process. That's why you find the expression \${header.VarFromSubprocess} inside the **Message Body**'s definition. It accesses the variable, which we set before in the called local integration process (refer to [Figure 6.37](#)). Note also the new XML tags labeled with result3, which again wrap the current payload plus the contents of the variable VarFromSubprocess. If everything works correctly, we should see three nested tags labeled result3, result2, and result1, respectively. So, let's see that integration flow in action. After invoking our demo process, you'll see the reply depicted in [Figure 6.42](#).

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <result3>

      <sub>Set in Subprocess</sub>

      <result2>
        OrderNumber greater equal 10250

        <result1>
          10250

          <demo:OrderNumber_MT xmlns:demo="http://cpi.sap.com/demo" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
            <orderNumber>10250</orderNumber>
          </demo:OrderNumber_MT>
        </result1>
      </result2>
    </result3>
  </soap:Body>
</soap:Envelope>
```

Figure 6.42 Response Message Produced by the Integration Flow

Everything worked as expected. The result tags are nested, and depending on the entered order number, you'll get the respective text message whether the number was lower than 10250.

To summarize, real-life integration scenarios can grow quite large. That's why a means to structure large process models is urgently required. SAP Cloud Platform Integration supports structuring large process models by using local integration processes to keep each individual process model at a reasonable size. The parameter transfer between parent and child processes is solved by the exchange, the standard container for managing data within an integration flow. As we've seen in the previous exercise, the exchange isn't only handed over from step to step, on one process level. It's also the vehicle for moving data around from parent to child processes and vice versa. This makes the definition of global or local variables superfluous. You're now able to model, run, and monitor really complex scenarios. If you follow the recommendations given in this section, you'll also ensure manageable process sizes, making it fun to work with.

6.3.3 Using Exception Subprocesses

The exception is a special kind of subprocess you can use to handle error situations that occur during message processing. We use a very simple integration flow that contains nothing more than a **SOAP Sender** and a **Mail Receiver** adapter as shown in [Figure 6.43](#) (similar to the first integration flow shown in [Chapter 2](#)).

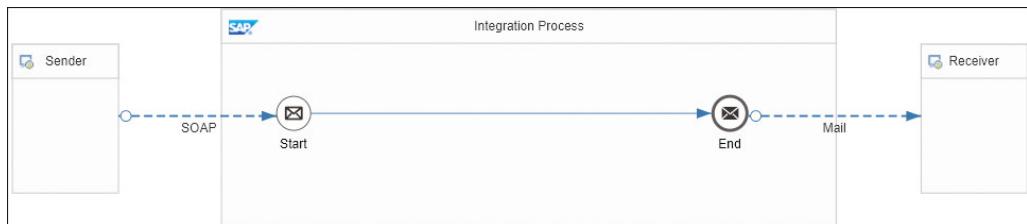


Figure 6.43 Simple Integration Flow with SOAP Sender and Mail Receiver

For the configuration of the mail adapter, we assume that a **User Credentials** artifact has been deployed that contains the credentials to access the email account. Follow these steps to modify the integration flow:

1. Add an exception subprocess to the integration flow by selecting **Exception Subprocess** under the **Process** root node ([Figure 6.44](#)).

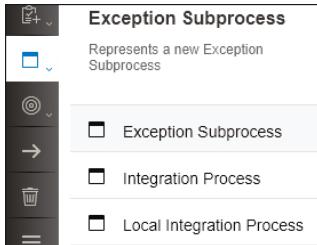


Figure 6.44 Selecting an Exception Subprocess in the Palette

2. You can only place the **Exception Subprocess** in the **Integration Process** shape, which results in the change shown in [Figure 6.45](#).

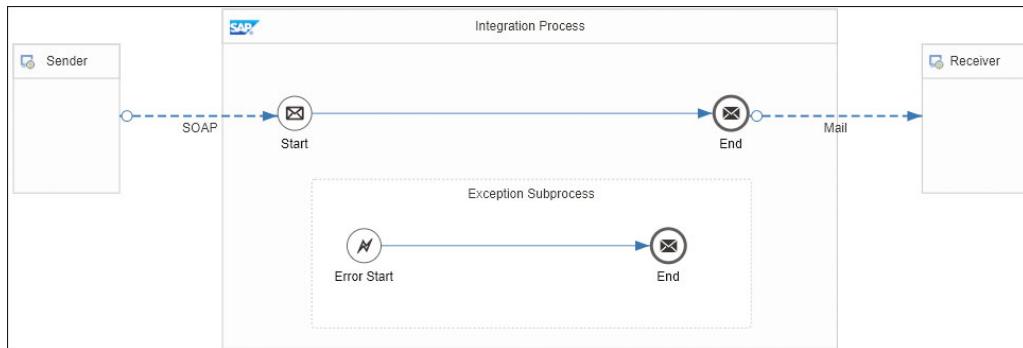


Figure 6.45 Exception Subprocess Placed within the Integration Process Shape

3. Add a **Content Modifier** to the **Exception Subprocess** between the **Error Start** and **End** events.
4. In this step, we define how to handle error situations. To do that, in the **Message Body** tab, specify the setting shown in [Figure 6.46](#).

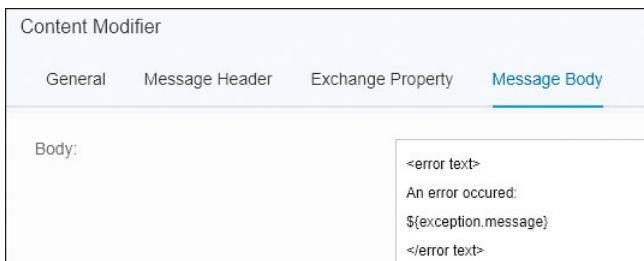


Figure 6.46 Error Message Configured in the Content Modifier in the Exception Subprocess

The expression `exception.message` is a variable provided within Camel's Simple Expression Language that allows you to access information on error situations stored in the exchange (see <http://camel.apache.org/simple.html>).

5. Connect the message **End** event of the **Exception Subprocess** with a second **Receiver**, and select the **Mail** adapter as the connectivity option. You can use the same **Mail** adapter settings as for the first receiver connected to the main **Integration Process** (Figure 6.47).

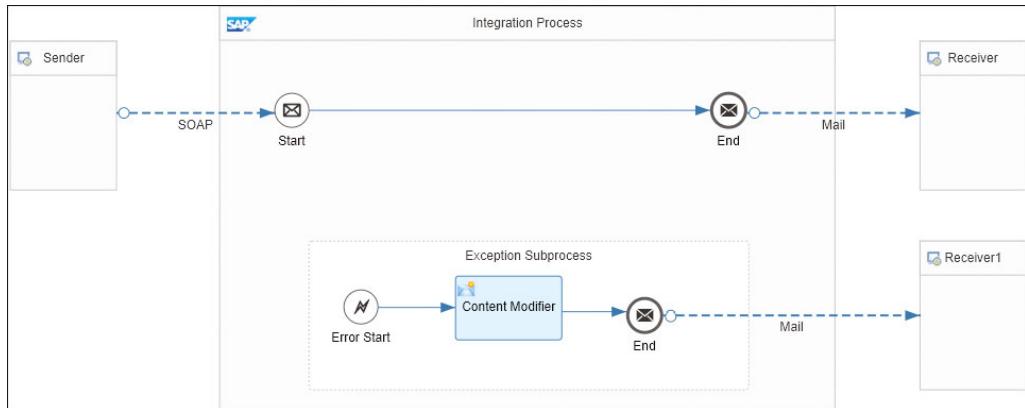


Figure 6.47 Final Integration Flow

Make sure, however, to refer to another **User Credentials** artifact for the **Mail** lower connection in Figure 6.47. The idea is to enforce an exception by editing the **User Credentials** artifact (referenced in the upper **Mail** connection between the **End** message event of the main process and the upper **Receiver**), and enter a wrong password there. Therefore, to make sure the error details are received in your mail account, in the lower **Mail** adapter (connecting the **End** message event of the **Exception Subprocess** with the email server), use another **User Credentials** artifact with the correct credentials. Note that for simplicity, we'll use one and the same email account for the different email receivers like we did in the integration flows before.

To summarize:

- The **Mail** adapter connected to the upper receiver (**Receiver**) refers to a **User Credentials** artifact with wrong credentials.
- The **Mail** adapter connected to the lower receiver (**Receiver1**) refers to a **User Credentials** artifact with correct credentials.

Running the integration flow should result in an error message received in your mail account, as shown in [Figure 6.48](#).

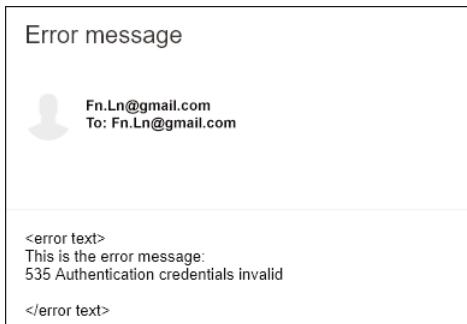


Figure 6.48 Email Containing the Error Message Configured in the Content Modifier of the Exception Subprocess

There's another use case where an exception subprocess comes into play. In [Chapter 5, Section 5.4.3](#), we showed you how to use an exception subprocess for the explicit retry configuration in a scenario with the JMS adapter.

You've now learned how you can modularize integration scenarios by using subprocesses and how to handle error situations by using the exception subprocess.

Next, we'll show you another option for modularizing integration content: the ProcessDirect adapter.

6.4 Connecting Integration Flows Using the ProcessDirect Adapter

In [Section 6.3](#), we showed how you can use subprocesses to modularize larger integration flows. This capability supports you in situations where you need to manage bigger integration projects with comprehensive integration scenarios and distributed responsibilities.

Another option to overcome such complex situations is obvious: designing different parts of the integration logics of a complex scenario in individual, "smaller" integration flows and connecting them with each other. SAP Cloud Platform Integration comes with a variety of different adapters, so you might already have had the idea to use certain adapters to connect different integration flows with each other to build a larger scenario. Obviously, the HTTP-based adapters might be the option of choice. So why not put parts of the integration logics into n different integration flows and con-

nect them with each other; for example, integration flow 1 calls integration flow 2, integration flow 2 calls another one, and so forth. But wait—you also know that each such HTTP connection (which is, to mention one example path, an outbound connection from the perspective of integration flow 1 and an inbound connection from the perspective of integration flow 2) is routed through the load balancer. Check out [Chapter 2, Section 2.1](#), to remind you of that (in particular, see Figure 2.3). This means that another component is involved in each such communication, and this brings about a higher network load so that you can expect the performance of your integration scenario in total to suffer from these extra intra integration flow connections—even if you connect integration flows deployed on the same tenant.

To overcome this issue, the SAP Cloud Platform Integration team has developed a dedicated adapter, the **ProcessDirect** adapter, which allows you to directly connect two integration flows without the load balancer being interconnected. [Figure 6.49](#) illustrates a sending (at left) **Integration Flow** and a receiving (at right) **Integration Flow** that are both connected over the **ProcessDirect** adapter.

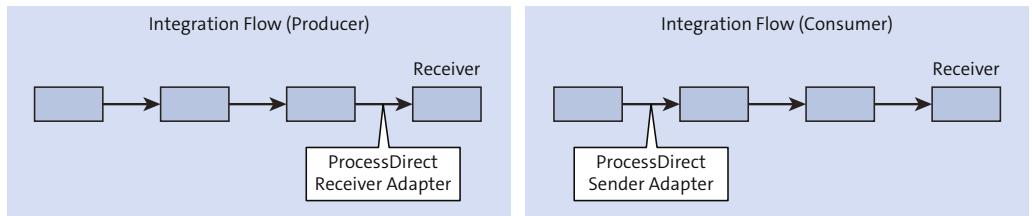


Figure 6.49 Producer and Consumer Integration Flow Communication over ProcessDirect Adapters

To simplify the following discussion, when we're talking about two integration flows connected over the ProcessDirect adapter, we distinguish between a *producer integration flow* and a *consumer integration flow* in the following sense:

- The producer integration flow sends the message to another integration flow over the **ProcessDirect** adapter.

In the producer integration flow, you need to configure a **ProcessDirect** receiver adapter (to send the message to the target integration flow).

- The consumer integration flow receives a message from another integration flow over the **ProcessDirect** adapter.

In the consumer integration flow, you need to configure a **ProcessDirect** sender adapter (to receive the message from the producer integration flow).

Limitations

Note the following limitations for the usage of the **ProcessDirect** adapter:

- Producer and consumer integration flows must be deployed on the same tenant.
- The cardinality of producers to consumers is restricted so that you can send messages from N producers to 1 consumer but not vice versa (cardinality rule reads producer:consumer = N:1).

However, you can use **ProcessDirect** adapters to connect integration flows from different integration packages (as long as they reside on the same tenant).

The **ProcessDirect** adapter is quite simple; it has exactly one attribute, which is the **Address** of the integration flow to connect to (defining the endpoint). We'll discuss how to use this adapter in just a minute, but let's briefly summarize the use cases for this adapter first.

6.4.1 Use Cases for the ProcessDirect Adapter

Following is an overview of possible use cases that you can address with the ProcessDirect adapter:

- **Structuring large integration flows and separation of concerns**

As mentioned previously, similar to how you can use local integration processes to structure larger integration flows (refer to [Section 6.3](#)), you can use separate (“smaller”) integration flows and connect them through ProcessDirect adapters. Note that using separate integration flows (also, if required, within different integration packages), enables you to clearly manage different responsibilities within an integration project (separation of concerns) in the following way: different integration developers can work independently on that part of the integration logic they are responsible for by decomposing the overall scenario into several “smaller” integration flows with different owners.

- **Reuse integration logic**

You can “source out” generic functions and parts of the integration logic that are used at many places into dedicated integration flows. Such (consumer) integration flows can be called from multiple (producer) integration flows independently. As an example of such generic integration logics, think about error-handling strategies. If you need to do changes to the generic part, this doesn't impact the producer integration flows, as long as you don't change the address of the consumer.

In the example of error-handling strategies, N producers can send messages to 1 consumer that contains the error-handling logic.

- **Dynamic endpoint configuration**

As we'll show later, you can dynamically configure the **Address** of the **ProcessDirect** adapter. This leaves room to define sophisticated integration scenarios where it can be dynamically "decided" during runtime which consumer integration flow is called by the producer.

- **Creating multiple message processing logs (MPLs)**

As the overall integration scenario is split into several independent integration flows, during the operations of the scenario, MPLs are generated. This might also make it easier to analyze errors and to also distribute the operations tasks among different people.

6.4.2 A Simple Example

We first show how to use the **ProcessDirect** adapter by modifying the integration scenario that we've used in [Section 6.3](#) to illustrate the usage of local integration processes. We change the design in a way that the overall integration logic is split into two individual integration flows.

Figure 6.50 shows the producer integration flow:

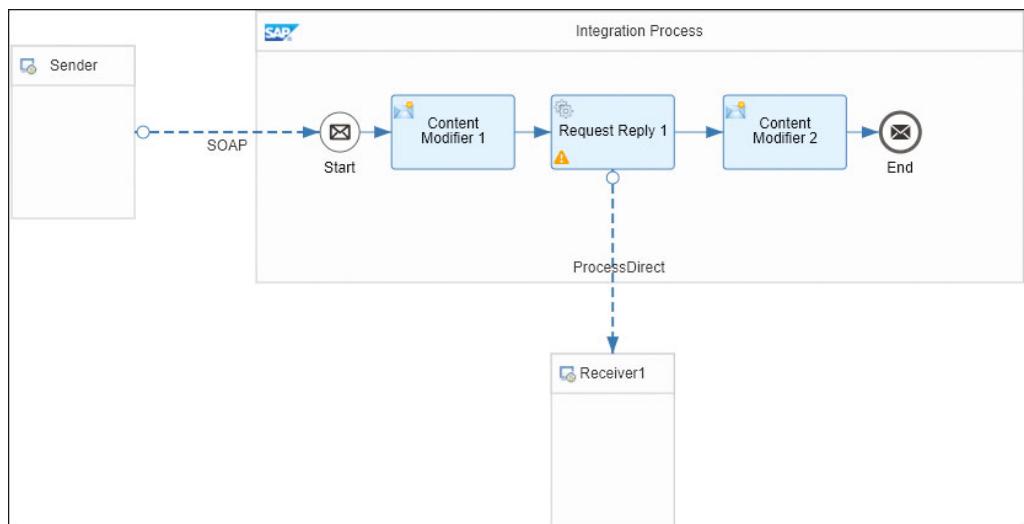


Figure 6.50 Producer Integration Flow

To configure the two **Content Modifier** steps, use the exact settings shown in [Section 6.3](#). However, add a receiver (**Receiver1**) and replace the **Process Call** with a **Request-Reply** step that calls **Receiver1** through the **ProcessDirect** adapter. Due to the **Request-Reply** step, the message that is produced by the second (consumer) integration flow is sent back as a response.

As **Address** in the **ProcessDirect** adapter, use [/pd2router](#) ([Figure 6.51](#)).

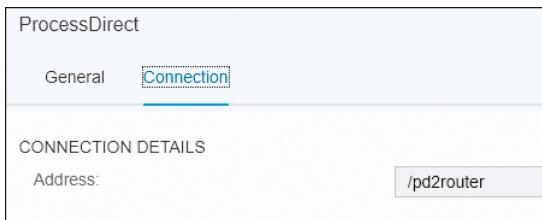


Figure 6.51 Address Configured in the ProcessDirect Adapter of the Producer Integration Flow

The consumer integration flow contains the integration logic that was sourced out into the local integration process in the previous section ([Figure 6.52](#)).

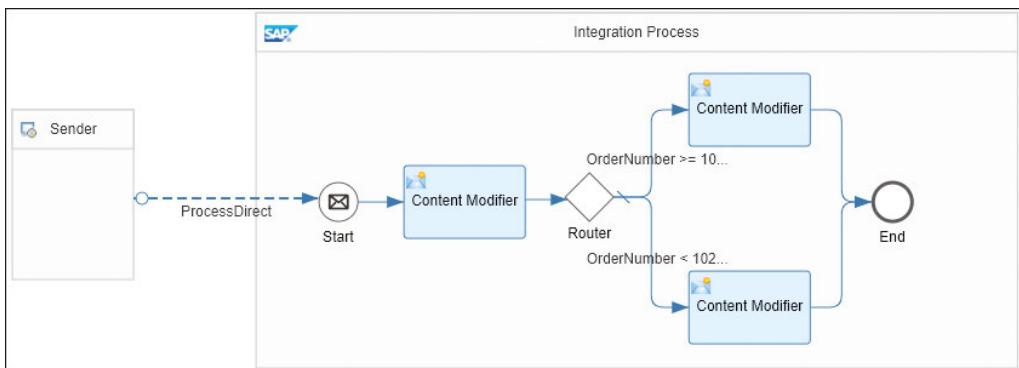


Figure 6.52 Consumer Integration Flow

Compared to the local integration process discussed in [Section 6.3](#), replace the **Start** event with a **Start Message** event, and add a **Sender** pool that connects to the **Start Message** event through the **ProcessDirect** adapter. In the **ProcessDirect** adapter, enter the same address as in the producer integration flow. Deploy both integration flows, and run the scenario by invoking the producer integration flow with SoapUI.

The result should be the same as in [Section 6.3](#) (with the local integration process) with one exception: in the **Monitoring** application of the Web UI, you'll notice that the two integration flows have been processed individually and that, consequently, two MPLs have been generated.

6.4.3 Dynamic Endpoint Configuration with the ProcessDirect Adapter

We'll wrap up the discussion of the ProcessDirect adapter capabilities with a simple example where we combine this feature with dynamic configuration (as explained in [Section 6.2](#)).

To keep it simple, we slightly modified the scenario explained in [Section 6.2](#). As you remember, in this scenario, an external OData resource is accessed by SAP Cloud Platform Integration. Depending on an order number provided with the initial SOAP request, we get different orders related to different values for the `ShipCountry` field. The integration flow further sends out an email with the details of the order. We showed in [Section 6.2](#) that we can dynamically configure the **Subject** attribute of the **Mail** adapter so that, depending on the value of the `ShipCountry` field, another email subject was written into the outbound email (**France** or **Brazil**, depending on whether it was a French or a Brazilian order).

Now we want to modify this scenario slightly: we'll use the same inbound processing (defined in a producer integration flow), but we'll define the outbound processing (defined in consumer integration flows) depending on the shipment country of the looked-up order.

Accordingly, we split this integration scenario in the following way:

- The producer integration flow contains the part of the original integration flow from [Section 6.2](#) (refer to [Figure 6.11](#)) that receives the SOAP request, modifies the inbound message, sends a request to the OData source, and then creates the `Country` header (which contains the value of the `ShipCountry` field of the order).

We dynamically configure the **Address** field of the **ProcessDirect** adapter. In the producer integration flow, we use the expression `${header.Country}` to specify the **Address**, which means that it will depend on the value of the header `Country` of the message as to which consumer integration flow will be called.

- We then create two consumer integration flows (each one for a different country) that do nothing but send the incoming message to the email account with different mail subjects. We define the **ProcessDirect** adapter **Address** field in the consumer integration flow by entering the name of a country. For the consumer

integration flow that handles French orders, we enter “France”, and for the consumer integration flow that handles Brazilian orders, we enter “Brazil”. Consequently, when the header **Country** (in the message coming from the producer integration flow) has the value **France**, the “French” consumer integration flow is called, and when this header has the value **Brazil**, the “Brazilian” consumer integration flow is called.

We could, of course, define different consumer integration flows for all existing countries. To keep it simple, we define only two different consumers and will show how to handle all other countries (besides France and Brazil) at the end of the section.

We don’t go any further into the details here, but you can easily imagine extending such an integration scenario by defining completely different further processing steps, depending on the shipment country. To keep it simple, we only configure different email subjects depending on the country.

Let’s get started.

First, define the producer integration flow shown in [Figure 6.53](#), and use the same settings for the **Content Modifiers** and the **OData** adapter as shown in [Section 6.2](#) (refer to [Figure 6.12](#)).

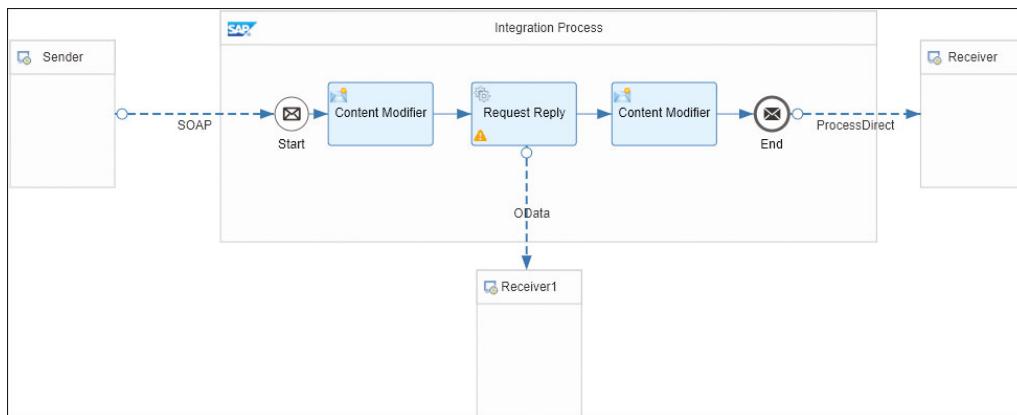


Figure 6.53 Producer Integration Flow

For the **ProcessDirect** adapter, specify the following **Address** attribute: “\${header.Country}” ([Figure 6.54](#)).



Figure 6.54 Dynamically Configured Address in the ProcessDirect Adapter of the Producer Integration Flow

Now design the consumer integration flow (e.g., to handle French orders) shown in [Figure 6.55](#).

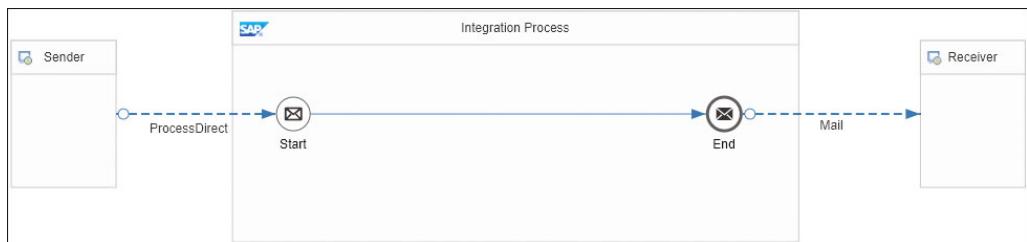


Figure 6.55 Consumer Integration Flow

In the **ProcessDirect** adapter, specify **France** as the **Address** ([Figure 6.56](#)).

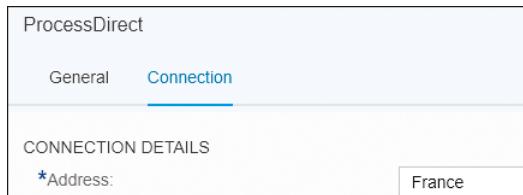


Figure 6.56 ProcessDirect Adapter for the “French” Consumer Integration Flow

In the **mail** adapter, enter the settings you’ve used already in the previous scenarios, except for the **Subject** attribute. For the **Subject**, enter “Bonjour!”.

Design a second consumer integration flow with the following different settings:

- In the **ProcessDirect** adapter, specify the **Address** as **Brazil**.
- For the **Subject** for the **Mail** adapter, enter “Boa tarde!”.

Deploy all integration flows and, first, run the producer integration flow by providing orderNumber 10250 with the SOAP request. You get an email in your inbox that looks like in [Figure 6.57](#).

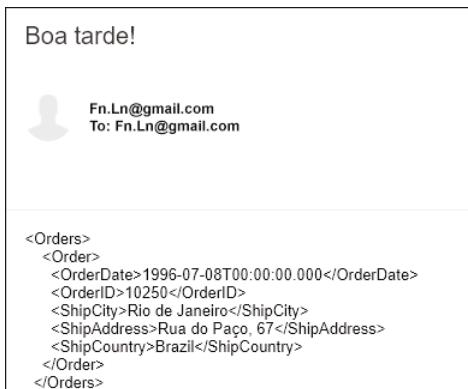


Figure 6.57 Received Email for an Order with Shipment Country Brazil

As expected, you get the details of a Brazilian order and the email subject is **Boa tarde!**. If you run the scenario with orderNumber 10251, you get an email with a French order and subject **Bonjour!**.

The differently configured **Mail** adapter **Subject** attribute should illustrate in a simple way that in the different consumer integration flows, you can define whatever outbound processing for your message that you like. Depending on the country, you might have additional processing steps to apply to the message, or you might want to enrich the outbound message with country-specific data. We skip such additional steps here for the sake of simplicity.

In addition, as mentioned, we've only designed consumer flows for two example countries; therefore, if you provide the SOAP request an orderNumber that doesn't match either of the two countries, the integration flow will run into an error. In a real-world situation, you need to invest in a strategy for how to handle such situations. One option to solve this could be that in the producer integration flow, you define a routing step after the second content modifier that makes sure another receiver is addressed for all orders with shipment countries other than France or Brazil. For example, you can enhance the producer integration as shown in [Figure 6.58](#).

In this integration flow, you only have added a **Router** step and a second email receiver. The upper path contains the default route.

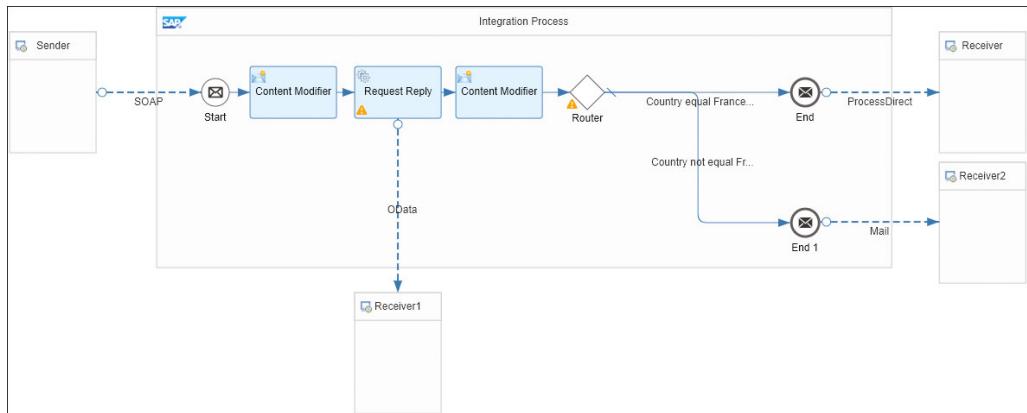


Figure 6.58 Producer Integration Flow Handles All Other Countries by Routing the Message to a Second Receiver

For the lower path, in the **Condition** field, you can enter the following (see [Figure 6.59](#)): “\${header.Country} != 'France' and \${header.Country} != 'Brazil'”.



Figure 6.59 Lower Route Forwarding All Messages Related to Shipment Countries Other Than France or Brazil to Another Receiver

The email receiver configured through the lower **Mail** adapter (connecting to **Receiver2**) will then receive all messages related to orders associated with any other country besides France or Brazil.

When you now run the scenario by providing **orderNumber** 10252 (which is associated with a Belgian order), you'll receive an email in the email account configured in the lower **Mail** adapter (**Receiver2**).

If you provide `orderNumber` 10251, you'll (like above) receive an email with subject **Bonjour!** and the French order in the email account configured in the upper **Mail** adapter (**Receiver**).

6.5 Versioning and Migration of Integration Flows

SAP Cloud Platform Integration is an on-demand software product, for which SAP provides monthly updates. SAP Cloud Platform Integration customers work and run their scenarios on a platform that is enhanced constantly, which makes it a critical requirement that the software lifecycle doesn't impact any productive scenarios. SAP also needs to ensure that integration developers can seamlessly enhance integration content and use new features while minimizing change efforts with regards to existing integration flows.

SAP Cloud Platform Integration offers also a hybrid SAP Cloud Platform Integration content approach in which integration content designed with the Web UI can also be used on the on-premise integration bus of SAP Process Orchestration (assuming that you have a license for that product). This makes high demands on the versioning concept behind the SAP Cloud Platform Integration software. In this section, we provide an overview of the versioning concept and the integration content migration capabilities, including how to migrate integration flow components to a higher version. Finally, we show how to "downgrade" integration flow components, which is required when using integration content on the SAP Process Orchestration integration bus.

6.5.1 Integration Flow Component Versions

With a monthly release of the SAP Cloud Platform Integration tools, integration flow components updated with new features (e.g., new adapter or flow step attributes) are released as components with a new version (also referred to as *component version*). (*Component*, refers in the following either to an adapter or an integration flow step.)

To find out more, open an integration flow. To check out the version of a SOAP adapter (as one example for an integration flow component), click on the **SOAP** channel in the model, and then click on the **Information**  icon ([Figure 6.60](#)).

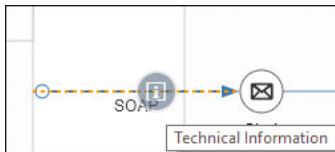


Figure 6.60 Selecting the Information Button for an Adapter

The **Technical Information** screen in this case indicates that the component version is 1.6 ([Figure 6.61](#)).



Figure 6.61 Technical Information for an Adapter

The general nomenclature of a component version is **<major version>.<minor version>**, where the parts of the version indicator have the following meaning:

- **Major version**

The major version is usually not changed to ensure backward compatibility of the component.

- **Minor version**

The minor version is incremented when new features have been added to the component (e.g., when a new adapter attribute has been added).

Playing in the background and not indicated for the user is a third, additional version counter for the micro version, which is related to smaller improvements such as changes of UI labels. We won't elaborate further on this concept.

To summarize, [Figure 6.62](#) shows how SAP Cloud Platform Integration software versions are related to component versions when the software is being updated.

[Figure 6.62](#) illustrates the situation for the monthly software release in April 2018 (to mention an example). Components A, B, and C stand for three different integration flow components that belong to the SAP Cloud Platform Integration design tooling. Imagine that A, B, and C can represent adapter or integration flow step types. In a real-life picture, we would need to sketch many more components. We only show three of them (each one with just a few features) for simplicity.

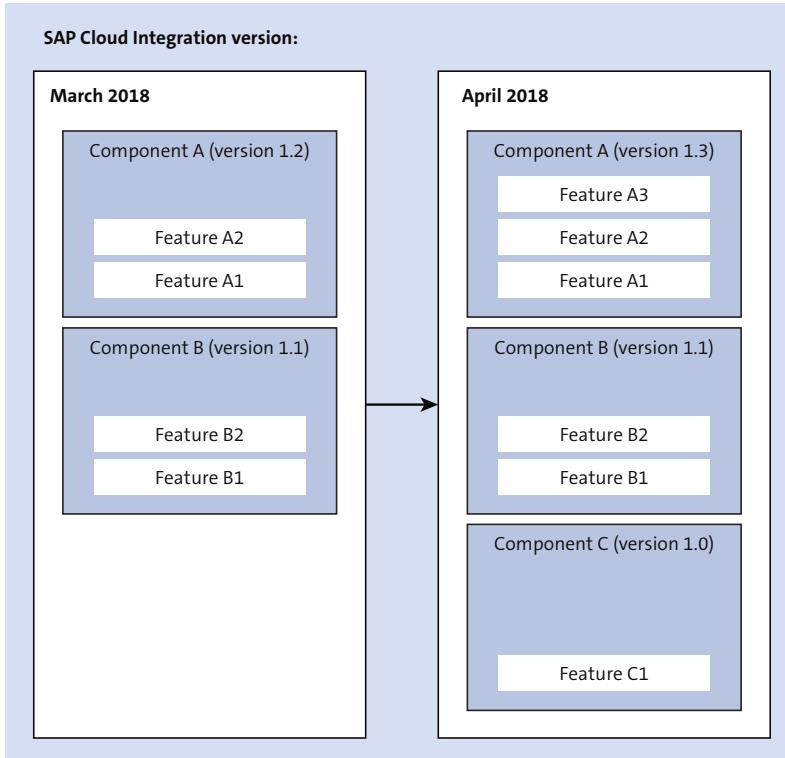


Figure 6.62 How Software Versions and Component Versions Are Related to Each Other

In the shown example, the software version from March 2018 already contains the two components A and B. With software updates to the April 2018 version, component A is enhanced by a new feature (A3), and, accordingly, the component version for A is changed from 1.2 to 1.3. Component B isn't changed during this update, so the feature scope remains the same. Consequently, the component version for B isn't changed, so it stays on 1.1. However, component C is added as a new component during this update. For C, you can imagine that SAP has added, for example, a new adapter type with a software update in April 2018. The new component gets the initial component version 1.0.

You'll understand the implications and importance of this versioning concept as soon as we introduce the concept of migrating an integration flow component. We first explain how *upgrading* an integration flow component works, which means migrating a component to a newer version.

6.5.2 Upgrading an Integration Flow Component

Consider the situation that you've created, deployed, and put into operation an integration flow at a certain point in time, for example, shortly before you left for a longer break or vacation. However, the SAP Cloud Platform Integration development team never rests and provides a software update every month. When you come back from vacation a few weeks later, a new feature for a component used in your integration flow is available. It goes without saying that your concrete integration flow that you deployed weeks ago doesn't reflect this new feature yet. However, reading the release notes of SAP Cloud Platform Integration, you become aware of the new feature and want to update your specific integration flow to also support the new feature in this certain component.

The best option is to migrate the integration flow component. This enables you to update your integration flow component to support the newest features made available by SAP without the need to re-create the component from scratch.

Save the Integration Flow as a Version First

After you've migrated a component to a new version, you can't undo this action to revert back to the old version. Therefore, it's recommended that before migrating a component, you save your integration flow as a version. When you've opened an integration flow, the **Save as version** option can be found next to the **Save** option on top of the integration flow model (see [Chapter 3](#), Figure 3.13). Using this option, you create a copy of the integration flow, and you can move back to the older version of your integration flow if the migration results in any issues.

To migrate a component, for example, an **Integration Process** component, proceed as follows:

1. Open the integration flow, and click **Edit**.
2. Click on the component you want to migrate (in this case, on the **Integration Process** shape).
3. To check the current version of the component, click on the **Information**  icon. The information screen shows that the version is 1.0 ([Figure 6.63](#)).

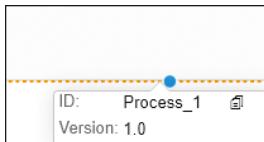


Figure 6.63 Version of the Selected Integration Process Prior to Migration

4. Click **Migrate** (Figure 6.64).

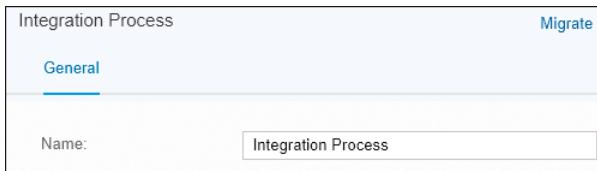


Figure 6.64 Migrating an Integration Process

At the time this book went to press, only a few integration flow components could be migrated (see the information box later in this section).

5. When you've clicked **Migrate**, a confirmation dialog shows the source and target version (Figure 6.65).

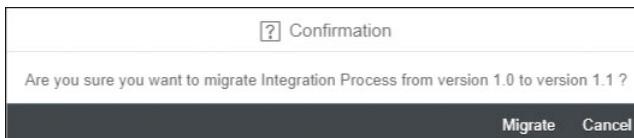


Figure 6.65 Confirmation Message

6. Choose **Migrate** to confirm your selection. The component will be migrated to the latest available version.
7. To check this, again click the **Information** icon to display the technical information. It will now show the new version 1.1 (Figure 6.66).



Figure 6.66 Version of Selected Component Increased to 1.1 after Migration

8. After migration, the new features can be consumed.

Component version 1.1 of the **Integration Process** element has a new feature called **Transaction Management**, as shown in [Figure 6.67](#) (compare with [Figure 6.64](#), which shows the **Integration Process** component prior to migration). For more information about this feature, see [Chapter 5, Section 5.4.2](#).

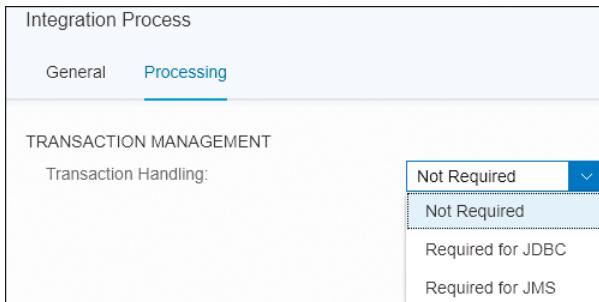


Figure 6.67 New Feature Transaction Management in Upgraded Process

9. Save the integration flow.

If there are incompatibilities with versions of other components within the integration flow, these are indicated in the integration flow model as errors (on explicit save of the integration flow).

Alternatively, you can delete the component and re-create it. This action should also bring up the up-to-date set of features for that component.

When Is Migration Supported

Migrating an integration flow component is only supported when there is a newer version of the component available (and your integration flow contains still an older one). When no migration option is available, no newer version is available for that component (and also no **Migrate** option is shown for the component).

Furthermore, note that migration of integration flow components is only available for editable integration flows. That means, for configure-only content (see [Chapter 3](#)), you can't use this feature.

As of this book's publishing, SAP supports migration of the following components:

- Integration flow
- Integration process
- Local integration process

It's on the SAP Cloud Platform Integration product roadmap that migration of further components, such as adapters or integration flow steps, will be supported in the future. Check out the product documentation regularly to find out more.

We would like to give another example: Migrating an integration flow supports consuming a newer HTTP session-handling feature for HTTP-based receiver adapters. This feature has been made available with the July 2017 release of SAP Cloud Platform Integration.

When you edit an “old” integration flow (created prior to that release date) and click on the area outside the **Integration Process** shape in the integration flow modeler, you find these features in the **Runtime Configuration** tab ([Figure 6.68](#)).

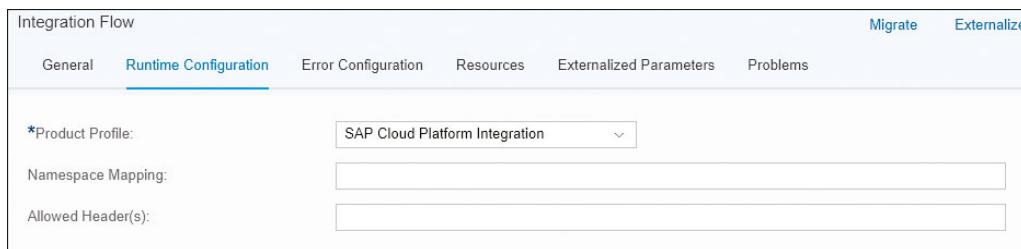


Figure 6.68 Runtime Configuration Features Prior to Migration

You also notice the **Migrate** option on the top-right corner of the screen, indicating that this component (the integration flow) can be migrated.

Now migrate this component as described previously. After migration, you'll notice that the **Runtime Configuration** tab now shows an additional feature, **HTTP Session Handling** ([Figure 6.69](#)).

Note

For more information on the session-handling feature, see the “Cloud Integration – How to Configure Session Handling in Integration Flow” blog in the SAP Community (www.sap.com/community.html).

So far, we've discussed how you can migrate *upward* to a newer version (in case you like to keep pace with the monthly updates of the SAP Cloud Platform Integration software and adapt your integration flow to the newest available features).

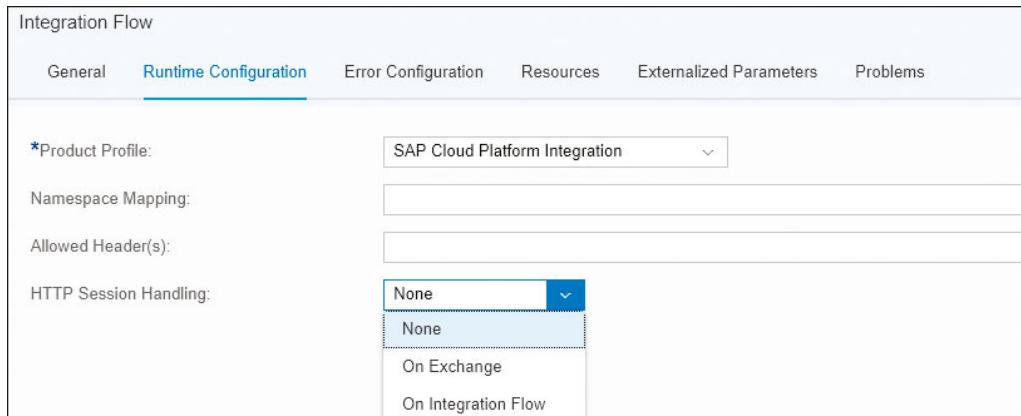


Figure 6.69 Runtime Configuration Features after Migration

To complete the story, we'll show you how SAP Cloud Platform Integration also supports *downward* migration to an older feature set. Check out the next section to learn more.

6.5.3 Downgrading Integration Content for SAP Process Orchestration

Why would it make sense to migrate integration flow components the other way around? Let's briefly set the context.

Throughout this book, we've focused on cases where you deploy and run integration flows on your SAP Cloud Platform Integration tenant. In other words, all integration flows we've developed throughout the chapters of this book have been executed on a runtime node of SAP Cloud Platform. SAP provides updates for both the capabilities of the integration runtime and of the integration design environment (the Web UI) on a monthly basis in the same development cycles. The development of new integration flow features and of the corresponding runtime capabilities is synchronized. That means, for each new capability released for the integration flow designer (e.g., for a new flow step offered in the palette), you can always be sure that this feature is also supported by the actual integration runtime on SAP Cloud Platform.

Product Profiles

If you're solely working with SAP Cloud Platform Integration, that is fine, and you're happy with the migration capability we explained in the previous section. However, SAP also provides another powerful integration solution that has been already been

in place for many years and has been used by many SAP customers: SAP Process Orchestration. For a comprehensive introduction of this product, check out *SAP Process Orchestration: The Comprehensive Guide* by Bilay and Blanco (SAP PRESS, 2017, www.sap-press.com/4431).

As explained already in [Chapter 1](#), many customers use a hybrid system and process landscape where parts of the scenario run in the cloud and other parts are handled by components installed on the premises of the customer. In addition, many customers use both integration platforms in combination: the cloud-based SAP Cloud Platform Integration and SAP Process Orchestration on premise. So why not use the Web UI centrally to design both integration flows that are able to run on SAP Cloud Platform Integration and integration flows that are able to run on SAP Process Orchestration? When you have a license for SAP Cloud Platform Integration *and* for SAP Process Orchestration, you can go for this hybrid approach. To support this use case, SAP has introduced product profiles in the Web UI (we've already briefly touched on this in [Chapter 2, Section 2.2.1, Table 2.3](#)).

What are product profiles good for? To understand this, note that SAP Process Orchestration and SAP Cloud Platform Integration are developed and updated in different cycles (as we'll illustrate in more detail later). However, it's important that the features you can use in the integration content design tool (Web UI) correspond to the capabilities of the target runtime (integration bus) where you intend to deploy the integration flow. Because the release cycles of the integration platforms—SAP Process Orchestration and SAP Cloud Platform Integration—also might differ slightly, SAP has introduced product profiles. Choosing the right product profile makes sure that the integration developer gets exactly those design and modeling features in the Web UI that are also supported by the target integration runtime corresponding to the product profile—and no more. You can choose among SAP Cloud Platform Integration and the recent versions of SAP Process Orchestration product profiles.

Product Profile

A product profile defines the capabilities of the Web UI design environment that are supported for a chosen target integration runtime. If you've only purchased an SAP Cloud Platform Integration license, you only need the product profile **SAP Cloud Platform Integration**. If you've purchased an SAP Process Orchestration license as well, you can get the option to choose between the following product profiles:

- SAP Cloud Platform Integration
- SAP PO <Support Package>

The latest available support packages of SAP Process Orchestration 7.5 are offered. Earlier releases of SAP Process Orchestration (prior to release 7.5) aren't supported: only the runtime components for that latest SAP Process Orchestration release are enhanced with regard to the runtime features of SAP Cloud Platform Integration.

To go into more detail and to understand how product profiles are related to the topic of downgrades, let's look at the different development cycles of both integration runtimes.

Updates for SAP Cloud Platform Integration (both the integration runtime components and the Web UI as the central, cloud-based design tool) are released monthly. On the other hand, updates for SAP Process Orchestration are released in "slower" cycles together with SAP NetWeaver (roughly in quarterly shipments). Therefore, the capabilities of SAP Process Orchestration (as one target integration runtime for your integration flows) lag behind the capabilities of the integration flow design tool (Web UI). When selecting a certain SAP Process Orchestration product profile, the scope of capabilities of the Web UI is adapted so that you can only choose among those features that are also supported by the respective SAP Process Orchestration runtime. Because SAP Process Orchestration is an on-premise solution, and the features of a dedicated release aren't enhanced further, there is always a maximum version of all components for a dedicated SAP Process Orchestration product profile, whereas for the SAP Cloud Platform Integration product profile, the set of features is enhanced constantly each month. Only when a new release of SAP Process Orchestration is made available can you also select a new product profile offering the exact updated set of design features that are then also supported by the new SAP Process Orchestration release ([Figure 6.70](#)).

In the example, in **SAP Process Orchestration 7.5 SP x**, the SAP Process Orchestration runtime can provide maximum support for those integration features that are developed in the Web UI prior to (and including) version n. That means, the Web UI product profile **SAP Process Orchestration 7.5 SP x** has to "filter out" all newer integration capabilities that are developed after version n. For the product profile **SAP Process Orchestration 7.5 SP x+1**, newer features can then be taken into account.

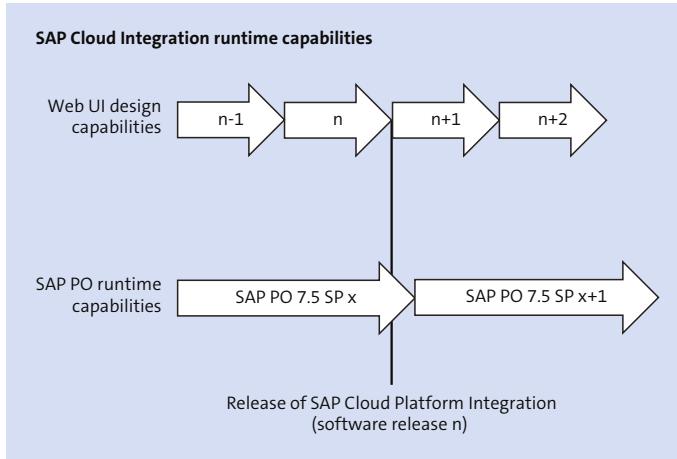


Figure 6.70 Release Cycles of SAP Cloud Platform Integration (with Web UI) Compared to SAP Process Orchestration

Versioning of an individual integration flow component looks like [Figure 6.71](#) (to give an example).

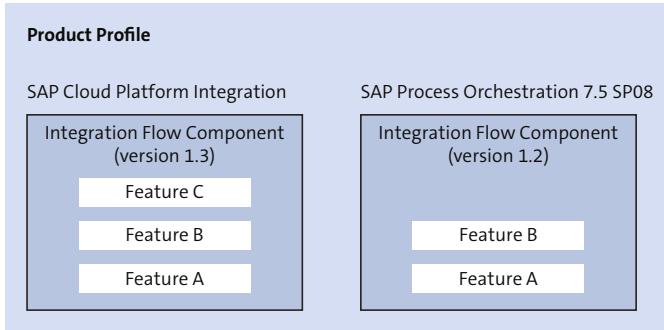


Figure 6.71 Integration Flow Components Showing Different Feature Sets Depending on the Chosen Product Profile

Working with Product Profiles

When you have both an SAP Process Orchestration and an SAP Cloud Platform Integration license, you can choose among different product profiles in the Web UI **Settings** section ([Figure 6.72](#)).

Note that you need to have assigned the authorization group `AuthGroup.Administrator` (tenant administrator) to be authorized to access the tenant settings.

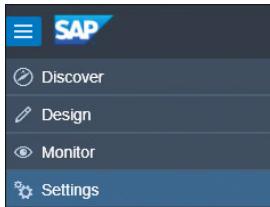


Figure 6.72 Choosing the Tenant Settings Section on the Web UI

Notice that for this tenant, the product profile **SAP Cloud Platform Integration** and those for the recent releases of **SAP Process Orchestration** are available ([Figure 6.73](#)).

The screenshot shows the SAP Cloud Platform Integration tenant settings. The top navigation bar includes the SAP logo, the title 'SAP Cloud Platform Integration', and a 'Data Services' button. On the left is a sidebar with icons for 'Discover', 'Design', 'Monitor', and 'Settings'. The main content area is titled 'Manage Product Profiles'. It lists several product profiles with a 'Default' column:

Name	Default
SAP Cloud Platform Integration	<input checked="" type="radio"/>
SAP Process Orchestration 7.50, SP05	<input type="radio"/>
SAP Process Orchestration 7.50, SP06	<input type="radio"/>
SAP Process Orchestration 7.50, SP07	<input type="radio"/>
SAP Process Orchestration 7.50, SP08	<input type="radio"/>
SAP Process Orchestration 7.50, SP09	<input type="radio"/>
SAP Process Orchestration 7.50, SP10	<input type="radio"/>
SAP Process Orchestration 7.50, SP11	<input type="radio"/>

Figure 6.73 Tenant Settings Showing the Available Product Profiles

If you see only the **SAP Cloud Platform Integration** product profile, but you want to work with the other product profiles also, create a ticket for your SAP Cloud Operations team (that provided you with the tenant). Note that when clicking on one of the product profiles, you'll get a list of component versions supported by the selected product profile.

By editing the settings, you can define a default product profile. When creating a new integration flow, the default setting will be used for this integration flow. You can also set product profiles not only globally for the tenant (as shown for the **Settings**

section) but also on an integration flow level. To see this, open an integration flow, click on the area outside the **Integration Process** shape, and open the **Runtime Configuration** tab ([Figure 6.74](#)).

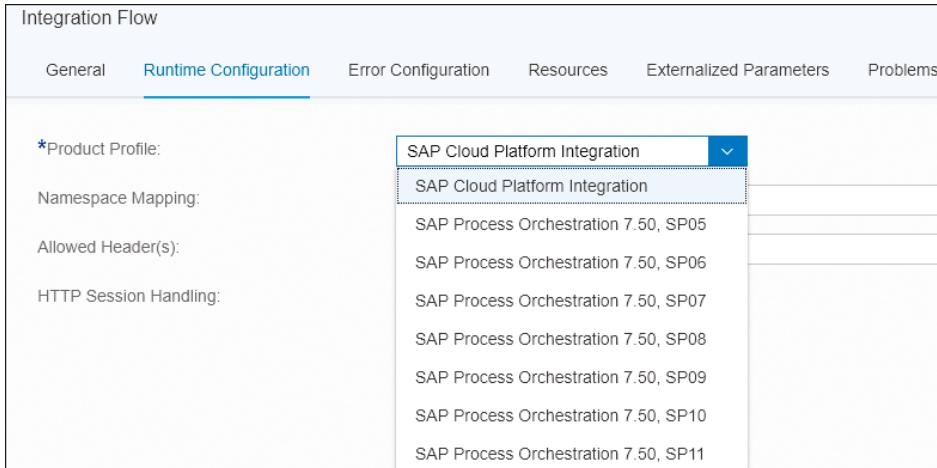


Figure 6.74 Setting a Product Profile for an Integration Flow

Select product profile **SAP Cloud Platform Integration**, and add a new **Receiver** communication channel to your integration flow. Notice that you have a large variation of receiver adapters among which you can select, as shown in [Figure 6.75](#).

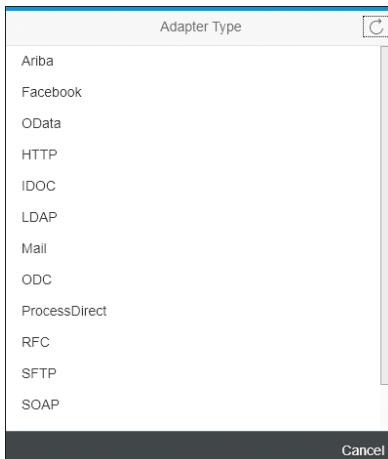


Figure 6.75 Offered Receiver Adapter Types for the SAP Cloud Platform Integration Product Profile

Cancel this activity, go back to the **Runtime Configuration** tab, and choose the product profile **SAP Process Orchestration 7.50, SP09**. Again, add a new **Receiver** channel, and you'll get the following selection of receiver adapters ([Figure 6.76](#)).

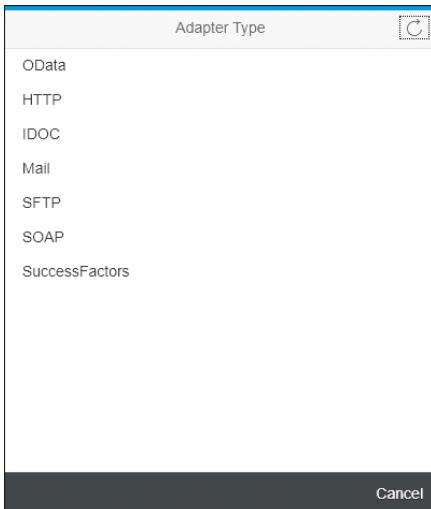


Figure 6.76 Offered Receiver Adapter Types for SAP Process Orchestration 7.50, SP 09 Product Profile

Some adapters are missing, for example, the Facebook and Twitter adapter, because the selected SAP Process Orchestration runtime (for the chosen release 7.50 SP 09) doesn't support the connectivity with these platforms. Future releases of SAP Process Orchestration might support such a connectivity, however. If that is the case, product profiles corresponding to such future SAP Process Orchestration releases available with future releases of the Web UI might offer these additional adapter types.

To show another example, edit an integration flow, specify product profile **SAP Cloud Platform Integration**, and go to the palette. Click the **Security Elements** icon (a shield with a key), and then select **Encryptor**. You'll notice that only the **PKCS7 Encryptor** is available ([Figure 6.77](#)).

If you select product profile **SAP Cloud Platform Integration** and repeat these steps, you notice (as expected) that the **PGP Encryptor** is also available ([Figure 6.78](#)).

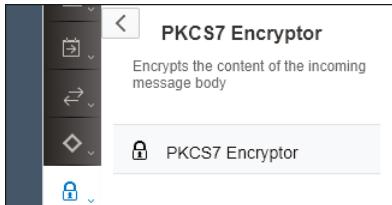


Figure 6.77 Offered Encryption Step for Product Profile SAP Process Orchestration 7.50, SP 09

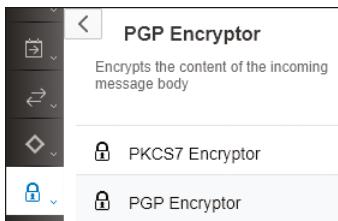


Figure 6.78 Offered Encryption Steps for Product Profile SAP Cloud Platform Integration

Web UI capabilities might also differ on the detail level of each individual integration flow component so that certain features of an adapter or flow step are missing when an SAP Process Orchestration product profile is selected (as shown earlier in [Figure 6.71](#)). But we won't go into more detail here.

Migrating within an SAP Process Orchestration Product Profile

To conclude this section, note that migration of integration flow components, as described in [Section 6.5.2](#), is also supported within an SAP Process Orchestration product profile. However, upgrading is then only possible to the maximum component version supported for the chosen SAP Process Orchestration product profile. If you want to use features of the component that are released later, you need to check out if a newer SAP Process Orchestration product profile (and, correspondingly, a newer SAP Process Orchestration release) is available.

For more information on the topic of migration, check out the "Versioning & Migration of Components of an Integration Flow in SAP Cloud Platform Integration's Web Application" blog in the SAP Community (www.sap.com/community.html). If you want to learn more about using integration content together with SAP Process

Orchestration, check out the “Best Practices Cloud Integration Content in SAP Process Orchestration – Overview” blog.

6.6 Transporting Integration Packages to Another Tenant

In this section, we provide an overview of the options that are available to transport integration content across tenants. Here, a typical use case is that you first design and run your integration scenarios in a development landscape and, after development is finished, you transfer your scenarios into a test landscape.

SAP provides the following options to transport integration content:

- Manual transport
- Using enhanced Change and Transport System (CTS+)
- Using the Transport Management Service

We describe these options in the following sections.

6.6.1 Manually Transporting Integration Packages

To manually transport integration packages, you don't need to do any prerequisite steps to set up this scenario. In the Web UI **Settings** in the **Transport** tab, you can keep the default **Transport Mode** setting of **None**.

In your source tenant, you then have to open the Web UI, go to the **Design** section, and open the integration package that you want to transport. You can use the **Export** option in the upper-right corner of the screen ([Figure 6.79](#)) to store the integration package as a **.zip** file on your computer.

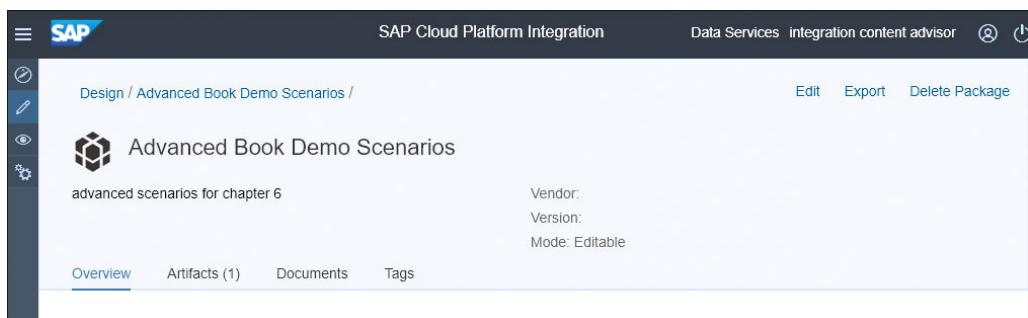
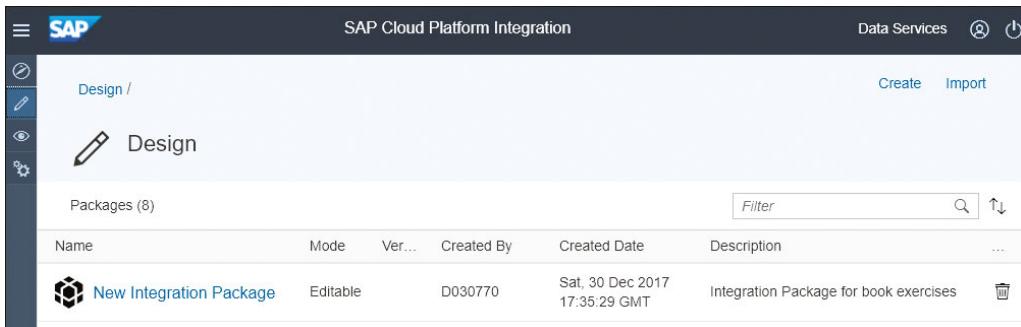


Figure 6.79 Export Function for an Integration Package (on Source Tenant)

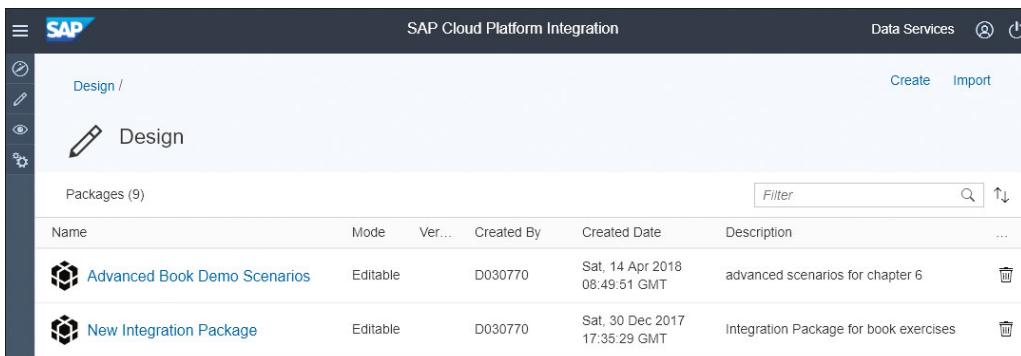
Subsequently, you open the Web UI for your target tenant, go to the **Design** section, and click **Import** ([Figure 6.80](#)).



The screenshot shows the SAP Cloud Platform Integration interface. In the top navigation bar, the title "SAP Cloud Platform Integration" is displayed. On the right side of the header, there are links for "Data Services" and a power icon. Below the header, the left sidebar has a "Design /" path and several icons: a magnifying glass, a pencil, a gear, and a refresh symbol. The main content area is titled "Design". At the top right of this area are "Create" and "Import" buttons. Below this, a table lists "Packages (8)". The columns are "Name", "Mode", "Ver...", "Created By", "Created Date", and "Description". A "Filter" input field and a search icon are at the top of the table. One row is selected, showing "New Integration Package" as the name, "Editable" as the mode, "D030770" as the created by, "Sat, 30 Dec 2017 17:35:29 GMT" as the created date, and "Integration Package for book exercises" as the description. A trash can icon is to the right of the last column.

Figure 6.80 Import Function in Web UI Design Section (of Target Tenant)

You can now browse for and double-click the `.zip` file on your computer. As a consequence, the integration package is added to the target tenant ([Figure 6.81](#)).



This screenshot is identical to Figure 6.80, showing the SAP Cloud Platform Integration interface in the Design section. The table "Packages (9)" now includes the newly imported package. The second row from the top is selected, showing "Advanced Book Demo Scenarios" as the name, "Editable" as the mode, "D030770" as the created by, "Sat, 14 Apr 2018 08:49:51 GMT" as the created date, and "advanced scenarios for chapter 6" as the description. A trash can icon is to the right of the last column. The other packages listed are "New Integration Package" (selected) and the previous ones.

Figure 6.81 The Imported Integration Package Added to the Target Tenant

This is quite straightforward and may be the best way to go when transporting content as an occasional task. However, you might want to look for a comprehensive framework for change and transport management when productively working with many integration packages and when you have to perform transports in a more coordinated way. Check out the next sections for the options.

6.6.2 Transporting Integration Packages Using CTS+

This option might be interesting for you if you already use SAP's enhanced Change and Transport System (CTS+). CTS+ is SAP's enhanced on-premise transport management system that comes with SAP NetWeaver. Note that using CTS+ as transport management system requires a system landscape where your source tenant is connected to a CTS+ system through SAP Cloud Platform Connectivity service.

In this book, we won't elaborate further on this option. Instead, we refer you to a blog series in SAP Community that provides a detailed step-by-step description of how to set up this transport scenario and how to use it. At www.sap.com/community.html, search for "Content Transport Using CTS - Cloud Integration – Part 1" and "Content Transport Using CTS - Cloud Integration – Part 2."

6.6.3 Transporting Integration Packages Using the Cloud-Based Transport Management Service

You can transport integration content across tenants with a few clicks through the cloud-based Transport Management Service. At the time this book is written, this feature is in beta phase.

To use this feature, you need to contact SAP first. For more information, check out the "Transport Integration Content across Tenants Using the Transport Management Service Released in Beta" blog in SAP Community (at www.sap.com/community.html), which provides good help to get started with the topic of adapter development.

For the setup, we assume that you'll use the Transport Management Service to transport integration content from a development tenant to a test tenant.

Figure 6.82 shows the landscape for our transport scenario.

Figure 6.82 indicates that SAP Cloud Platform Integration tenants run in the SAP Cloud Platform Neo environment, whereas the Transport Management Service runs in the Cloud Foundry environment. This fact is important for the setup of the transport scenario as described in detail in a separate blog (see further below). For more information on the different SAP Cloud Platform environments, go to the online documentation at <https://help.sap.com/viewer/p/CP>, and search for "Environments".

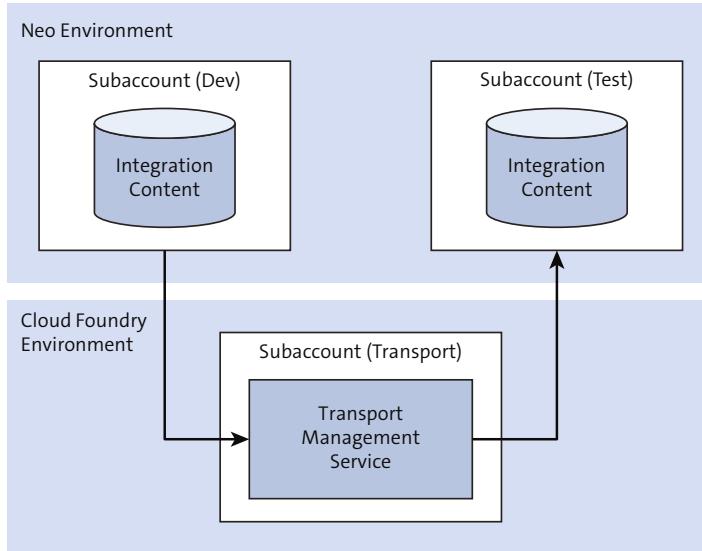


Figure 6.82 Transport Scenario

To enable the Transport Management Service and configure a transport landscape, the following steps are required:

1. Get access to a Transport Management Service account.
2. In the Cloud Foundry environment, create a subaccount of your global account, and subscribe to Transport Management Service.
3. Enable API access to Transport Management Service.
4. Create the required destinations.
 - In the Neo dev subaccount: Create two destinations to a Cloud Foundry subaccount (Transport Management Service).
 - In the Neo dev subaccount: Create one destination used by the Solution Lifecycle Management Service in the Neo dev account to point to the SAP Cloud Platform Integration dev tenant (same Neo subaccount).
 - In the Cloud Foundry subaccount (Transport Management Service): Create one destination to point to each target subaccount (Neo) of your transport landscape. In our example (transport from Neo dev tenant to Neo test tenant), create one destination that points to the test subaccount.
5. In Transport Management Service, create source and target nodes and a transport route to connect both.

All these tasks are described in detail and step by step in the “Cloud Integration – Using Transport Management Service (Beta) for a Simple Transport Landscape” SAP Community blog.

In the following parts of this section, we focus on the content transport itself and describe how to transport content from a dev to a test tenant.

In the mentioned blog, exactly the same transport landscape as depicted in [Figure 6.82](#) is used, so that you easily can tie up the following description to the steps described in the blog.

As we assume that all these steps have been performed successfully, you can now start transporting content. As the first task, go to your source tenant (dev), and specify the transport mode. Then, you can transport an integration package to the target (test) tenant.

Configuring the Transport Mode in the SAP Cloud Platform Integration Web UI

In the Cloud Integration Web UI, you need to specify the desired option how to transport integration content (also referred to as transport mode). As described in the beginning of [Section 6.6](#), there are different options such like using CTS+ or the Transport Management Service (beta).

1. Open the Web UI for the dev tenant, choose **Settings**, and select the **Transport** tile.
2. Click **Edit**.
3. As **Transport Mode**, select **Transport Management Service (Beta)** ([Figure 6.83](#)).

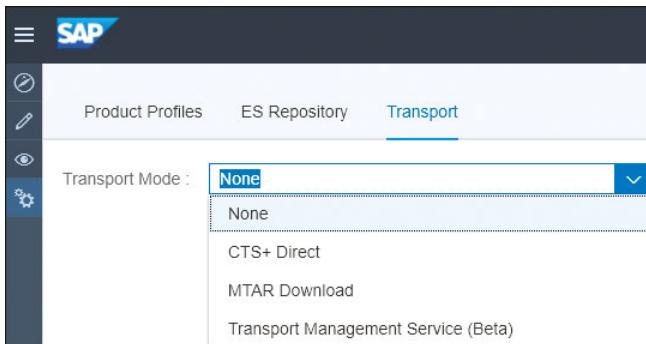


Figure 6.83 Transport Modes in the Web UI Settings Section

4. Click **Save**.

For the sake of completeness, [Table 6.1](#) summarizes the supported transport modes.

Transport Mode	Description
None	Manually export an integration package from the source tenant to your computer, and import it to the target tenant from there.
CTS+ Direct	<p>Transport an integration package directly (with one click) from a source tenant (e.g., a dev tenant) to a target tenant (e.g., a test tenant) through CTS+. <i>Directly</i> means that the integration package will be attached to your open transport request in CTS+.</p> <p>With this option, the integration content to transport is transferred directly as a Multi-Target Application (MTA) Archives file (MTAR file with extension <i>.mtar</i>) to an open transport request in the configured CTS+ system.</p> <p>The MTA defines a file format to package a heterogeneous set of software pieces that can be created with different technologies but that all share a common lifecycle.</p>
MTAR Download	Download an MTAR file from the tenant you want to export integration content from, and manually upload the MTAR file to a CTS+ system (or to Transport Management Service, depending on your setup).
Transport Management Service (Beta)	Transport integration content across tenants with a few clicks through the cloud-based Transport Management Service of SAP Cloud Platform. This feature currently is in beta phase.

Table 6.1 Transport Modes Available for Integration Content Development

You can now start transporting integration content from the dev to the test tenant.

Transporting Integration Content Using the Transport Management Service

You have now prepared everything to be ready to transport integration content. The following steps show how to transport integration content across two tenants:

1. Open the Web UI for the dev tenant (from where you want to transport the integration content).
2. Select the integration package you want to transport (let's assume it's the package **Advanced Demo Scenarios**).
3. Choose **Transport**.

This option is only shown when you've enabled Transport Management Service as shown earlier.

- Enter a transport comment, and choose **Transport** (Figure 6.84).

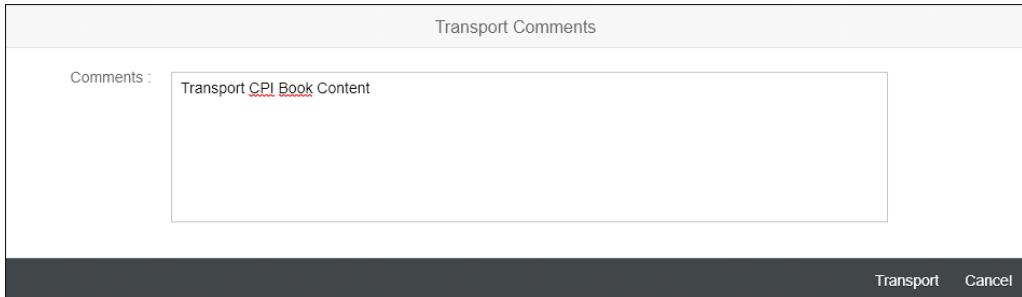


Figure 6.84 Transport Comments Dialog in Web UI

You get an **Information** screen stating that the transport request has been created (Figure 6.85).

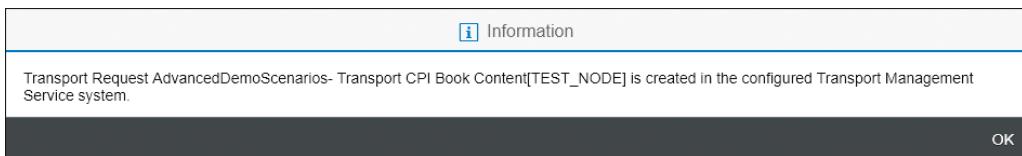
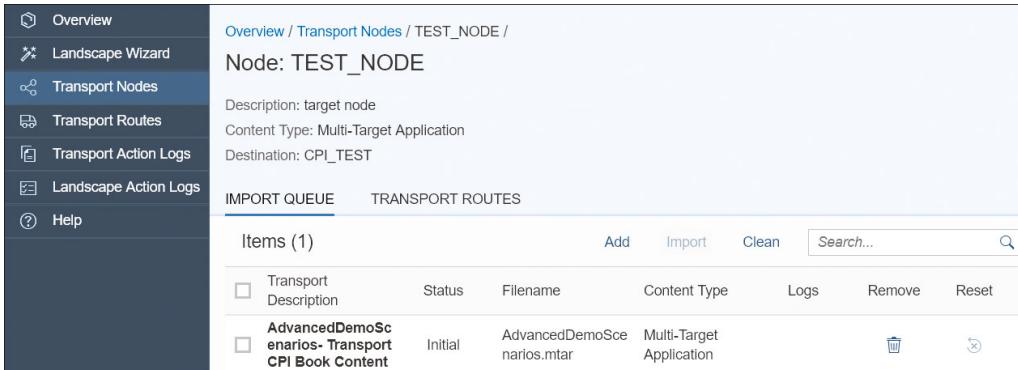


Figure 6.85 Transport Information Screen

Notice the name of the target transport node (`TEST_NODE`) configured in Transport Management Service for the target tenant (which is the test tenant) is part of the transport landscape defined in the Transport Management Service. In the blog mentioned earlier, we show how to create this node and how it's related to the other required configuration settings in all involved accounts.

- Check the transport import queue by opening the Transport Management Service and choosing **Transport Nodes** (see the blog referred to earlier to find more details on how to access the Transport Management Service).
- Select the transport node for your target account (`TEST_NODE` in our example). You should find the transport in the **IMPORT QUEUE** tab (Figure 6.86).
- Select the queue, and choose **Import** (Figure 6.87).



The screenshot shows the SAP Transport Node interface. On the left is a sidebar with navigation links: Overview, Landscape Wizard, Transport Nodes (selected), Transport Routes, Transport Action Logs, Landscape Action Logs, and Help. The main area displays the path: Overview / Transport Nodes / TEST_NODE / Node: TEST_NODE. Below this, it shows the node's description as 'target node', content type as 'Multi-Target Application', and destination as 'CPI_TEST'. A table titled 'IMPORT QUEUE' lists one item: 'AdvancedDemoScenarios- Transport CPI Book Content' (Status: Initial, Content Type: Multi-Target Application). There are buttons for Add, Import, Clean, and a search bar.

Figure 6.86 Import Queue for Selected Transport Node

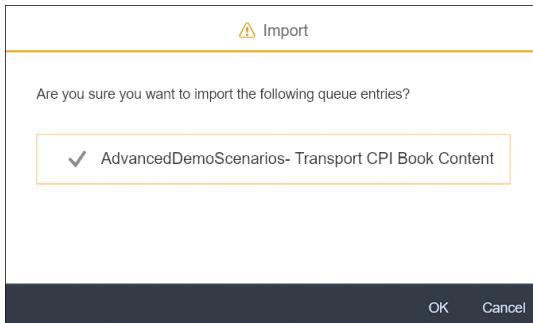


Figure 6.87 Confirmation of Import

8. On the confirmation screen, choose **OK**.
9. Go to the test tenant (the target tenant of your transport), and check whether the package arrived there ([Figure 6.88](#)).



The screenshot shows the SAP Design interface. It has a 'Design' icon and the word 'Design'. Below it, under 'Packages (13)', are two entries: 'Advanced Demo Scenarios' (Mode: Editable) and 'Book Demo Content' (Mode: Editable, Version: 1.0.1).

Figure 6.88 Target Tenant Now Showing the Imported Package

The transported package (**Advanced Demo Scenarios**) has been imported. That's it! You've successfully used the Transport Management Service to transport an integration package from a source to a target tenant.

6.7 Using the Adapter Development Kit

SAP Cloud Platform Integration is able to connect to a multitude of systems (in the cloud and on premise). It already comes with an impressive number of adapters that allow communication with other systems on different levels, considering different technologies, security standards, and application-specific requirements. Typical examples are adapters connecting to Twitter, SAP Ariba, and SAP SuccessFactors applications, or adapters connecting via protocols, such as HTTP, SOAP, IDoc, OData, or Secure Shell File Transfer Protocol (SFTP). However, SAP will never be able to cover the huge variety and combinations of applications, protocols, security standards, and versions solely on its own. This gives customers and partners the unique opportunity to fill those gaps by providing their own adapters. Note that SAP Partners have already developed various adapters and published them in the Integration Content Catalog. You can find this information by choosing the **Discover** section of the Web UI and filtering for "adapter."

In this section, we provide you a brief introduction into the Adapter Development Kit (ADK), which enables you to develop your own adapters for SAP Cloud Platform Integration.

6.7.1 The Adapter Development Kit (ADK)

In previous chapters, we frequently referred to the Apache Camel engine as the central integration framework working under the hood of SAP Cloud Platform Integration. As such, when it comes to the development of adapters for SAP Cloud Platform Integration, it essentially means developing adapters for Camel. The official term in Apache Camel's nomenclature for adapters is *components*. As Camel is an open-source framework, many articles have been published explaining how to implement components to increase Camel's connectivity options. See Camel's official documentation web page for component development at <http://camel.apache.org/writing-components.html>, where you can find more information about a component's implementation. It's also recommended that you take a closer look at the book *Camel in Action* (Manning Publications, 2010). Finally, to get to know the Camel components

that are already available, we recommend checking out <http://camel.apache.org/components.html>. You'll be surprised at how many Camel components simplify your life! In any case, there's no need to dive into those details here again.

However, to make those components enterprise-ready, provide a configuration UI for the adapters in the SAP Cloud Platform Integration Web UI, and provide the integration of components within the SAP Cloud Platform Integration monitoring infrastructure, there is more to be done than merely writing a simple Camel component. This is why SAP released the ADK for SAP Cloud Platform Integration. Here's the trick: after you've developed a component following the Apache Camel guidelines, as described in the various resources mentioned previously, the ADK allows you to wrap that component with additional code to make it compliant with the SAP Cloud Platform Integration infrastructure.

In this section, we show how you can develop a new adapter. For that purpose, we show you how to work with the ADK and how you can add a new sample adapter to your portfolio of adapters. For more detailed guidelines for adapter development, we also refer you to a number of blogs.

6.7.2 Installing the Adapter Development Kit

The development process for an adapter differs significantly from modeling and running integration flows, as we've discussed so far in this book. It can't be done using a web-based graphical modeling tool. It requires a full-blown development environment, such as Eclipse. That's why the ADK comes as an Eclipse installation.

Note that to successfully execute the steps described in this section, you need to make sure that you've installed Apache Maven and that you point to a Java Development Kit (JDK) in your Eclipse rather than a Java Runtime Environment (JRE) installed on your computer. For more details on these steps, refer to the documentation of SAP Cloud Platform Integration at https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud, and search for "Adapter Development Process."

The current ADK was tested with Eclipse Oxygen (equivalent to Eclipse version 4.7). Download the respective release for your operating system from the Eclipse web page at www.eclipse.org/oxygen/. Once downloaded, extract the ZIP file into a folder of your choice. The folder will contain another folder named *Eclipse*. The *eclipse.exe* file inside the *Eclipse* folder can be started by double-clicking on it. You'll be asked for an appropriate folder where Eclipse can store the project files during development.

This special area is called an Eclipse workspace. To separate several Eclipse workspaces, it's recommended to have a dedicated folder for your new installation. We've chosen the *C:\EclipseWorkspaces\Oxygen* folder for this purpose.

Click on **OK** to continue. After the Eclipse environment is up and running, we can install the ADK by selecting **Install New Software** from Eclipse's **Help** main menu (Figure 6.89).

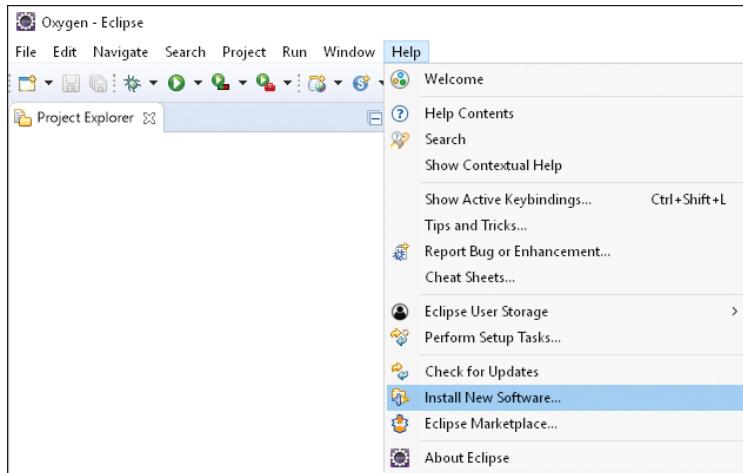


Figure 6.89 Starting the Wizard for Installing Additional Software

The installation wizard opens. The wizard asks you to provide the location on the Internet from where the new software can be downloaded. Click on the **Add** button next to the **Work with** dropdown list (Figure 6.90).

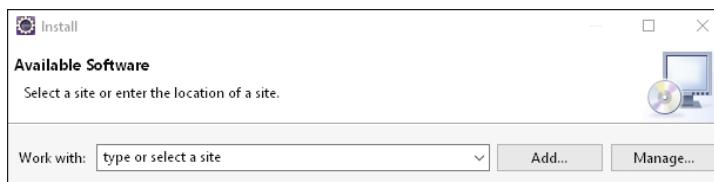


Figure 6.90 Assigning the Download Location for New Software

The **Add Repository** dialog opens (Figure 6.91).

Give your download location an appropriate name (e.g., “SAP Development Tools for Eclipse Oxygen”), and assign it to the location URL <https://tools.hana.ondemand.com/oxygen>. Click on **OK** to proceed.

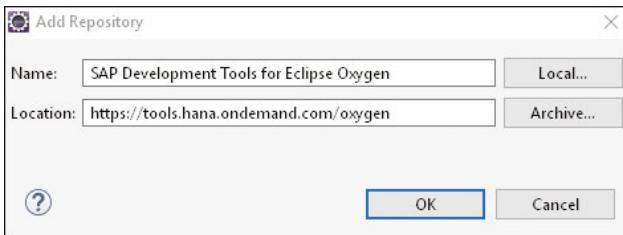


Figure 6.91 Adding the URL for Downloading the SAP Cloud Platform Integration Tools

On the next screen, a list of available software components appears. Select the **Cloud Platform Integration Tools** package. It's recommended to select the complete package, as shown in Figure 6.92.



Figure 6.92 Selecting the SAP Cloud Platform Integration Tools for Installation

Click **Next**, confirm this selection, and all forthcoming steps without modifications. At the end of the wizard, accept the license agreement, and then click on **Finish**. The tools will be installed, and a restart of Eclipse is necessary to complete the installation. After restarting, we have to connect the Eclipse environment with your SAP Cloud Platform Integration tenant, on which we'll deploy our newly developed adapter. Open the **Preferences** dialog by selecting **Window • Preferences** from the Eclipse main menu. The **Preferences** dialog opens (Figure 6.93).

Expand the **SAP Cloud Platform Integration** node on the left of the **Preferences** dialog, select the **Operations Server** node beneath, and enter the **URL** to your tenant. This URL has been sent to you by your SAP Cloud Operations team, including username and password. Add the credentials in the respective fields of the **Preferences** dialog, as well. Test the connection by clicking the **Test Connection** button to verify the validity of your entries. Once done, close the dialog by clicking the **Apply and Close** button. We're now ready to develop the adapter.

There are some additional prerequisites you have to meet with regard to your Eclipse installation. Make sure that you've implemented Maven (download from <https://maven.apache.org/download.cgi>).

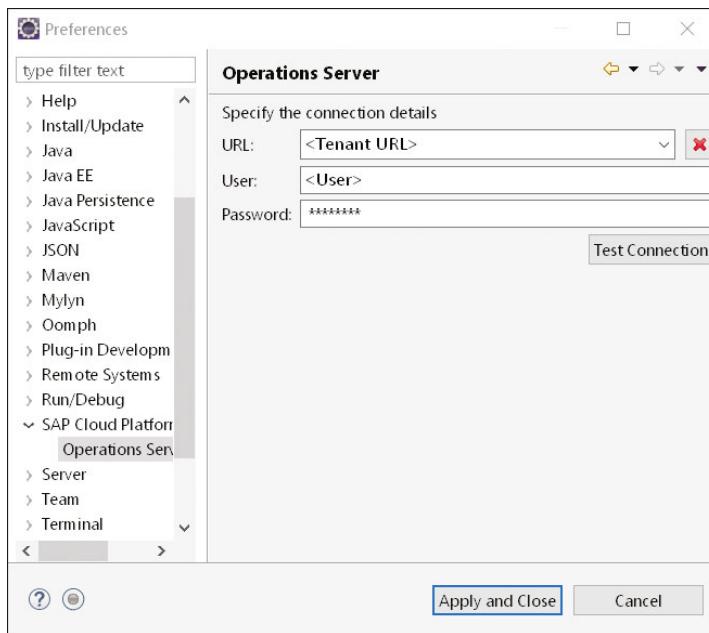


Figure 6.93 Preferences Dialog for Specifying the Connection to the SAP Cloud Platform Integration Tenant

Furthermore, make sure that your Eclipse installation uses and points to a JDK instead of a JRE. You can specify the location of your JDK in Eclipse under **Windows • Preferences • Java • Installed JREs**.

In the next section, we show how you can start developing a sample adapter with just a few clicks.

6.7.3 Developing a Sample Adapter

With the following steps, you can develop and deploy a simple sample adapter project that has been predefined by the SAP Cloud Platform Integration team.

People familiar with software development, and in particular, with Apache Camel and OSGi, can easily enhance such a project to define their own custom adapter. In this section, we won't go into the details of adapter development. Instead, at the end of the section, we refer to the online documentation and a number of blogs that will help you get into the topic of adapter development. In this section, however, we'll show you how to start developing an adapter project by simply using the predefined

sample adapter and how you can immediately use the sample adapter in an integration flow.

The sample adapter built in to SAP Cloud Platform Integration does nothing more than either reading a greetings message from a sender or sending such a message to a receiver. We focus on the sender part and leave it to you to enhance the project as required.

To start creating an adapter project, we first need to choose the appropriate perspective in Eclipse. Check if the Java EE (default) perspective is already opened. If not, perform the following steps:

1. In Eclipse, choose **Window • Perspective • Open Perspective • Other** (Figure 6.94).

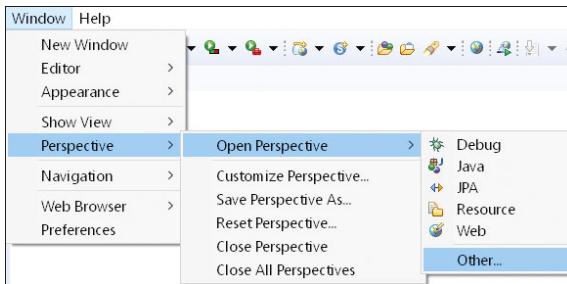


Figure 6.94 Opening a New Eclipse Perspective

2. Select **Java EE (default)** (Figure 6.95).

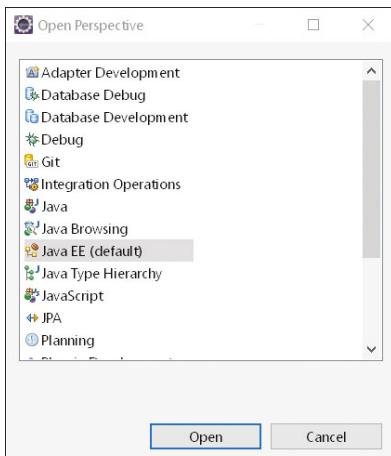


Figure 6.95 Selecting the Integration Designer Perspective

3. Choose **Open**.

We'll now create an adapter project. Here, we define the basic data for the new adapter type (for example, its name). Based on our entries, the system generates a Java project structure which contains all components required to implement the adapters' runtime and its configuration user interface. Perform the following steps:

1. Choose **File • New • Other** ([Figure 6.96](#)).

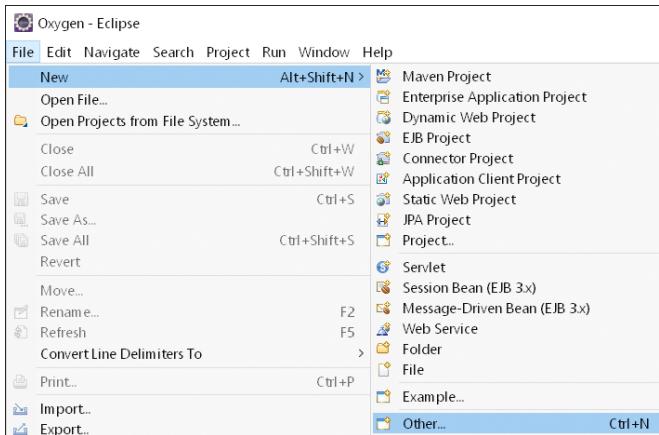


Figure 6.96 Creating a New Project in Eclipse

2. On the next screen, expand the **SAP Cloud Platform Integration** node, and select **Adapter Project** ([Figure 6.97](#)).

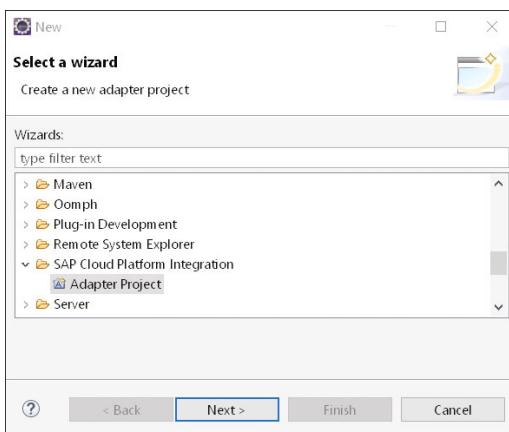


Figure 6.97 Selecting a Wizard for an Adapter Project

3. Choose **Next**.

4. Enter meaningful basic data for the project ([Figure 6.98](#)). The next screenshot shows some proposals (note that the sample adapter does nothing but send a greetings message, unless we further change it).

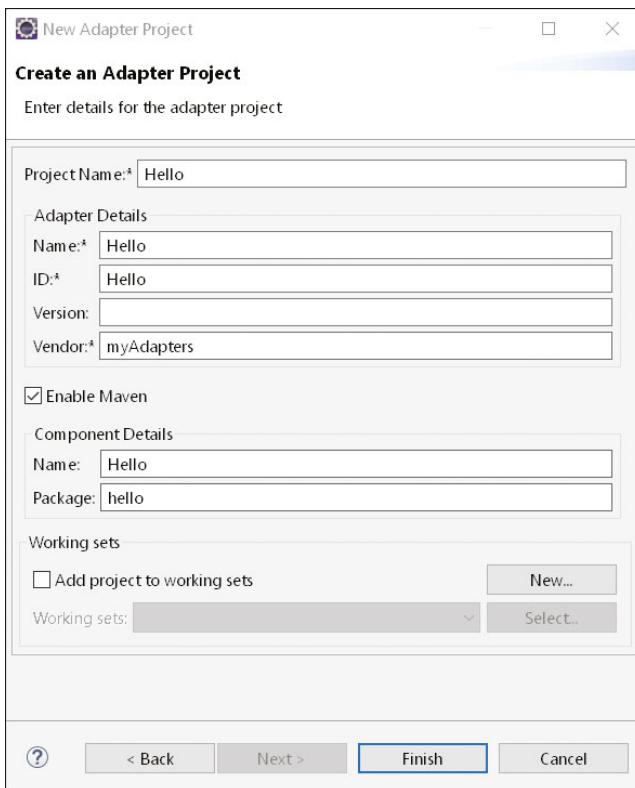


Figure 6.98 Project Details

5. Keep the **Enable Maven** checkbox selected, which is the default setting.

Maven Support for Adapter Development

Adapter development for SAP Cloud Platform Integration also integrates Maven support. Maven is a build management tool from the Apache software foundation. Selecting the **Enable Maven** checkbox makes sure that development of a Camel component is integrated into your adapter project and that dependencies are automatically resolved. This option is selected by default. If you keep this setting, you can

create a sample adapter with a few clicks and modify it later. Maven support also has the advantage that all components required for an adapter can be maintained at one place: the runtime components that control how the adapter processes messages at runtime, as well as the user interface (UI)-related components that determine the adapter UI in the integration flow.

If you want to reuse an existing Camel component (as you can find, for example, under <http://camel.apache.org/components.html>), we recommend deselecting **Enable Maven**. In that case, however, you need to maintain all dependencies of your software packages manually, which can sometimes imply tedious effort.

6. Click **Finish**. The project is added to the **Project Explorer** view (Figure 6.99).

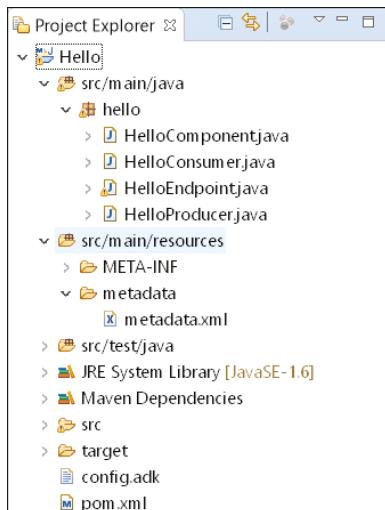


Figure 6.99 New Adapter Project Added to the Project Explorer View

In this figure we've expanded some of its nodes and briefly explain what they are for in the following list:

- *HelloConsumer.java*: Implements the sender adapter, which polls messages according to a certain schedule. It generates a greetings message with the current timestamp.
- *HelloProducer.java*: Implements the receiver adapter.
- *HelloEndpoint.java*: Implements the endpoint and contains the string variable `greetingsMessage` for the message to be polled or sent when keeping the sample

adapter as it is. Furthermore, this component provides a logger to log data during the processing of the adapter.

- *metadata.xml*: Contains the metadata for the adapter UI.
7. You can open the file *metadata.xml* by double-clicking on it.

Figure 6.100 shows an excerpt of the file. Note the selection of the **Source** tab at the bottom of the screen. The configurable attributes of the adapter are assembled in attribute groups. You'll find identically named XML tags throughout the metadata file.

```
<Tab id="connection">
    <GuiLabels guid="b4c970da-a1f8-443c-b063-046773f93135">
        <Label language="EN">Connection</Label>
        <Label language="DE">Connection</Label>
    </GuiLabels>
    <AttributeGroup id="defaultUriParameter">
        <Name xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <GuiLabels guid="041a3129-aedb-4e5b-a351-dc8d82ae7fb8">
                <Label language="EN">URI Setting</Label>
                <Label language="DE">URI Setting</Label>
            </GuiLabels>
            <AttributeReference>
                <ReferenceName>firstUriPart</ReferenceName>
                <description>Configure First URI Part</description>
            </AttributeReference>
        </AttributeGroup>
        <AttributeGroup id="Message">
            <Name xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">
                <GuiLabels guid="eaa0dc7f-e350-44ae-85bf-6a7a46853302">
                    <Label language="EN">Message Details</Label>
                    <Label language="DE">Message Details</Label>
                </GuiLabels>
                <AttributeReference>
                    <ReferenceName>greetingsMessage</ReferenceName>
                    <description>Configure Greetings Message</description>
                </AttributeReference>
            </AttributeGroup>
            <AttributeGroup id="ScheduledPollConsumer">
                <Name xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema">
                    <GuiLabels guid="813cbd00-ea82-47a9-8302-bf16aa6727b9">
                        <Label language="EN">Scheduled Poll Consumer</Label>
                        <Label language="DE">Scheduled Poll Consumer</Label>
                    </GuiLabels>
                </AttributeGroup>
```

Figure 6.100 Excerpt from the Metadata.xml File

We leave everything as it is.

As the next step, we build the adapter project by following these steps:

1. Right-click the adapter project **Hello**, and choose **Run As • 3 Maven build** in the context menu (Figure 6.101).

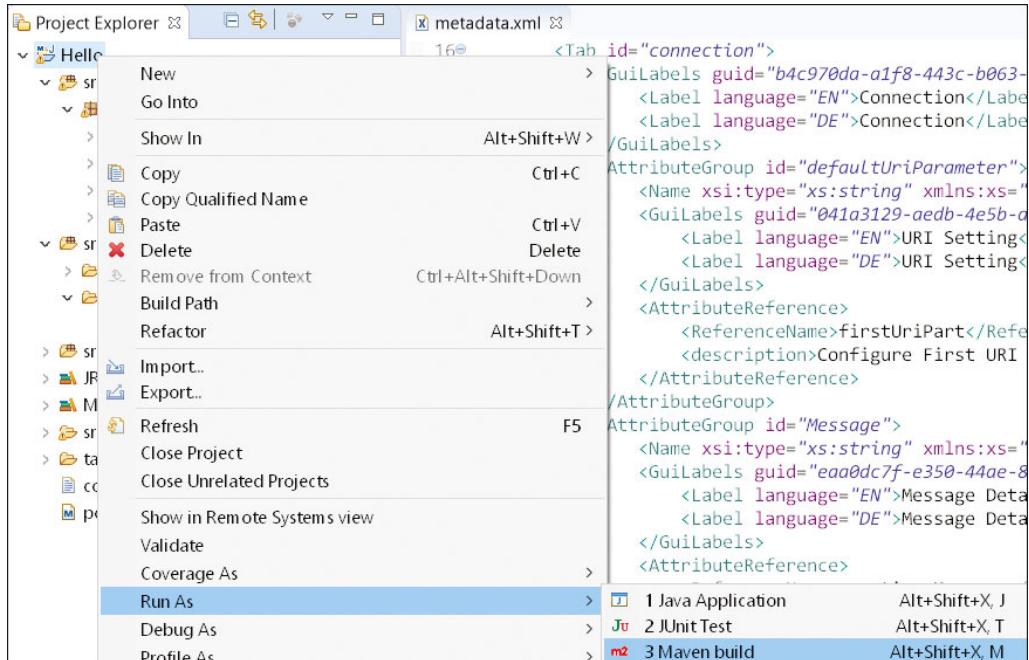


Figure 6.101 Build Adapter Project

After a successful build of the project, you should find additional artifacts in the project tree under **target**.

2. After the success message appears, select again the root node of the project, and choose **Deploy Adapter Project** in the context menu (see [Figure 6.102](#)).
 3. The **Deploy Adapter Project** dialog box appears ([Figure 6.103](#)) where the tenant is displayed (it should be the tenant to which you're connected with Eclipse; if there are doubts, check the settings under **Window • Preferences • SAP Cloud Platform Integration • Operations Server**).

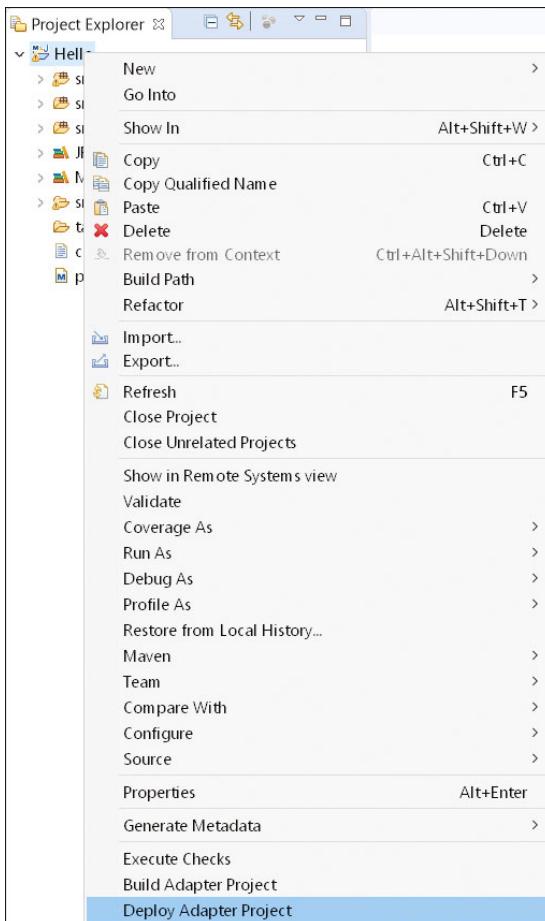


Figure 6.102 Deploy Adapter Project

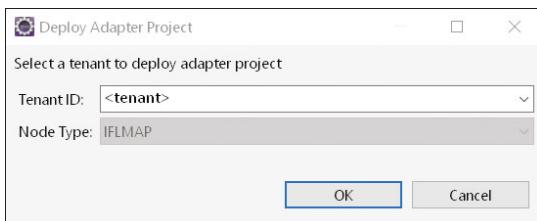


Figure 6.103 Confirming the Tenant to Which the Adapter Project Will Be Deployed

4. Choose **OK**.

Now you can check for the deploy state of your project by following these steps:

1. Open the **Integration Operations** perspective in Eclipse. Choose **Window • Perspective • Open Perspective • Other**, and select the **Integration Operations** perspective (Figure 6.104).

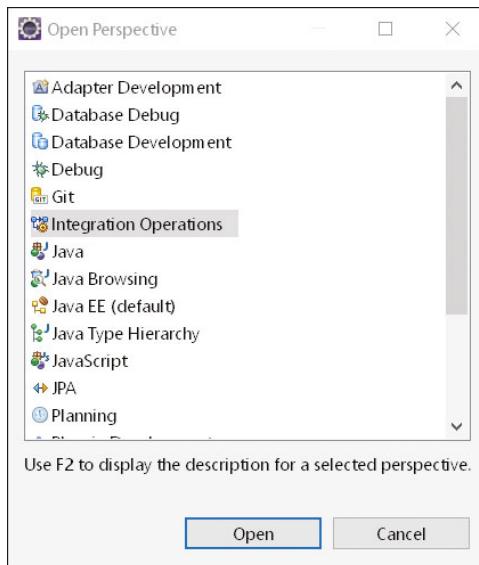


Figure 6.104 Selecting the Integration Operations Perspective in Eclipse

2. Double-click the tenant in the **Node Explorer** view, and select the **Deployed Artifacts** editor (Figure 6.105).



Figure 6.105 Deployed Artifacts Editor for the Tenant Selected in the Node Explorer

Notice that the adapter project is shown as a deployed artifact (with the project name, in our example: **Hello**). When everything goes correctly, the **Deploy State** is **DEPLOYED**.

Now we're ready to test the sample adapter. To keep it simple, we propose using an integration flow with an email receiver (as configured several times within the course of this book). Now let's test the new **Hello** sender adapter by following these steps:

1. Open the Web UI for the tenant, and go to the **Design** section.
2. Either create a new integration flow or reuse an existing one. We propose that you start with this integration flow model ([Figure 6.106](#)).

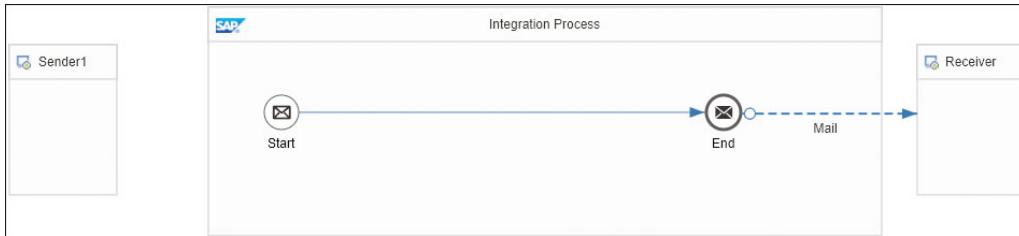


Figure 6.106 Simple Integration Flow to Start With

3. Add a new **Sender** channel between the **Sender** pool and the message **Start** event. You'll notice the new adapter type **Hello** in the **Adapter Type** list ([Figure 6.107](#)).

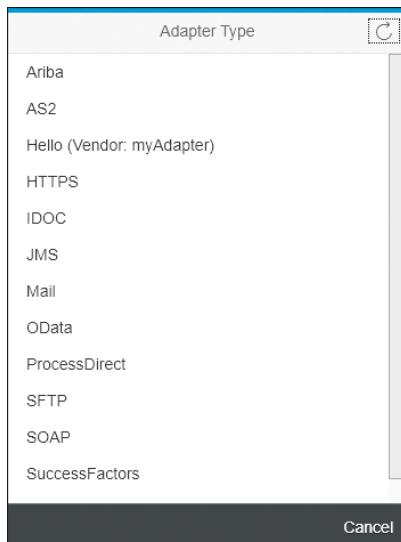


Figure 6.107 New Adapter Offered When Creating a Channel

Also notice that the vendor specified in the project is indicated to show that this adapter is a custom adapter that isn't part of the SAP Cloud Platform Integration portfolio provided by SAP (refer to [Figure 6.98](#)).

4. Select the **Hello** adapter type, and go to the **Connection** section of the adapter ([Figure 6.108](#)).

Use these settings. Notice that the adapter sends hello messages to your email account. Choose some larger intervals for the scheduling options not to spam your email account.

The screenshot shows the 'Hello' adapter configuration page. The 'Connection' tab is selected. The settings are organized into several sections:

- URI SETTING**: First URI Part: [empty input field]
- MESSAGE DETAILS**: Greetings Message: Hello from the Book Demo!
- SCHEDULED POLL CONSUMER**: Backoff Multiplier: [empty input field]
 - Send Empty Message When Idle
 - Initial Delay: 1
 - Backoff Idle Threshold: [empty input field]
 - Use Fixed Delay
 - Delay: 120
 - Start Scheduler
 - Run Logging Level: OFF
 - Backoff Error Threshold: [empty input field]
 - Greedy
- Time Unit:** SECONDS

Figure 6.108 Example Settings for the Sender Hello Adapter

Predefined Settings of the Sample Adapter

The sample adapter implements the Camel polling consumer integration pattern. Similar to the mail sender adapter (described in [Chapter 2, Section 2.3.8](#)), SAP Cloud

Platform Integration polls (reads) a message from a sender before actually processing it. The settings predefined for the sample adapter are given by the scheduler component of Apache Camel. For example, with a combination of the settings **Back-off Multiplier** and **Backoff Error Threshold**, you can control how the adapter should behave if there are erroneous polls. With the attribute **Backoff Error Threshold**, you can specify the number of subsequent erroneous polls before the attribute **Backoff Multiplier** becomes effective. With the latter one, you can then specify the number of skipped polling attempts before polling occurs again. For example, if you enter the value “2” for **Backoff Error Threshold** and the value “3” for **Backoff Multiplier**, then after two erroneous polling attempts, the adapter waits for another three attempts before becoming active again.

For more information about all predefined attributes, check out the documentation of the Camel scheduler component at <http://camel.apache.org/scheduler.html>.

5. Deploy the integration flow.
6. Check the email account (specified in the mail receiver adapter).

The email will contain a message similar to the following (with the actual timestamp when the message has been processed):

Hello from the Book Demo! Now it is Tue May 01 14:15:19 UTC 2018

According to the settings in [Figure 6.108](#), the adapter will send an email every two minutes. Finally, don’t forget to undeploy the adapter after you do this test. Otherwise, SAP Cloud Platform Integration will continue to send messages to your email account.

We won’t go any further into the details. If you’re interested in developing your own adapters with more meaningful features than those shown for the sample adapter, refer to the following blogs in the SAP Community (www.sap.com/community.html), which provide help getting started with the topic of adapter development:

- “SAP CPI Adapter Development – Consuming an External JAR into an Adapter”
- “Extension of Runtime Capabilities Using Blueprint Metadata in Cloud Platform Integration SDK”

You should also check the online documentation of SAP Cloud Platform Integration at regular intervals to find out news about adapter development. Go to https://help.sap.com/viewer/product/CLOUD_INTEGRATION/Cloud, and search for the topic “Adapter Development Process.”

For more information on available Camel components, refer to the documentation at <http://camel.apache.org>.

Now that the previous chapter and this chapter have given you a good and detailed overview of integration development, let's round up this chapter with a summary of integration design best practices.

6.8 Best Practices for Integration Flow Development

Each integration flow processes data that is transferred from sender systems to the SAP Cloud Platform Integration platform. These practices will make sure that your integration flows are optimized with regard to performance of the scenario and memory consumption:

- Limit the size of inbound messages. Certain sender adapters—SOAP (SOAP 1.x), SOAP (SAP Reliable Messaging), and IDoc—allow you to restrict the incoming message size (**Maximum Message Size** parameter). You can restrict the size of the body and of the attachments independently. The smallest value for a size limit is 1 MB in that case. The default setting for these adapters are maximum body size = 40 MB and maximum attachments size = 100 MB.
- Don't store payloads in the MPL. This recommendation holds in particular for productive integration flows. Using a script step, you can store the message payload in the MPL, which is a convenient measure to analyze the message body in the monitoring. However, doing this can cause issues with memory consumption and even with the failure of the integration scenario because memory and CPU available on your tenant are shared by tasks such as message processing (which should never fail) and message monitoring. Therefore, use this option with care. If, nevertheless, you need to write the message payload into the MPL, you can do so in error cases only.

Use integration flow tracing by setting the **Trace** log level instead of this.

For more information, read the “How to Avoid Excessive Storage Load Caused by Using MPL Attachments for Message Logging” blog in SAP Community (www.sap.com/community.html).

Note

Use the **Trace** log level only in test environments. Intentionally, the **Trace** log level is only active for 10 minutes.

- Consider certain best practices when using JavaScript and Groovy Script (with the script step). There are certain risks with regard to memory consumption and overall performance of your scenario if you don't use this option with care. The following blogs in SAP Community can help you avoid some of these risks: "Avoid Binding Variables in Groovy Scripts" and "Stream the XMLSlurper Input in Groovy Scripts."
- When using the JMS adapter (as explained in [Chapter 5, Section 5.4](#)), keep in mind certain resource limits for JMS queues. To find out the actual limitations, check out the online documentation or read the "Cloud Integration – JMS Resource and Size Limits" blog in SAP Community (www.sap.com/community.html).
- Configure transaction handling in an appropriate way. There are different adapter and integration flow step types that store data (either in the database or in a JMS queue). If there is an erroneous processing of the integration flow, any actions that relate to persistence (storing or deleting data) must be rolled back in a consistent way (so that no data inconsistencies are generated). Transaction handling is the way to go here. In [Chapter 5, Section 5.4](#), we discussed transaction handling in the context of the JMS adapter.

Recommendations and limitations with regard to transaction handling are explained in detail (also together with example scenarios) in the "How to Configure Transaction Handling in Integration Flow" blog in the SAP Community. However, note the following recommendation with regard to an optimized resource consumption: Use as short as possible transactions by "sourcing out" transactional processing to subprocesses. As every transaction consumes resources in a considerable amount, make sure that you avoid keeping a transaction open until the whole integration process is finished.

- If you enable streaming (if this is supported), a document is processed in parts or segments. Use streaming whenever this is applicable. For example, the splitter step allows you to enable the **Streaming** option. If you deactivate this option, in the splitter case, the message is transferred fully to the memory before it's split and processed further. Activating **Streaming**, on the other hand, can help you to avoid such a behavior. Another example for a step where streaming is supported is the XML-to-JSON Converter.
- Use HTTP session handling. Various HTTP-based adapters (e.g., the SOAP and IDoc adapters, but many more) support HTTP session handling (to be configured under **Runtime Configuration** after clicking outside the **Integration Process** shape; refer

to [Figure 6.69](#)). One advantage of session handling is that the client is only authenticated once with the first call, which will result in better performance. However, there are also certain restrictions that you should consider when using session handling. For more information about HTTP session handling and its restrictions, as well as to find example scenarios, check out the following SAP Community blog: “Cloud Integration – How to Configure Session Handling in Integration Flow.”

- Avoid writing large message parts or even complete messages into headers or properties. Writing much data into headers can cause issues when sending out the message because headers are transferred along with the message to the receiver. Writing much data into properties might cause memory issues.
- When modifying a message, use properties instead of headers because properties aren't passed along with the message by the receiver adapter.
- Delete headers and properties explicitly (in particular, the large ones) if not required anymore. This helps to save memory. You can use the content modifier to do that.
- Avoid memory-intensive steps such as mapping or accessing elements with XPath expressions when processing large messages. Such operations will blow up the message in memory.

As a tip, check out the *SAP Cloud Platform Integration Community* in regular intervals at www.sap.com/germany/community/topic/cloud-integration.html.

6.9 Summary

We've reached the end of our journey through the more development-oriented topics of the book. In this chapter, you learned how to trigger integration processes on time-based criteria, how to modularize complex real-life scenarios using subprocesses, and, finally, how to extend SAP Cloud Platform Integration's capabilities by developing your own adapters, allowing you to connect to a wide variety of external systems and technologies. Now your task is to play around with SAP Cloud Platform Integration's modeling environment to further stabilize your knowledge. You can then continue with the next chapter, which deals with a more specific topic: business-to-business integration and how SAP Cloud Platform Integration can help you master use cases related to this. Thereafter, you'll learn more about typical operational tasks you'll need to take care of to run SAP Cloud Platform Integration and the deployed artifacts.

