Agenda

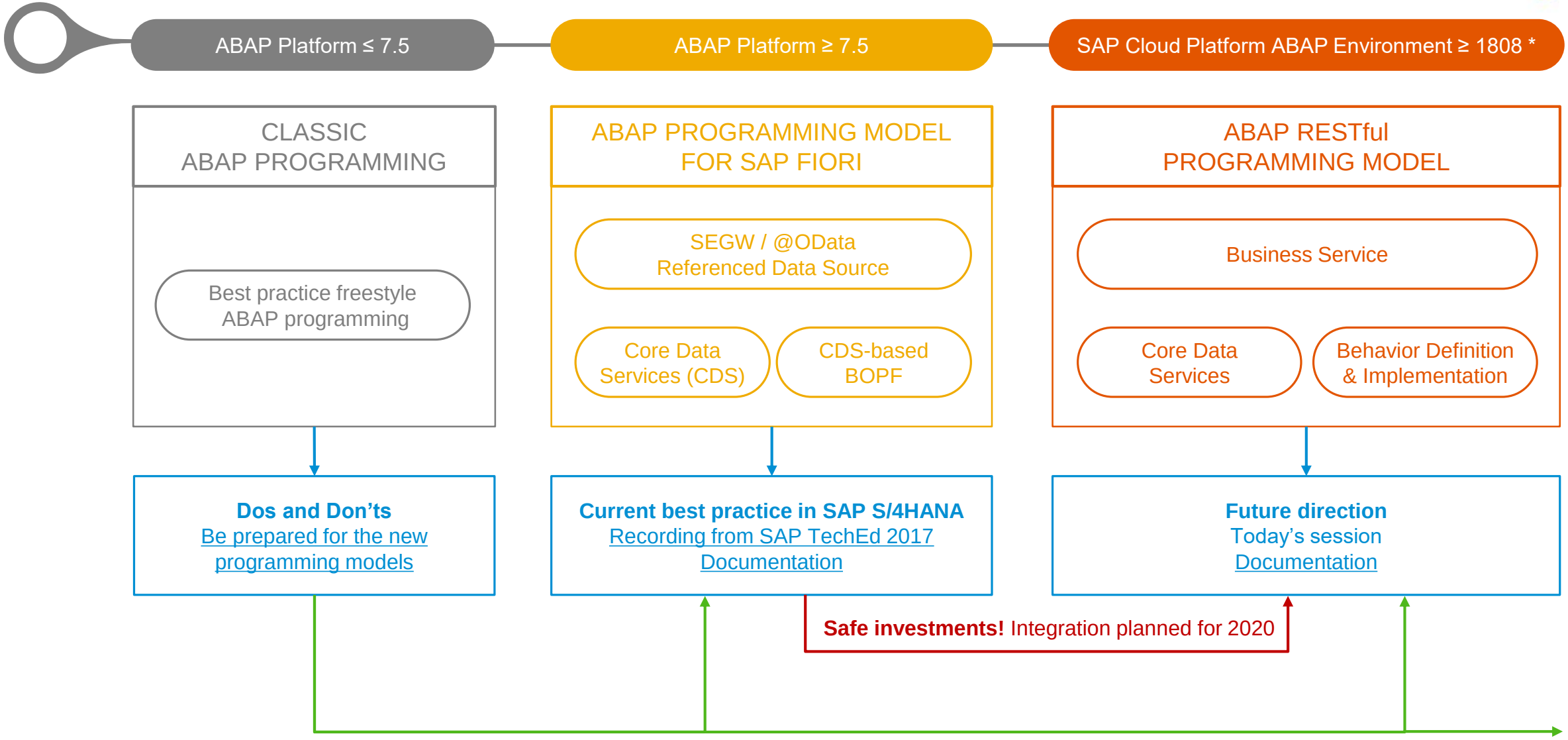
Day 1
Evolution and Big Picture of ABAP RESTful Programming Model
Business Object, Business Object Implementation types, Business Services, Unmanaged Implementation Exercise
Day 2 & Day 3
Entity Manipulation Language
Unmanaged Implementation Exercise – Continuation
Managed Implementation Exercise

Introduction

ABAP RESTful Programming Model

Evolution of the ABAP programming model

SAP Development Learning



* Reduced scope available on-premise as of SAP S/4HANA 1909

ABAP RESTful Programming Model – The key players

SAP Development Learning

ABAP Development Tools in Eclipse for all development tasks

Easy developer onboarding
End-to-end development flow

Language: ABAP and CDS

Standard implementation tasks via typed APIs supporting
static code checks, auto-completion, element info

Frameworks

Take over technical implementation tasks
Business logic added in code exits on protocol agnostic layers

ABAP RESTful Programming Model – The big picture

SAP Development Learning

SERVICE CONSUMPTION

SAP Fiori UI
Consume OData UI services

Web API
Consume OData Web APIs

BUSINESS SERVICES PROVISIONING

SERVICE BINDING - Bind to protocol version and scenario

SERVICE DEFINITION - Define scope to be exposed

BUSINESS OBJECT PROJECTION
- CDS: Projection views
- BDEF: Behavior projection

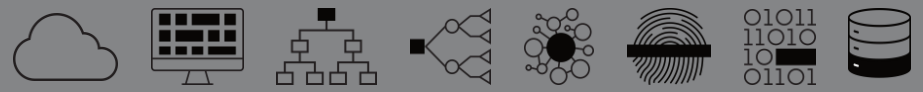
DATA MODELING & BEHAVIOR

BUSINESS OBJECTS
- CDS: Data modeling
- BDEF: Behavior definition
- ABAP: Behavior implementation

QUERIES
- CDS: Data modeling

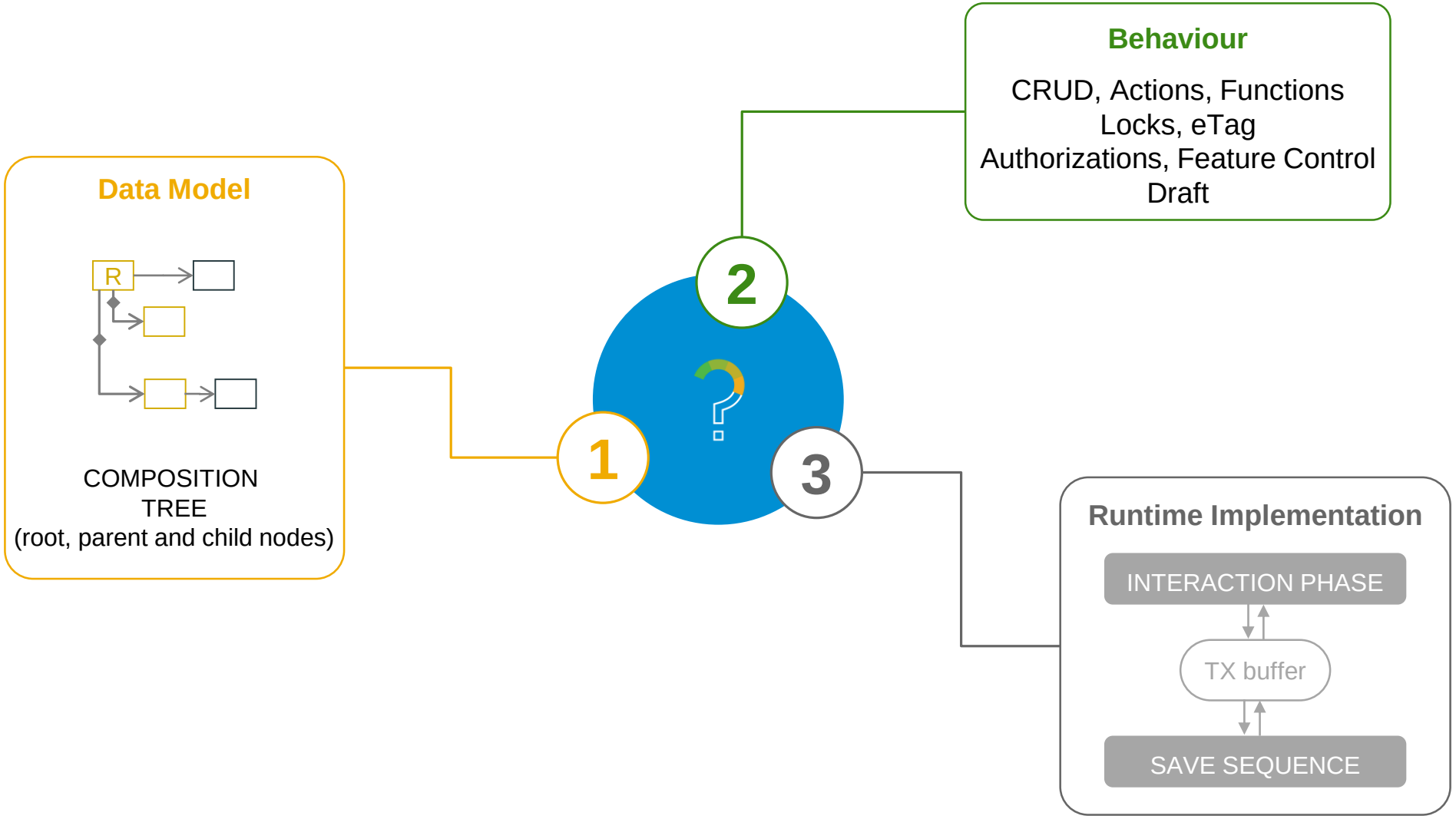
Business Objects (BOs)

ABAP

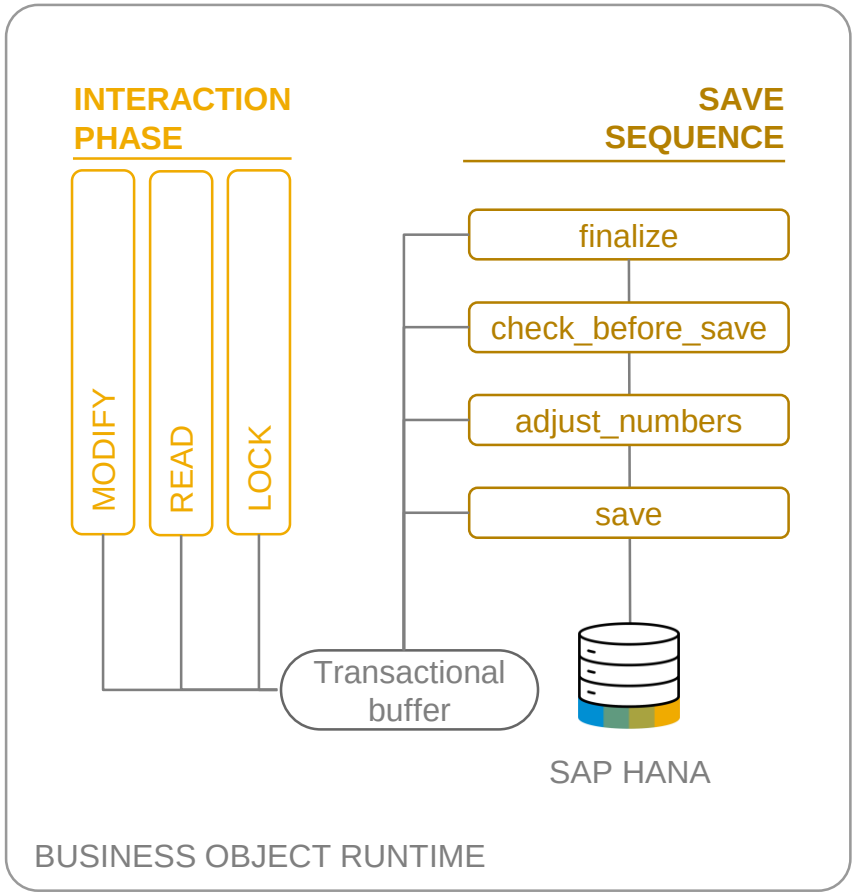


What is a business object?

SAP Development Learning



Business objects – Implementation types



UNMANAGED

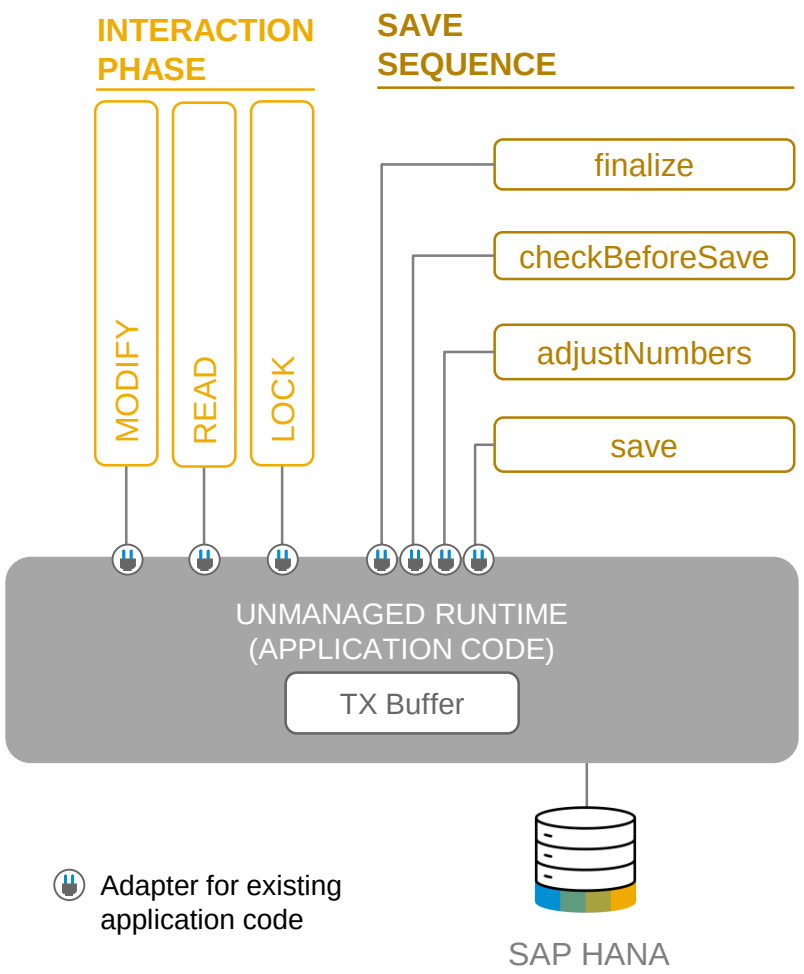
Brownfield development with application coding fully available:
Interaction phase + Transactional buffer + Save sequence

MANAGED*

Greenfield development with standard implementation
(opt. unmanaged appl. components: DB tables, lock/PFCG object, update task FM)

* Currently only available in SAP CP ABAP Environment

Business objects – Unmanaged implementation



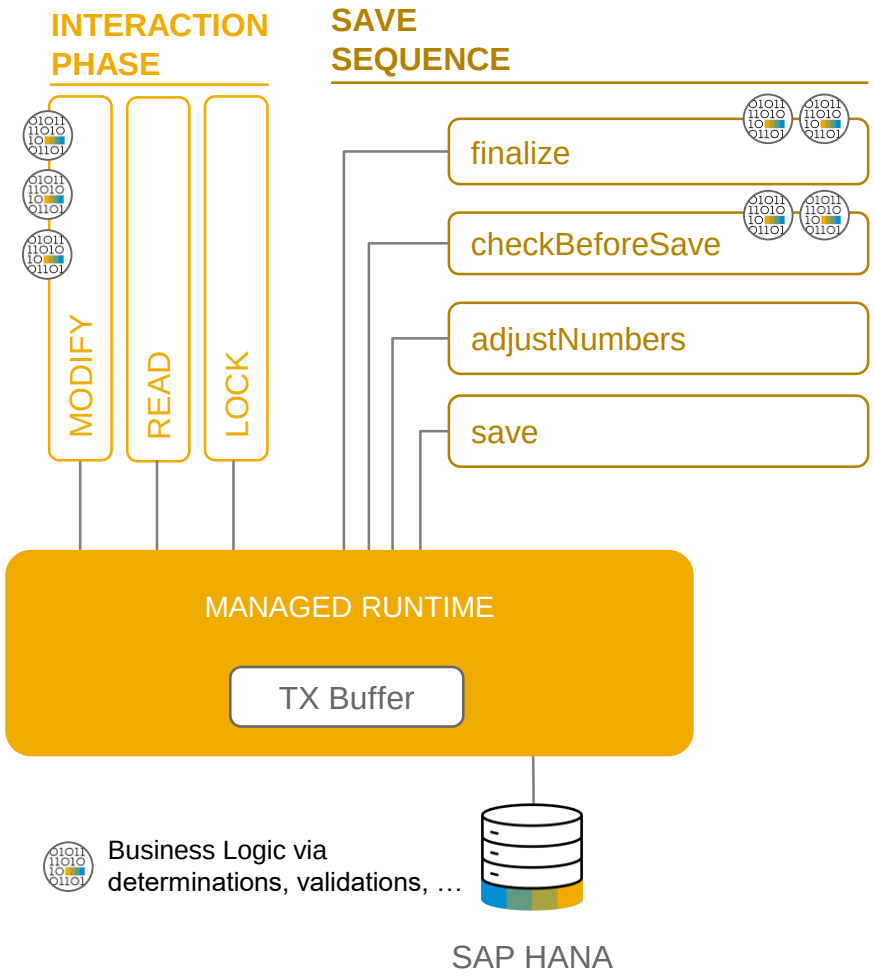
Application coding

- Already available
- For interaction phase, transactional buffer and save sequence
- Decoupled from UI technology

Examples

- Sales Order, Purchase Order

Business objects – Managed implementation



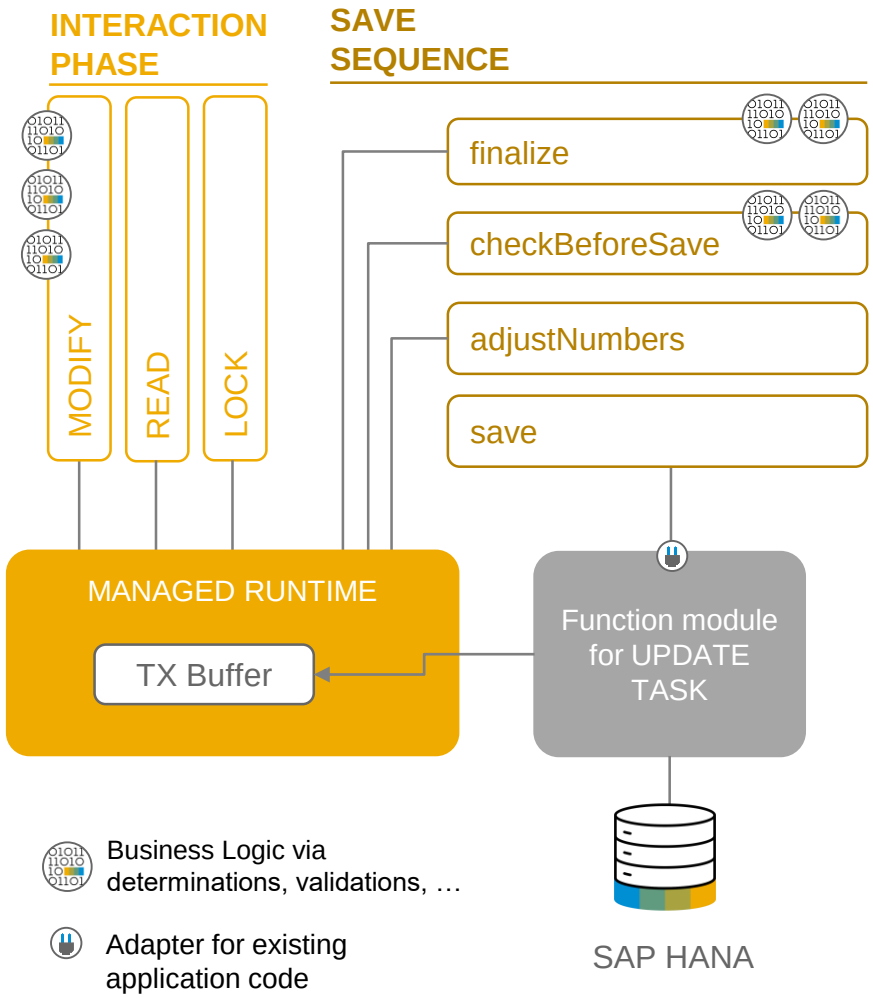
Application coding

- Not yet available or fine granular reusable code available
- Technical implementation tasks taken over by BO infrastructure
- Developer focus on business logic, implemented via code exits: determinations, validation, actions,…

Examples

- New applications in SAP Cloud Platform ABAP Environment

Business objects – Managed impl. with unmanaged appl. components



Application coding

- "update-task function module" available
- Coding for interaction phase not available
  (e.g. highly coupled in older UI technology: DYNP - PBO / PAI)
- Technical implementation aspects to be taken over by BO infrastructure

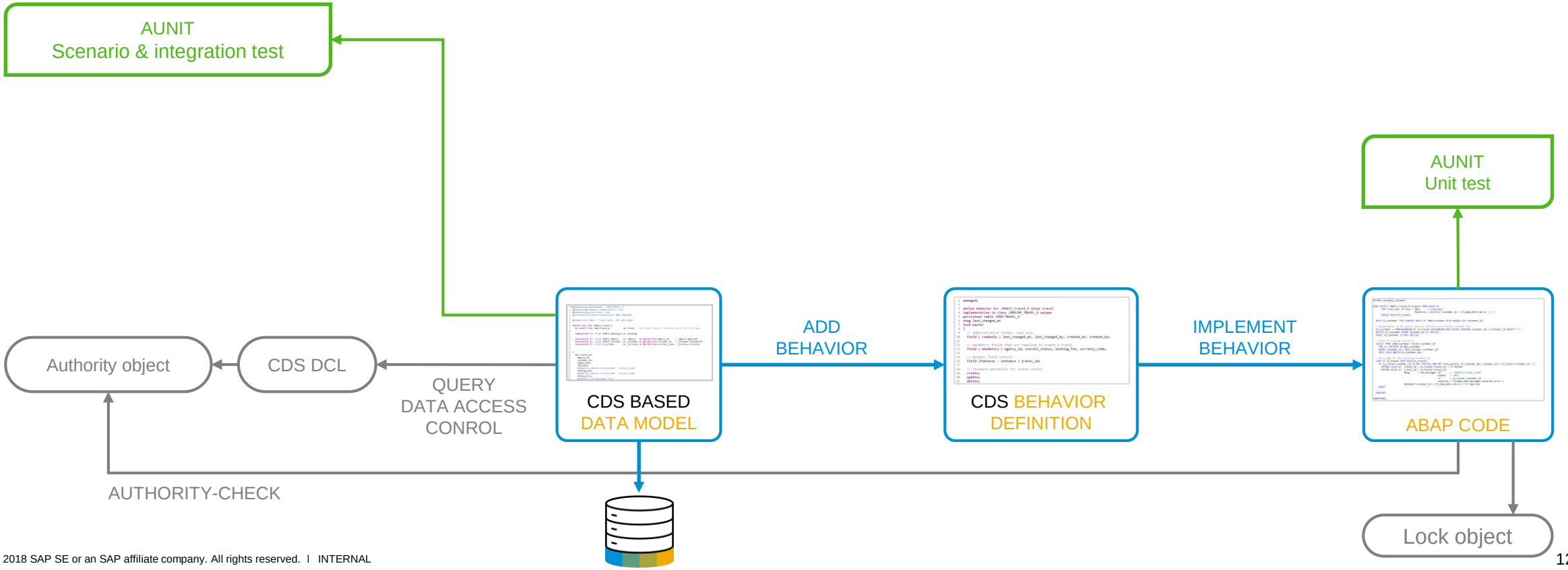
Examples

- Business Partner, Product

Optional unmanaged application components

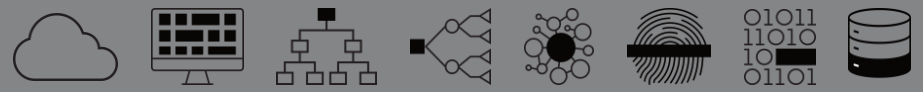
- Update task FM: *unmanaged save*
- Own lock object: *unmanaged lock*
- Mapping between old and new world (e.g. DB tables): type mapping
- Old PFCG Object: *authorization master*

ABAP RESTful Programming Model – Development flow



Business Services

ABAP



What is a business service

SAP Development Learning

Service Binding

Protocol: OData V2 or OData V4?

Scenario: Web API or UI?

Web-API Lifecycle

Release

Versioning

Deprecation

2

3

1

4

Tool support

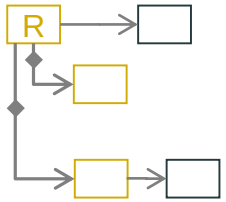
UI: SAP Fiori Elements preview

Web API: Swagger UI preview

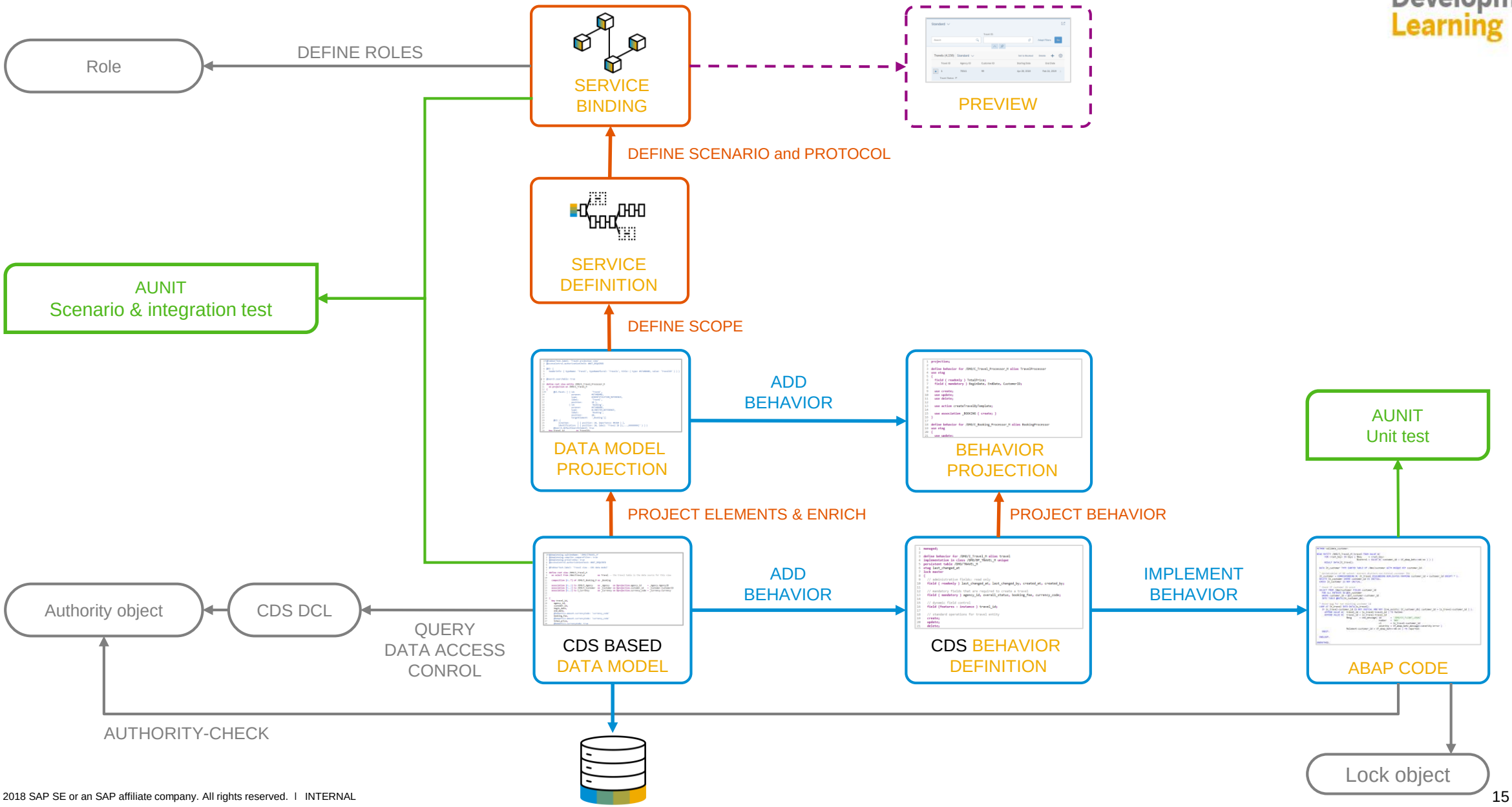
Service Definition: which entities?

BO Projection View: which elements?

Behavior Projection: which behavior?



ABAP RESTful Programming Model – Development flow



Modern ABAP development in Eclipse

SAP Development Learning

MODERN DEVELOPMENT TOOLSET

Fully eclipse-based
Syntax check, Code completion
Syntax highlighting, Pretty printing
Navigation, Search, Quick Fixes

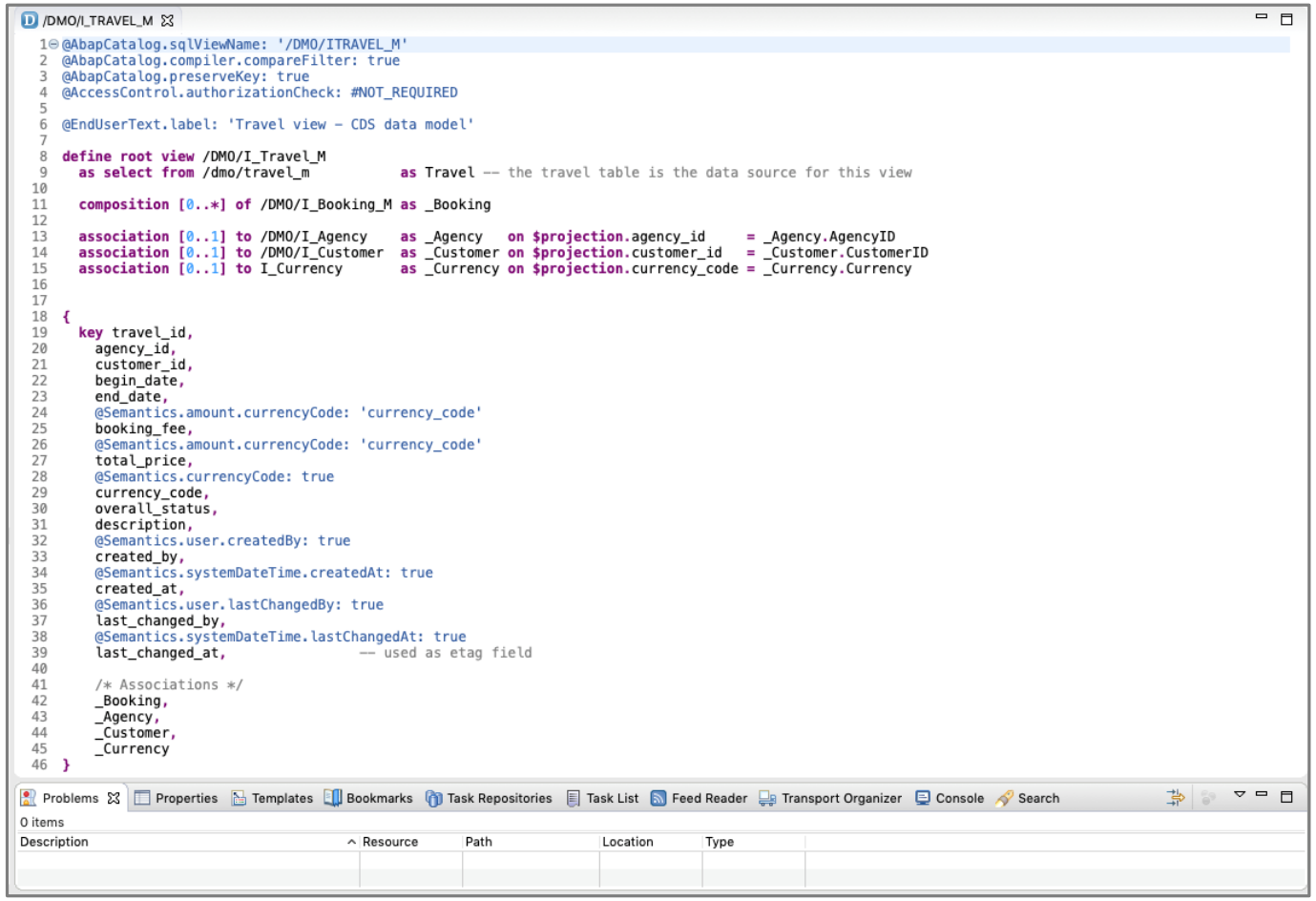
QUALITY ASSURANCE

Static code checks (CVA, ATC) with remote and local scenarios
Unit testing incl. isolation frameworks
Test seams and injections

SUPPORTABILITY

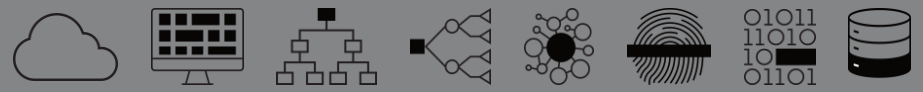
Debugging, profiling
Static and dynamic logging
Runtime monitoring and analysis

ABAP DEVELOPMENT TOOLS (ADT)

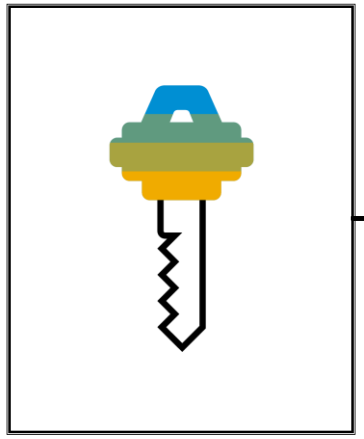


Summary and Outlook

ABAP



Summary – Key takeaways



The **ABAP RESTful Programming Model** is available as of SAP Cloud Platform, ABAP Environment 1808: Cloud-first delivery.

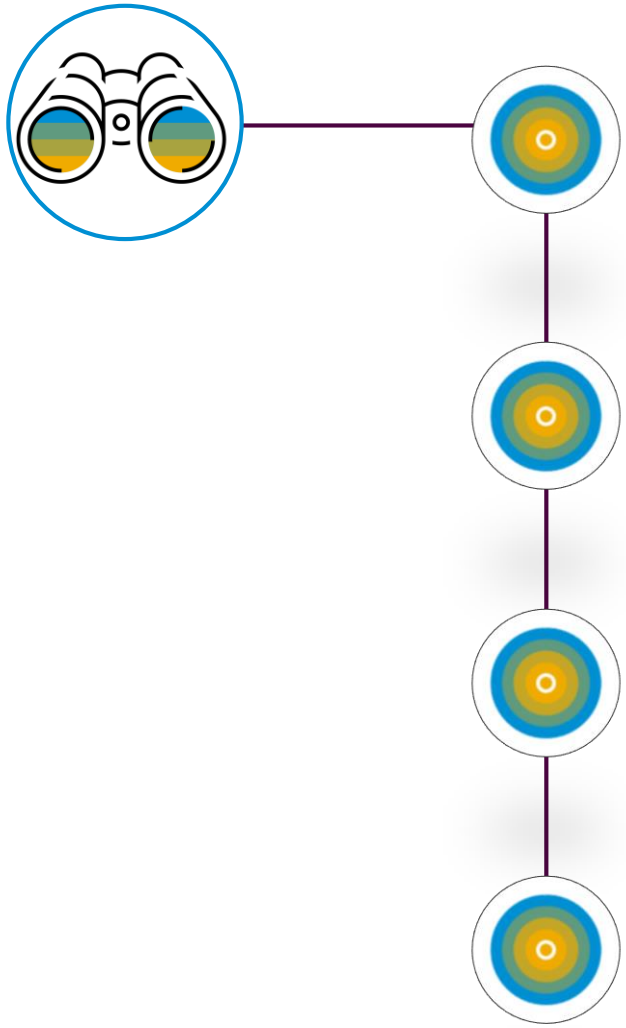
The **ABAP RESTful Programming Model** is now available on-premise with SAP S/4HANA 1909 – with reduced feature scope: *UNMANAGED BO IMPLEMENTATION.*

The feature scope of the **ABAP RESTful Programming Model** is enhanced on a quarterly basis in SAP CP ABAP Environment and on-premise on a yearly basis in SAP S/4HANA.

The **ABAP Programming Model for SAP Fiori** *WITH DRAFT SUPPORT* is available as of AS ABAP 7.5 and remains the current best practice in SAP S/4HANA.

Outlook – Next steps planned for SAP Cloud Platform ABAP Environment

SAP Development Learning



Business objects

Enhanced managed scenario: unmanaged appl. components, draft handling, managed early numbering, late numbering

Business Services

Service binding for OData V4 WebAPIs

Web API consumption

Support for OData V4, REST

Actions & functions

Cross topics

Extensibility

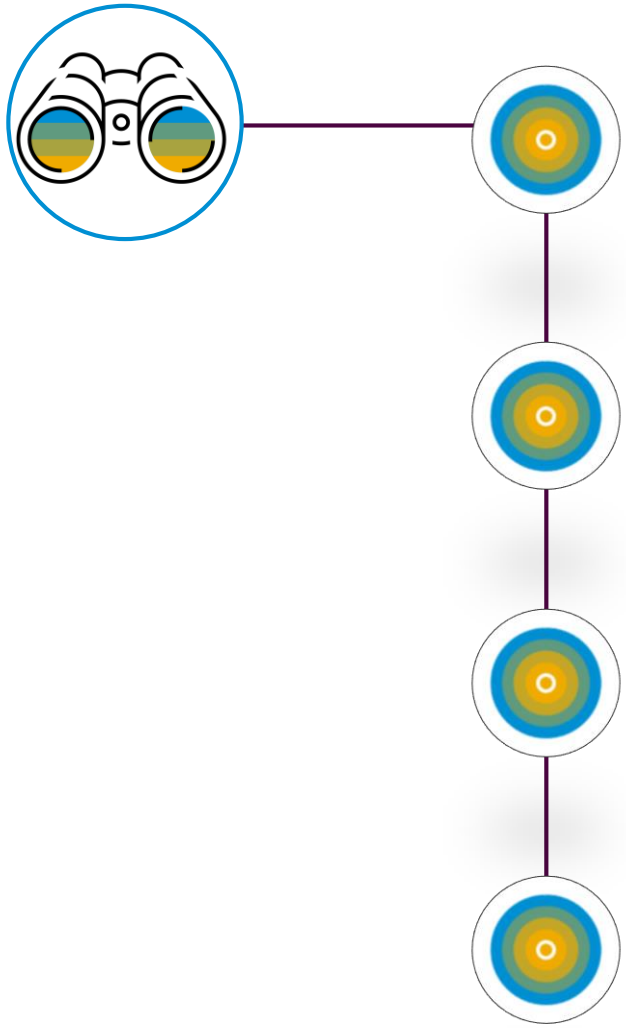
Supportability

Documentation

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

Outlook – Next steps planned for SAP S/4HANA

SAP Development Learning



Business objects

Managed scenario with CRUD based on determinations, validations, actions and functions, eTag, locks, feature control, authorization, draft

Migration of existing CDS-BOPF based BOs

Business Services

Service binding for OData V4 WebAPIs

Web API consumption

Support for OData V4, REST

Actions & functions

Cross topics

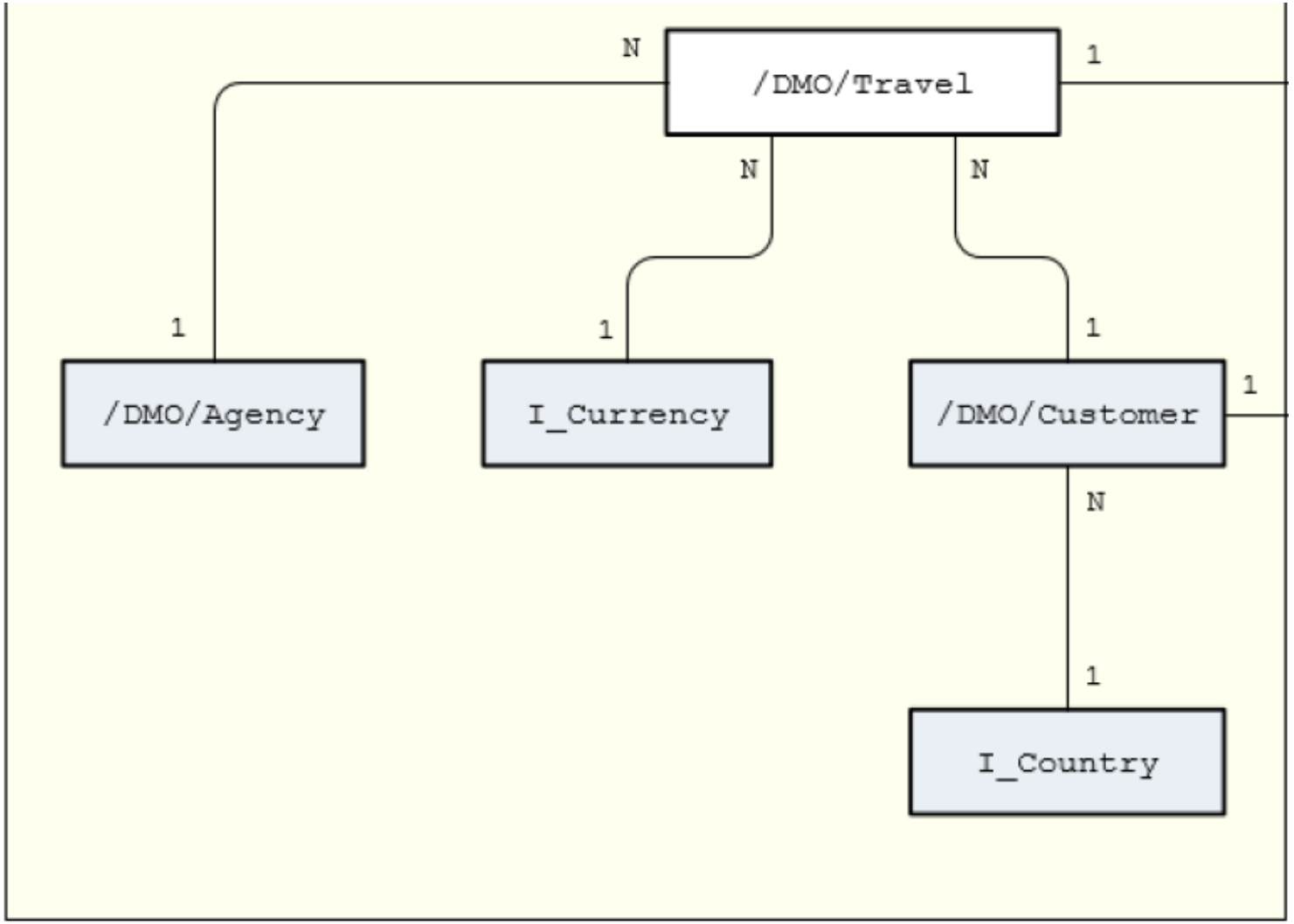
Extensibility

Supportability

Documentation

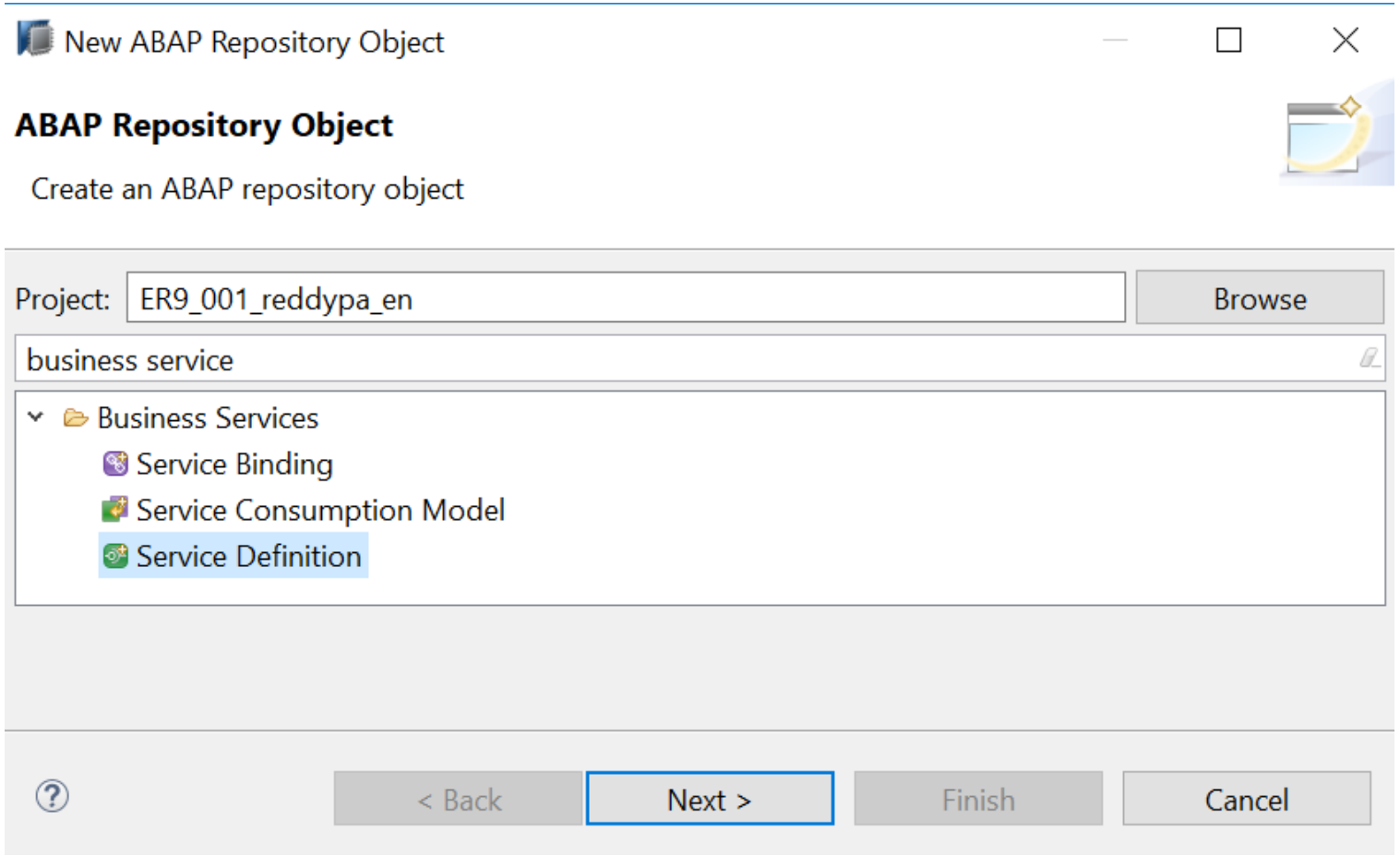
This is the current state of planning and may be changed by SAP at any time.

Data Model



Service Definition

- The service definition is a projection of the models and related behavior that you want to expose.
- You define the OData service to determine which CDS entities are part of the service.
- This service is then bound to a protocol and to either a UI technology or an A2X service by a service binding artifact



Service Definition Editor

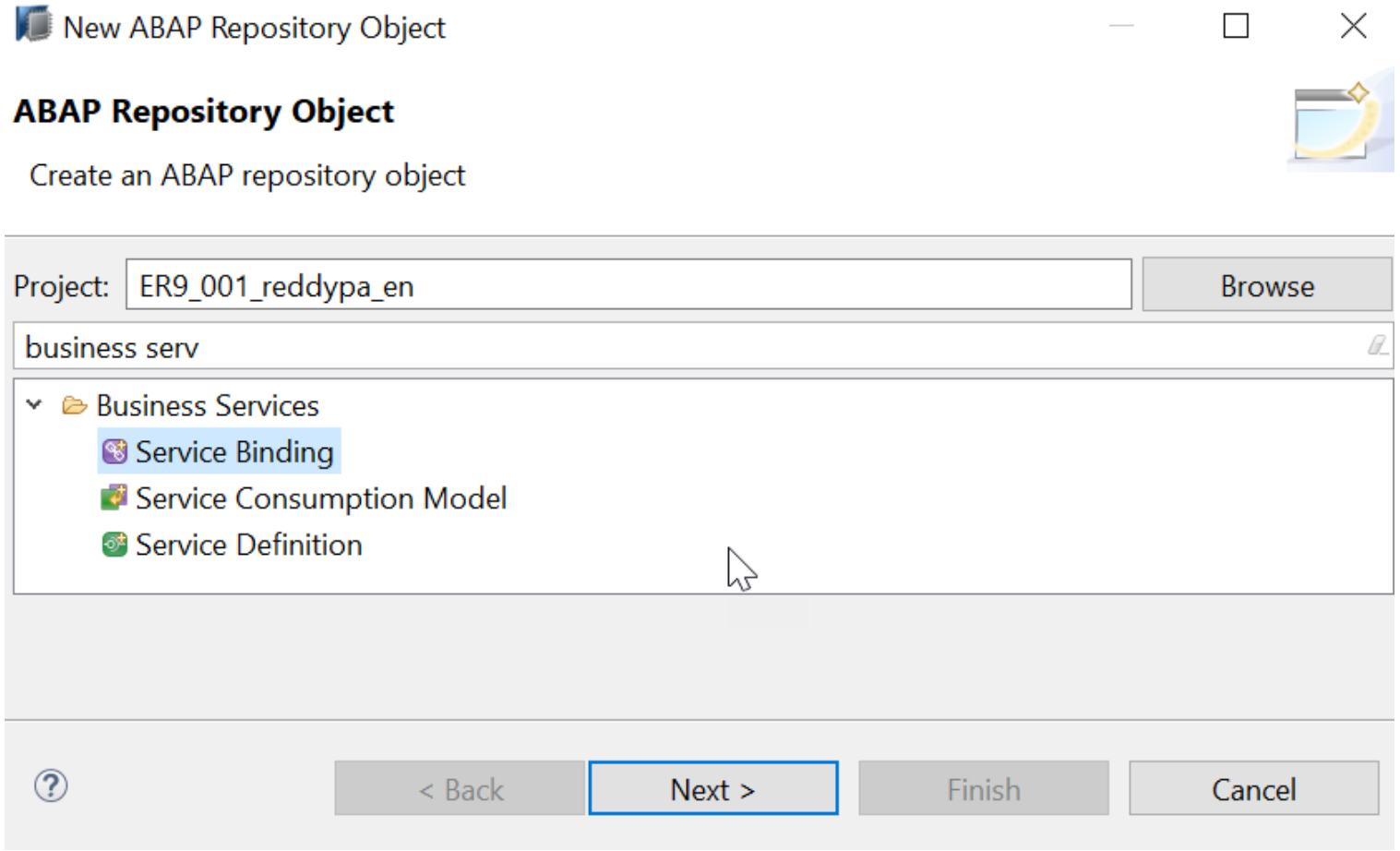
- service definition is used to assign the scope of the OData service.
- Travel, TravelAgency, Passenger, Currency, country CDS views are exposed in the service definition for OData Service
- Optionally, you can assign an alias for the each exposed CDS view

*[ER9] ZI_TRAVEL_U_SD_XX

```
@EndUserText.label: 'Service Definition for Managing Travel'
define service ZI_TRAVEL_U_SD_XX {
  expose ZI_TRAVEL_U_XX as Travel;
  expose ZI_Agency_U_XX as TravelAgency;
  expose ZI_CUSTOMER_U_XX as Passenger;
  expose I_Currency as Currency;
  expose I_Country as Country;
}
```

Service Binding

- A service binding implements the protocol that is used for the OData service.
- It uses a service definition that projects the data models and their related behaviors to the service.



Service Binding Editor

[ER9] ZTRAVEL_SB_U_V2_XX

Service Binding: ZTRAVEL_SB_U_V2_XX

General Information

This section describes general information about this service binding

Binding Type: ODATA V2 (UI - User Interface: Consumed in SAPUI5 Apps)

Service Versions

Define service versions associated with the service binding

type filter text

Version	Service Definition
0001	ZI_TRAVEL_U_SD_XX

Add...

Remove

Service Version Details

View information on selected service version

Local Service Endpoint: Active

Deactivate

Local Service Endpoint

Service URL: /sap/opu/odata/sap/ZTRAVEL_SB_U_V2_XX

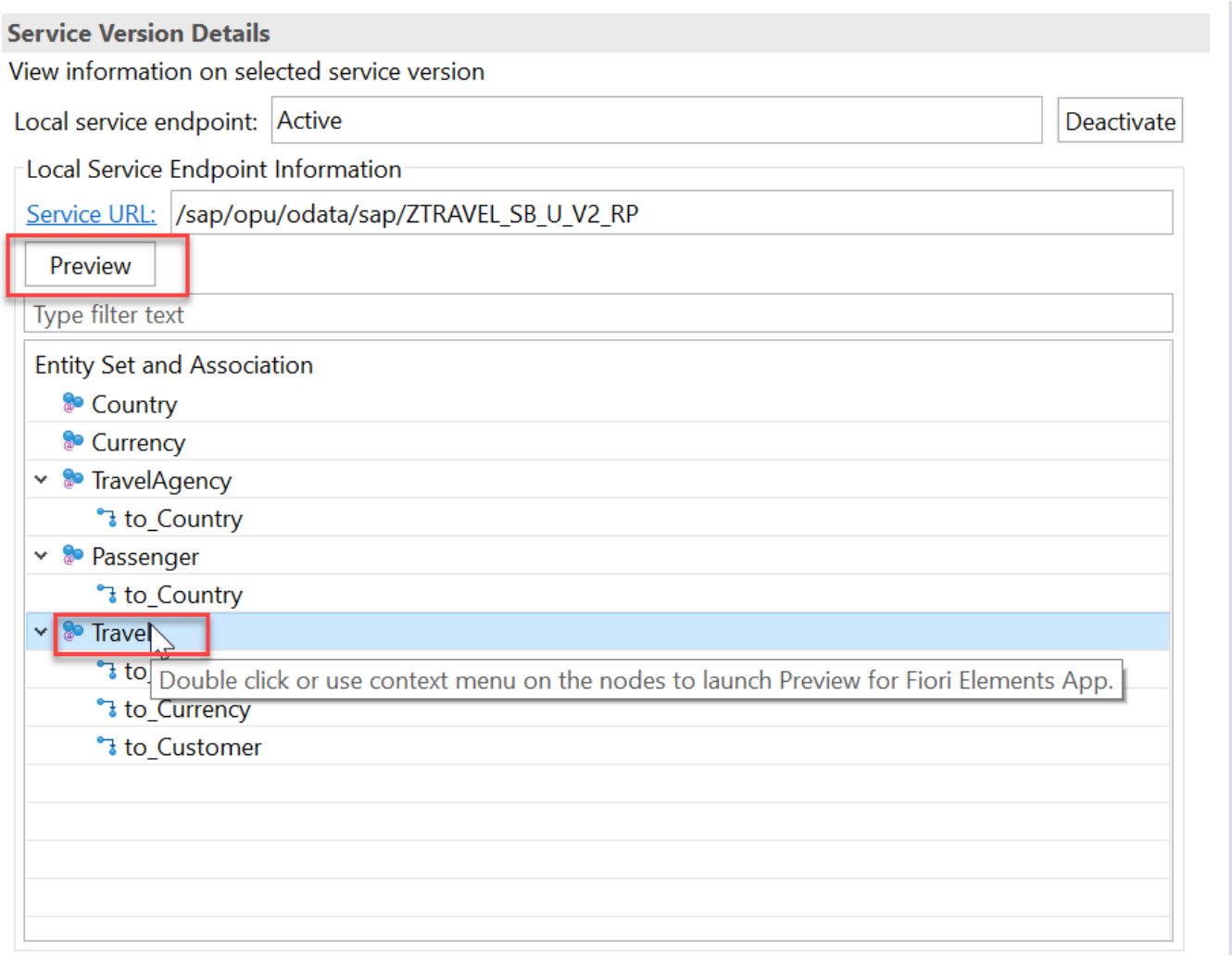
Preview

type filter text

Entity Set and Association

- Country
- Currency
- TravelAgency
  - to_Country
- Passenger
  - to_Country
- Travel
  - to_Agency
  - to_Currency
  - to_Customer

Preview for Fiori Elements App



UI Annotations

- UI annotations can be used in CDS to configure the look of the user interface of a Fiori App.

Annotation	Description
@UI.headerInfo.typeNamePlural:'name'	For the headline of the list, This annotation is an entity annotation because it concerns the whole entity rather than a specific element
@UI.lineItem: [{ position:decfloat }]	Specify a position for each element that you want to show in the list report
@UI.lineItem.label: 'label'	The label is displayed in the column header of the list report.
@UI.selectionField.position:decfloat	Implement selection fields on top of the list report to filter for a specific item
@UI.headerInfo.typeName: 'name'	Specify the title of the object page
@UI.facet.purpose: #STANDARD	Create a standard facet for the object page

UI Annotations

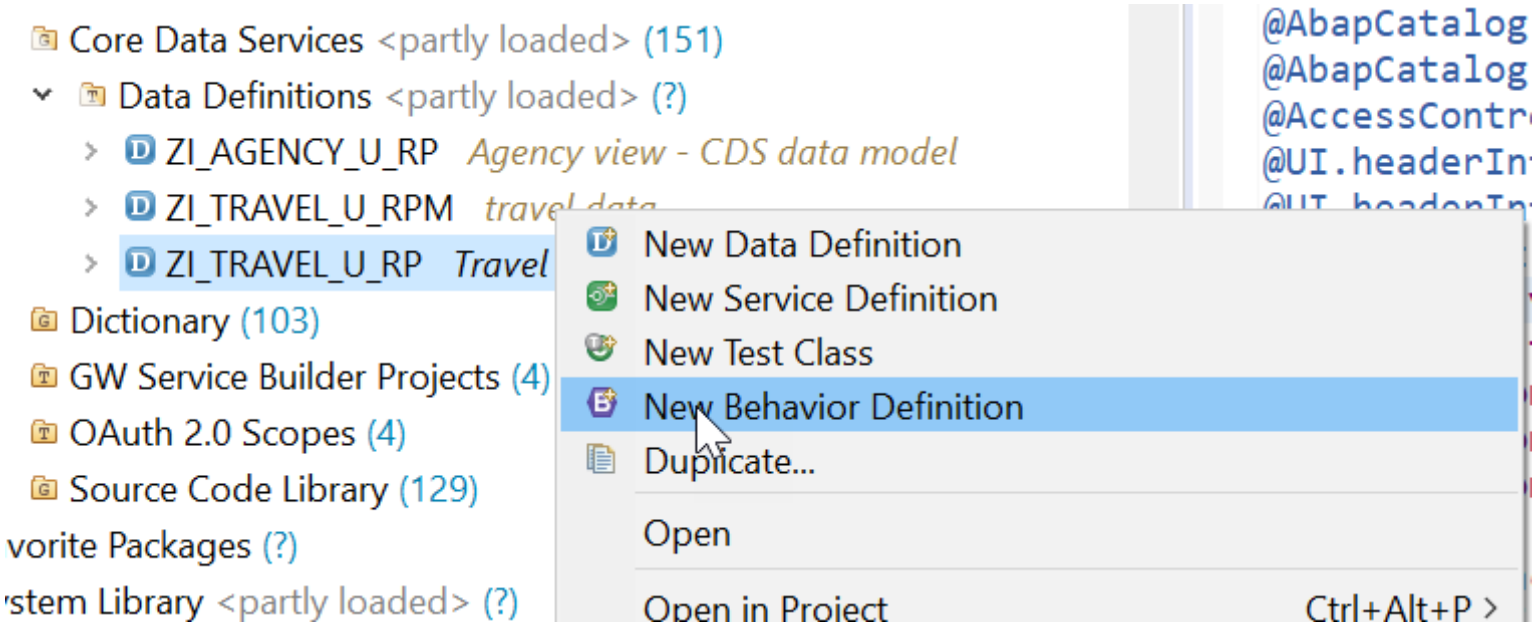
@UI.facet.type: #IDENTIFICATION_REFERENCE	object page displays the detailed information of one list item
@UI.facet.label: 'name'	Specify a name for the object page facet header
@UI.facet.position: decfloat	To define the position of the facet
@UI.identification.label: 'name'	Specify the label for each element that you want to show in the object page
@UI.identification.position: 'decfloat'	Specify the position for each element that you want to show in the object page
@Consumption.valueHelpDefinition	To provide a value help for the selection fields of the elements
@Semantics.text: true	To identify the annotated element as a text
@ObjectModel.text.association: '<_AssocToTextProvider>'	To reference the association as a text association.

UI Annotations

@Search.searchable: true	To enable the CDS view for searches and to expose a standard search field on the UI.If you use this annotation in a CDS view, you must assign at least one default search element.
@Search.defaultSearchElement: true.	Choose the elements that you want to search for
@Search.fuzzinessThreshold: <fuzziness_value>	Define a fuzziness threshold for the searchable elements
@EndUserText.label: '<text>'	Database description labels can be redefined and given more information by using the annotations.Every text that is used in EndUserText annotations is translated into every relevant language by the SAP translation process, along with the labels that are given to the data elements.
@EndUserText.quickInfo: '<text>'	The tooltip is displayed on the UI as mouse over text
@Semantics.currencyCode: true	This annotation tags a field containing a currency code
@Semantics.amount.currencyCode: 'CurrencyCode'	The annotated field contains a monetary amount, and the corresponding currency code is contained in the referenced field.

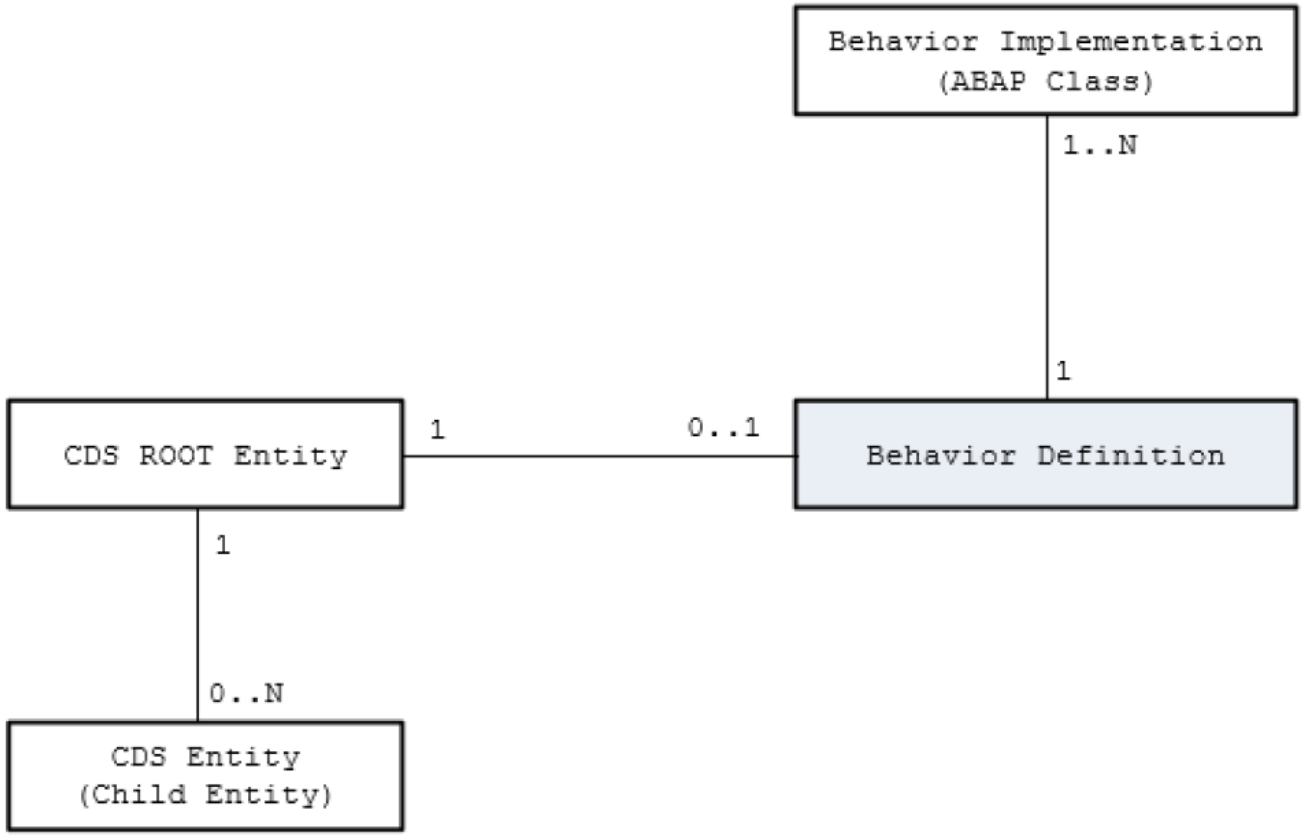
Behavior Definition

- A behavior definition is defined using the Behavior Definition Language (BDL) and comprises capabilities and modelling aspects of the business object node or nodes.
- for example the supported operations (such as create, update, and delete actions) or the definition of lock dependencies between the parent and child nodes.



Relation Between Behavior Definition and Behavior Implementation

- A behavior definition always refers to a CDS data model.
- This means that a CDS data model must always exist before the behavior definition is created.
- One behavior definition refers exactly to one root entity and one CDS root entity has a maximum of one behavior definition, which handles all associated(child) entities



Relationship Between the Data Model, the Behavior Definition, and the Behavior Implementation

Behavior definition editor

- By creating a behavior definition, the referenced root entity and its compositions (in the upcoming versions of the programming model) gain a transactional character.
- The behavior definition is hence a realization of the BO concept.
- All supported transactional operations of a concrete business object must be specified in the same behavior definition

[ER9] ZI_TRAVEL_U_XX

```
implementation unmanaged;
define behavior for ZI_TRAVEL_U_XX alias Travel
{
  field (readonly) CreatedBy;
  create;
  update;
  delete;
  action set_status_booked result [1] $self;
}
```

Behavior Definition Language (BDL) Syntax

- The syntax of the Behavior Definition Language (BDL) is oriented to the Data Definition Language (DDL) used to define CDS entities (camel-case notation).
- Behavior definitions are managed in the ABAP compiler and not ABAP Dictionary.

```
/* Header of behavior definition */
implementation {unmanaged | managed | abstract};
/* Definition of entity behavior */
define behavior for CDSEntity [alias AliasName]
/* Entity properties */
[late numbering]
[etag (field)]
[lock {master | dependent (PropertyDependent = PropertyMaster)}] //Only supported for root entities
{
/* Standard operations */
[internal] create;
[internal] update;
[internal] delete;
/* Actions */
[static] action ActionName
[parameter {InputParameterEntity | $self)}]
[result [cardinality] {OutputParameterEntity | $self}];
/* Associations */
association AssociationName [abbreviation AbbreviationName] {[create;]}
}
```

Behavior Implementation types

SAP Development Learning

Unmanaged

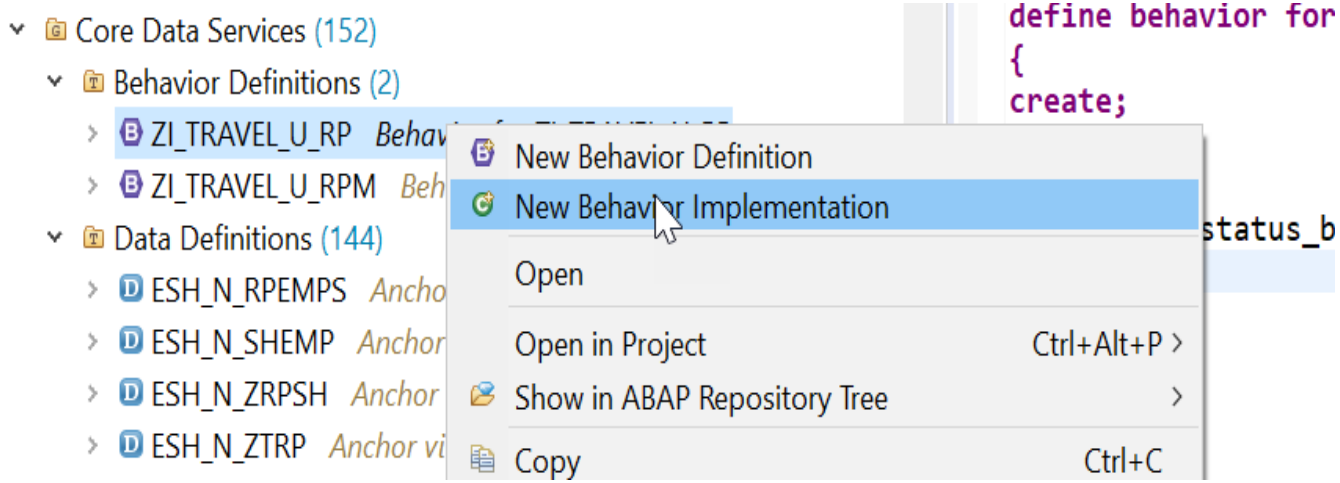
- The application developer must implement essential components of the REST contract itself. In this case, all required operations (create, update, delete, or any application-specific actions) must be specified in the corresponding behavior definition before they are manually implemented in ABAP.

Managed

- In a **managed** implementation type (which is not supported yet), on the other hand, a behavior definition would on its own be sufficient to obtain a ready-to-run business object.

Behavior Implementation

- The implementation of a behavior definition can be done in a single ABAP class (behavior pool) or can be split between an arbitrary set of ABAP classes (behavior pools). You can assign any number of behavior pools to a behavior definition (a 1: n relationship).



Behavior Implementation Class

[ER9] ZI_TRAVEL_U_RP | [ER9] ZI_TRAVEL_U_RP | [ER9] ZCL_TRAVEL_U_RP

ZCL_TRAVEL_U_RP

```
CLASS zcl_travel_u_rp DEFINITION
  PUBLIC
  ABSTRACT
  FINAL
  FOR BEHAVIOR OF zi_travel_u_rp .

  PUBLIC SECTION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.


CLASS zcl_travel_u_rp IMPLEMENTATION.
ENDCLASS.
```

Global Class | Class-relevant Local Types | Local Types | Test Classes | Macros

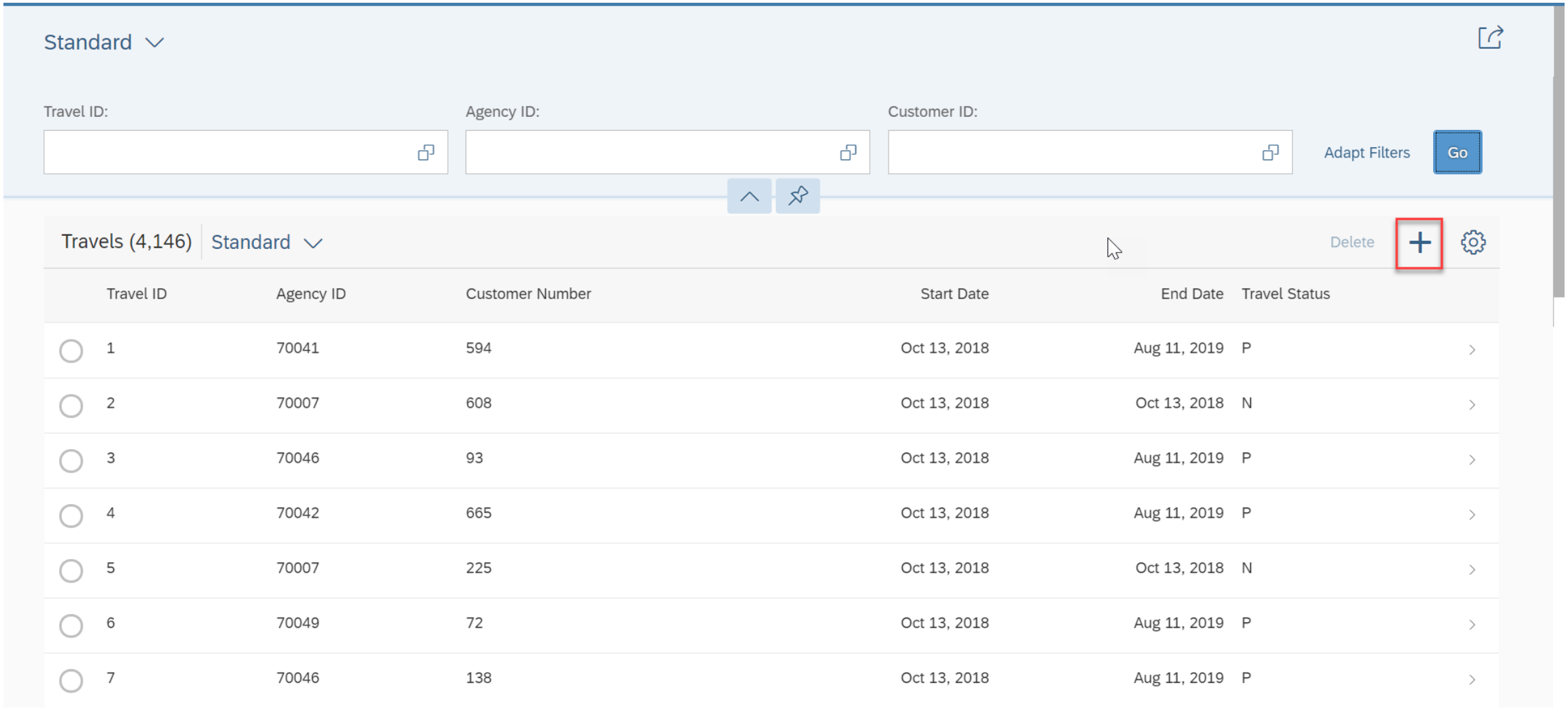
[ER9] /DMO/I_TRAVEL_U | [ER9] ZI_TRAVEL_U_XX | [ER9] ZCL_TRAVEL_U_XX

ZCL_TRAVEL_U_XX ▸ LHC_TRAVEL ▸ CREATE

```
CLASS lhc_travel DEFINITION INHERITING FROM cl_abap_behavior_handler.
  PRIVATE SECTION.
    METHODS create FOR MODIFY IMPORTING entities FOR CREATE travel.
    METHODS delete FOR MODIFY IMPORTING keys FOR DELETE travel.
    METHODS update FOR MODIFY IMPORTING entities FOR UPDATE travel.
    METHODS read FOR READ IMPORTING keys FOR READ travel RESULT result.
    METHODS set_status_booked FOR MODIFY IMPORTING keys FOR ACTION travel~set_status_booked RESULT result.
ENDCLASS.

CLASS lhc_travel IMPLEMENTATION.
  METHOD create.
  ENDMETHOD.

  METHOD delete.
  ENDMETHOD.

  METHOD update.
  ENDMETHOD.

  METHOD read.
  ENDMETHOD.

  METHOD set_status_booked.
  ENDMETHOD.

ENDCLASS.

CLASS lsc_zi_travel_u_xx DEFINITION INHERITING FROM cl_abap_behavior_saver.
  PROTECTED SECTION.
    METHODS check_before_save REDEFINITION.
    METHODS finalize          REDEFINITION.
    METHODS save              REDEFINITION.
ENDCLASS.

CLASS lsc_zi_travel_u_xx IMPLEMENTATION.
  METHOD check_before_save.
  ENDMETHOD.

  METHOD finalize.
  ENDMETHOD.

  METHOD save.
  ENDMETHOD.
ENDCLASS.
```

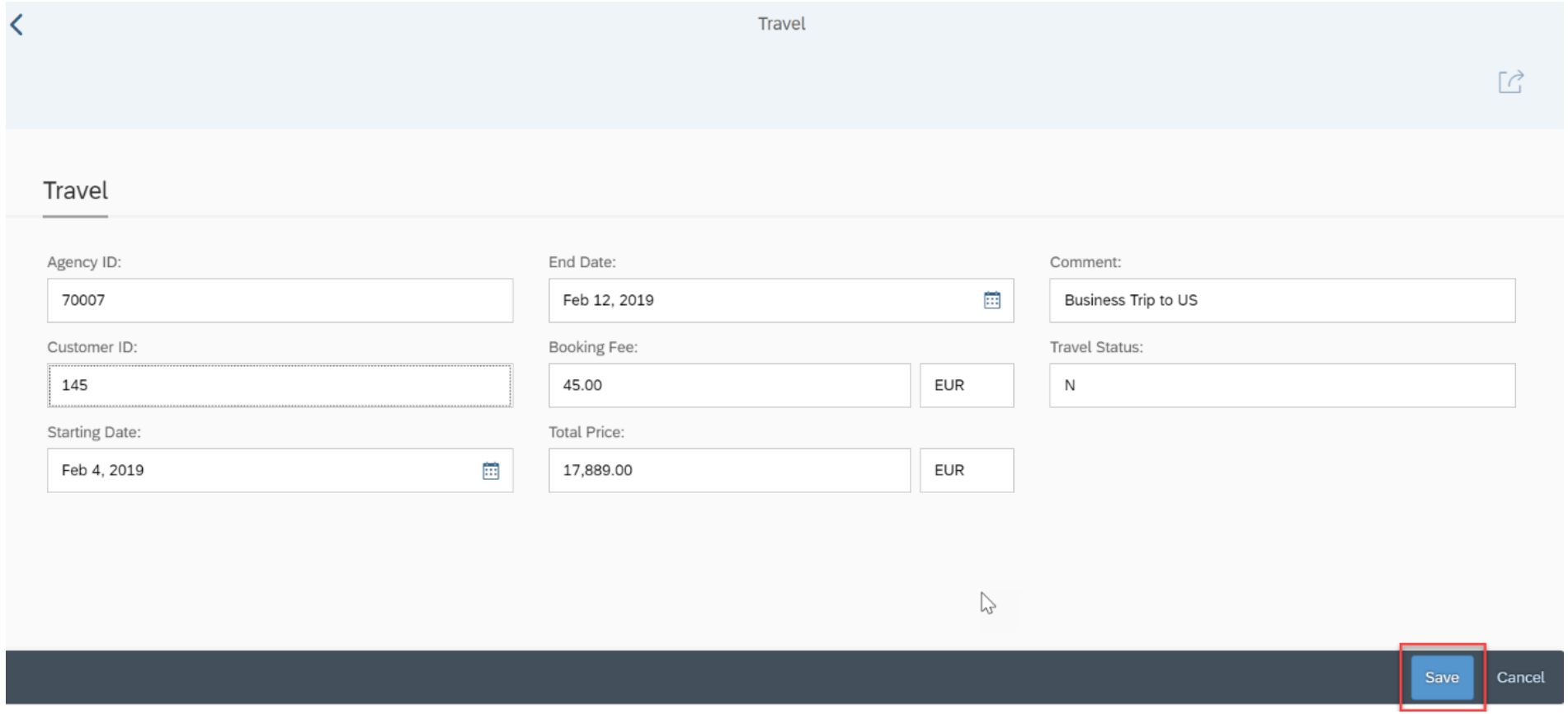
Global Class | Class-relevant Local Types | Local Types | Test Classes | Macros

Travel Application

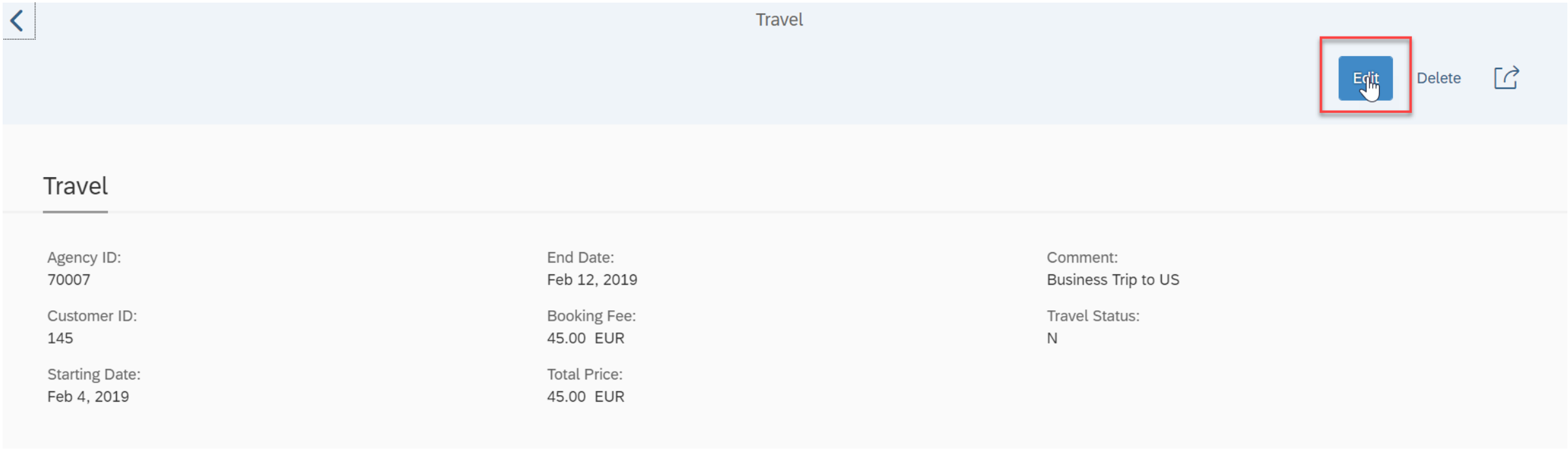


Travel ID	Agency ID	Customer Number	Start Date	End Date	Travel Status
1	70041	594	Oct 13, 2018	Aug 11, 2019	P
2	70007	608	Oct 13, 2018	Oct 13, 2018	N
3	70046	93	Oct 13, 2018	Aug 11, 2019	P
4	70042	665	Oct 13, 2018	Aug 11, 2019	P
5	70007	225	Oct 13, 2018	Oct 13, 2018	N
6	70049	72	Oct 13, 2018	Aug 11, 2019	P
7	70046	138	Oct 13, 2018	Aug 11, 2019	P

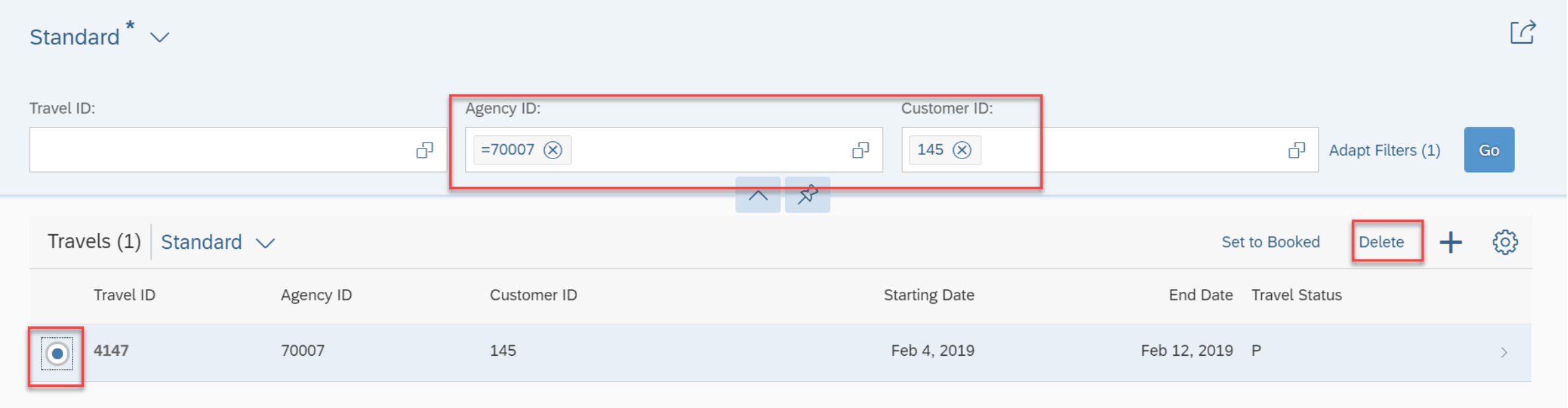
Travel Application - Create



Travel Application - Update



Travel Application - Delete



Travel Application - Action