LOGALI

Teoría

ABAP Cloud Developer Extensibility – Validaciones y Determinaciones

SAP S/4HANA Cloud – Modelo de extensibilidad Clean Core

Contenido

1. ABAP Cloud Developer Extensibility - Validaciones y Determinaciones

1.1. Requerimiento – Extensión RAP Validaciones y Determinaciones

En entornos empresariales, es común encontrar la necesidad de extender las aplicaciones estándar para incluir validaciones adicionales en campos existentes o nuevos, así como la determinación automática de valores. Esto asegura que los datos ingresados sean correctos y consistentes, y facilita la automatización de ciertos procesos.

Requerimiento Específico:

El requerimiento específico incluye la agregación de tres nuevos campos en una aplicación RAP:

- **Tasa:** El porcentaje que cobrará la agencia por un determinado valor, con una validación que acepta valores entre 0 y 100.
- **Moneda:** Campo que incluye una ayuda de búsqueda para seleccionar la moneda adecuada y validaciones sobre los valores ingresados.
- **VAT:** Un campo de tipo read-only que se determinará automáticamente en base al valor del código del país ingresado.

Implementación de los Campos:

Inicialmente, los campos se agregan sin ninguna validación ni determinación implementada. Esto permite tener una visión clara de lo que se desea por parte del negocio y proporciona una base para las futuras implementaciones de validaciones y determinaciones.

Validaciones:

Las validaciones se implementan para asegurar que los valores ingresados en los campos cumplan con ciertos criterios. En el caso del campo de la tasa, la validación asegura que el valor esté entre 0 y 100. Para el campo de moneda, se incluyen ayudas de búsqueda y validaciones específicas sobre los posibles valores a introducir.

Determinaciones:

Las determinaciones se utilizan para asignar automáticamente valores a ciertos campos en función de otros datos ingresados. En el caso del campo VAT, su valor se determina automáticamente basándose en el código del país ingresado. Este campo es de tipo read-only, lo que asegura que su valor no pueda ser modificado manualmente por el usuario.

Este requerimiento de extensión RAP se centra en agregar nuevas validaciones y determinaciones automáticas de valores a los campos de una aplicación estándar. Estas extensiones mejoran la usabilidad y funcionalidad de la aplicación, asegurando la integridad de los datos y automatizando procesos clave. A través de un enfoque paso a paso, se implementan las validaciones y determinaciones necesarias para cumplir con las demandas específicas del negocio.

1.2. Agregación Nuevos Campos

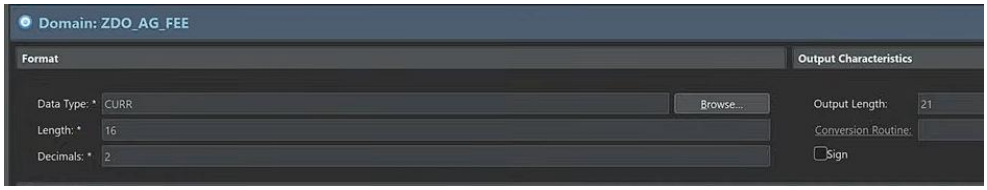
La agregación de nuevos campos en una aplicación RAP estándar es un proceso que involucra múltiples etapas, desde la definición y creación de dominios y elementos de datos, hasta la ampliación de estructuras de persistencia y la implementación de validaciones y determinaciones. Este enfoque asegura que la aplicación no solo cumpla con los requisitos específicos del negocio, sino que también mantenga la integridad y coherencia de los datos.

En el desarrollo de este documento se estará explicando la extensión de las validaciones y determinaciones sobre aplicaciones RAP estándar, identificando dos escenarios principales: la solicitud de validaciones sobre campos ya existentes y la solicitud de nuevos campos. Por otra parte se agregarán nuevos campos que se utilizarán para la explicación de dichos conceptos con el conocimiento explicado en documentaciones anteriores:

Creación de Dominios:

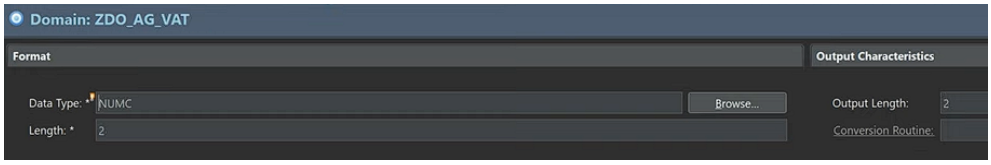
En la carpeta del proyecto de desarrollo de extensión se crearán los siguientes dominios para su utilización posterior en los elementos de datos:

- **ZDO_AG_FEE:** Del tipo **CURR** con una longitud de 16 con 2 decimales.



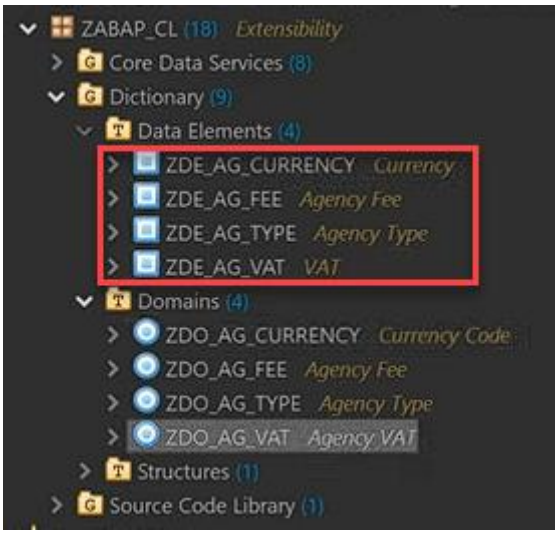
- **ZDO_AG_CURRENCY:** Del tipo **CUKY** con una longitud de 5.

- **ZDO_AG_VAT:** Del tipo **NUMC** con una longitud de 2.



Creación de Elementos de datos:

En la carpeta del proyecto de desarrollo de extensión se crearán los siguientes elementos de datos para su utilización posterior en los elementos de datos:



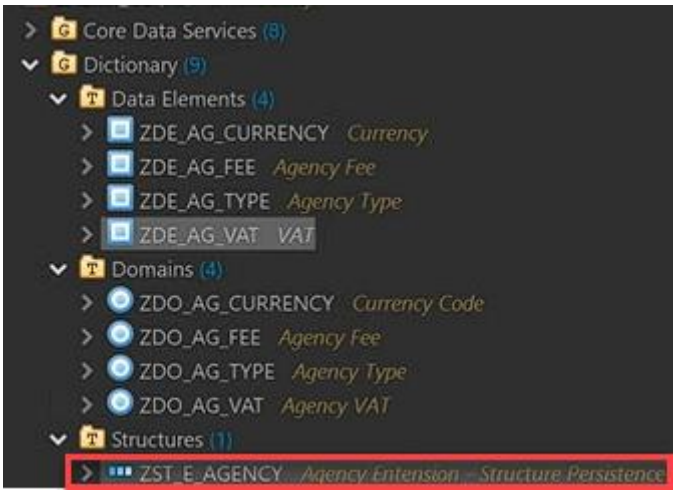
- **ZDE_AG_FEE:** Que utiliza el dominio **ZDO_AG_FEE**.

- **ZDE_AG_CURRENCY:** Que utiliza el dominio **ZDO_AG_CURRENCY**.

- **ZDE_AG_VAT:** Que utiliza el dominio **ZDO_AG_VAT**.

Ampliación de la capa de persistencia:

Se ampliará la estructura de ampliación de la capa de persistencia para incluir 3 nuevos campos que harán referencia a los elementos de datos creados con anterioridad:

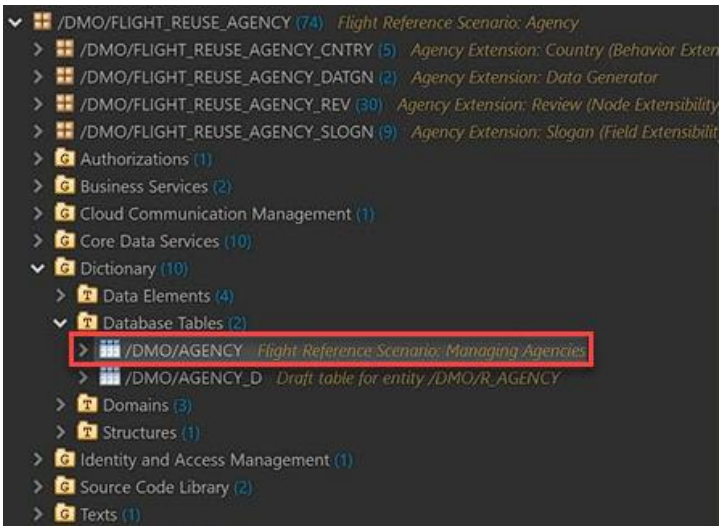



```
@EndUserText.label : 'Agency Extension - Structure Persistence'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
extend type /dmo/s_ext_incl_agency with zst_e_agency {

  zzagtypezag  : zde_ag_type;
  @Semantics.amount.currencyCode : 'zst_e_agency.zzcurencyzag'
  zzagfeezag   : zde_ag_fee;
  zzcurencyzag : zde_ag_currency;
  zzvatzag     : zde_ag_vat;

}
```

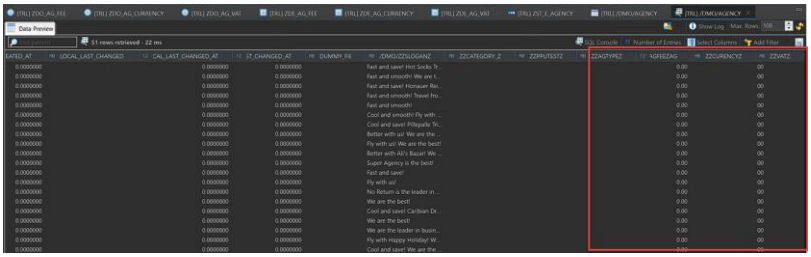
Donde el campo del tipo currency se asocia al campo de moneda por medio de una anotación semántica.

```
@EndUserText.label : 'Agency Extension - Structure Persistence'
@AbapCatalog.enhancement.category : #NOT_EXTENSIBLE
extend type /dmo/s_ext_incl_agency with zst_e_agency {

  zzagtypezag  : zde_ag_type;
  @Semantics.amount.currencyCode : 'zst_e_agency.zzcurencyzag'
  zzagfeezag   : zde_ag_fee;
  zzcurencyzag : zde_ag_currency;
  zzvatzag     : zde_ag_vat;

}
```

Donde al ampliar la estructura se puede comprobar que han sido agregados los nuevos campos en la tabla de base de datos.




```
@EndUserText.label : 'Flight Reference Scenario: Managing Agencies'
@AbapCatalog.enhancement.category : #EXTENSIBLE_ANY
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table /dmo/agency {

  key client           : abap.clnt not null;
  key agency_id        : /dmo/agency_id not null;
  name                 : /dmo/agency_name;
  street               : /dmo/street;
  postal_code          : /dmo/postal_code;
  city                 : /dmo/city;
  country_code         : land1;
  phone_number         : /dmo/phone_number;
  email_address        : /dmo/email_address;
  web_address          : /dmo/web_address;
  attachment           : /dmo/attachment;
```



Ampliación de la capa de draft:

Como la tabla draft está utilizando la misma estructura que la capa de persistencia para añadir los campos no es necesario agregar una nueva que haga referencia a ella. Pero si en necesario ampliar la vista de extensión de la entidad draft con los nuevos campos

```
extend view entity /DMO/R_AgencyDraft with
{
  Agency.zzagtypezag,
  @Semantics.amount.currencyCode: 'zzcurencyzag'
  Agency.zzagfeezag,
  Agency.zzcurencyzag,
  Agency.zzvatzag
}
```

Ampliación de la entidad raíz:

Luego se amplía la vista de extensión de la entidad asociación para añadir los campos que se utilizaran como fuente de datos en la entidad raíz.

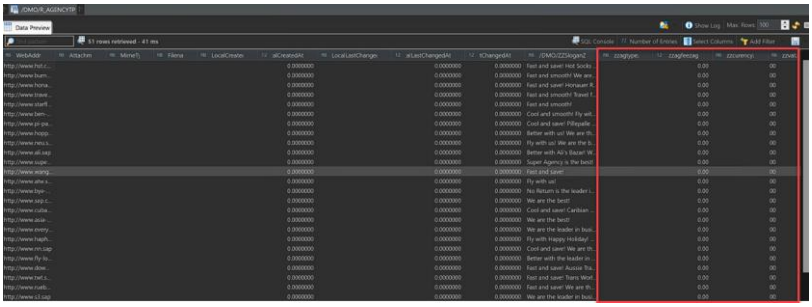

```
extend view entity /DMO/E_Agency with {
    Agency.zzagtypezag,
    @Semantics.amount.currencyCode: 'zzcurencyzag'
    Agency.zzagfeezag,
    Agency.zzcurencyzag,
    Agency.zzvatzag
}
```

Para luego ampliar la vista de extensión de la entidad raíz.


```
extend view entity /DMO/R_AgencyTP with {
    _Extension.zzagtypezag,
    @Semantics.amount.currencyCode: 'zzcurencyzag'
    _Extension.zzagfeezag,
    _Extension.zzcurencyzag,
    _Extension.zzvatzag

}
```

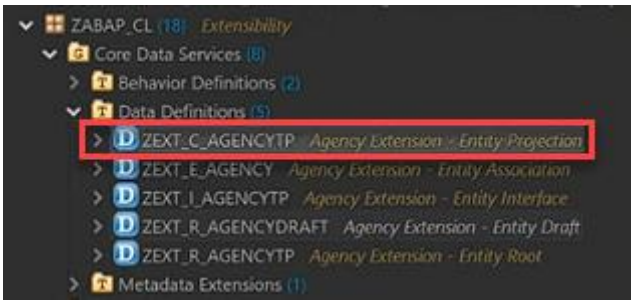
Comprobando mediante la ejecución de la entidad raíz que los campos fueron agregados exitosamente.

```
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'Agency'
@Search.searchable: true

@AbapCatalog.extensibility: {
  extensible: true,
  elementSuffix: 'ZAG',
  allowNewDatasources: false,
  dataSources: ['_Extension'],
  quota: {
    maximumFields: 500,
    maximumBytes: 50000
  },
  allowNewCompositions: true
}

define root view entity /DMO/R_AgencyTP
  as select from /DMO/I_Agency as Agency
  association [0..1] to I_Country     as _Country   on $projection.CountryCode = _Country.Country
  association [1]    to /DMO/E_Agency as _Extension on $projection.AgencyID = _Extension.AgencyIC
```



Ampliación de la entidad raíz:

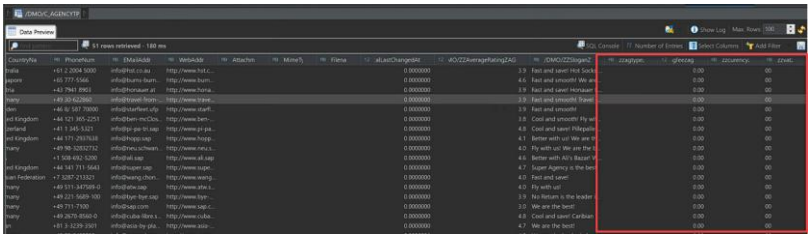
Posteriormente ampliar la vista de extensión de la entidad de consumo.




```
extend view entity /DMO/C_AgencyTP with
{
  Agency.zzagtypezag,
  @Semantics.amount.currencyCode: 'zzcurencyzag'
  Agency.zzagfeezag,
  Agency.zzcurencyzag,
  Agency.zzvatzag
}
```

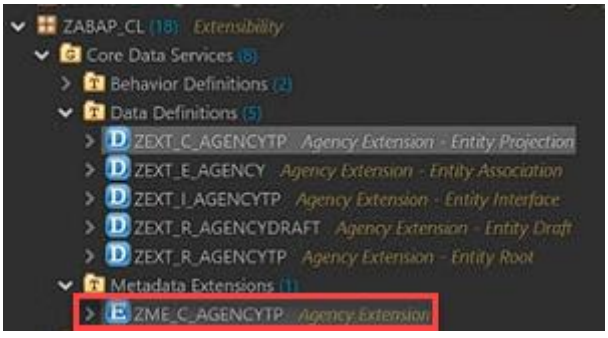
Y realizando las comprobaciones mediante la ejecución de la misma.

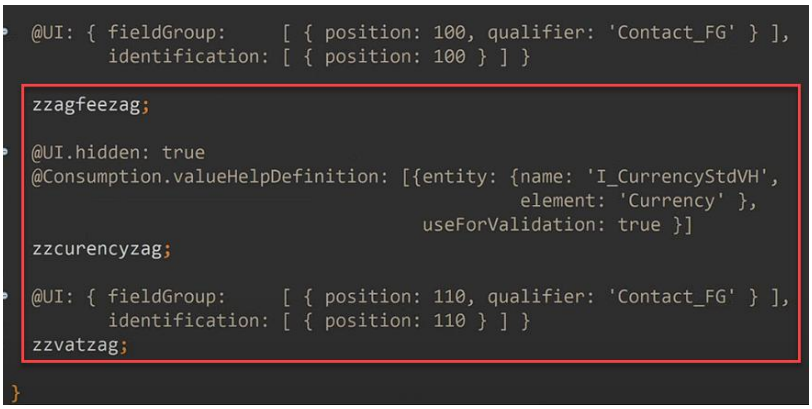
```
@EndUserText.label: 'Agency'
@AccessControl.authorizationCheck: #NOT_REQUIRED

@Search.searchable: true
@Metadata.allowExtensions: true
@ObjectModel.semanticKey: ['AgencyID']

@AbapCatalog.extensibility: {
  extensible: true,
  dataSources: ['Agency'],
  elementSuffix: 'ZAG',
  quota: {
    maximumFields: 500,
    maximumBytes: 50000
  }, allowNewCompositions: true
}

define root view entity /DMO/C_AgencyTP
  provider contract transactional_query
  as projection on /DMO/R_AgencyTP as Agency
```



Ampliación de la entidad raíz:

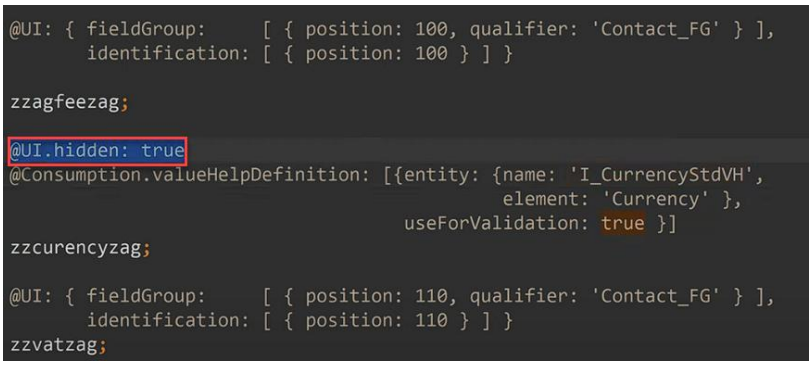
Por último agregar los campos en el metadata extensión de una capa superior **#CUSTOMER,** para poder habilitar características especiales en la interfaz de usuario a los nuevos campos agregados.





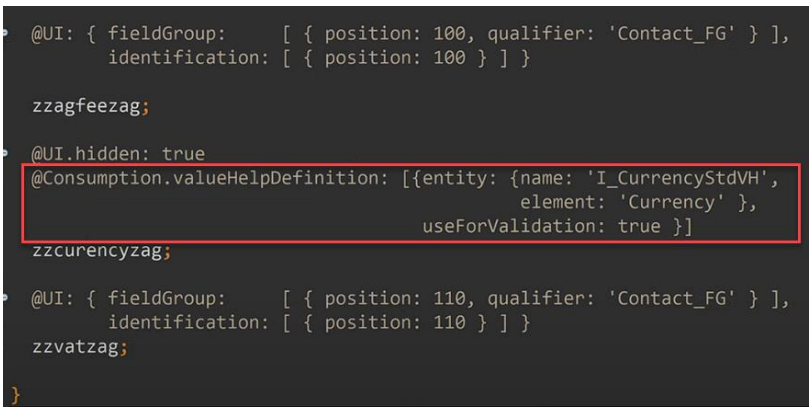
```
  @UI: { fieldGroup:     [ { position: 100, qualifier: 'Contact_FG' } ],
         identification: [ { position: 100 } ] }

  zzagfeezag;

  @UI.hidden: true
  @Consumption.valueHelpDefinition: [{entity: {name: 'I_CurrencyStdVH',
                                               element: 'Currency' },
                                     useForValidation: true }]
  zzcurencyzag;

  @UI: { fieldGroup:     [ { position: 110, qualifier: 'Contact_FG' } ],
         identification: [ { position: 110 } ] }
  zzvatzag;

}
```

Aunque por los momentos los campos deberán permanecer ocultos por lo que se añadió a cada uno de ellos la anotación **@UI.hidden: true.** Que permitirá ocultar los campos



```
@UI: { fieldGroup:     [ { position: 100, qualifier: 'Contact_FG' } ],
       identification: [ { position: 100 } ] }

zzagfeezag;

@UI.hidden: true
@Consumption.valueHelpDefinition: [{entity: {name: 'I_CurrencyStdVH',
                                             element: 'Currency' },
                                   useForValidation: true }]
zzcurencyzag;

@UI: { fieldGroup:     [ { position: 110, qualifier: 'Contact_FG' } ],
       identification: [ { position: 110 } ] }
zzvatzag;
```

Mediante la anotación **@Consumption.valueHelpDefinition.** Permitirá añadir una ayuda de búsqueda o matchcode al campo en la interfaz de usuario y por medio la propiedad **entity.name,** asociarlo con un entidad con los datos que se consultarán.

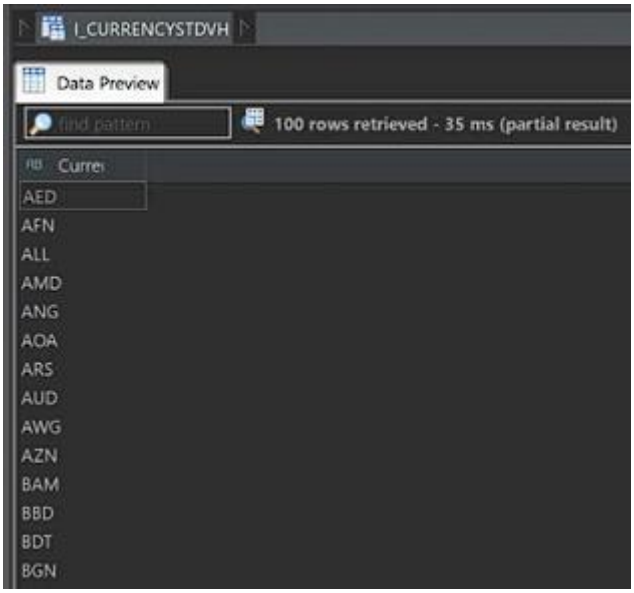


```
  @UI: { fieldGroup:     [ { position: 100, qualifier: 'Contact_FG' } ],
         identification: [ { position: 100 } ] }

  zzagfeezag;

  @UI.hidden: true
  @Consumption.valueHelpDefinition: [{entity: {name: 'I_CurrencyStdVH',
                                               element: 'Currency' },
                                     useForValidation: true }]
  zzcurencyzag;

  @UI: { fieldGroup:     [ { position: 110, qualifier: 'Contact_FG' } ],
         identification: [ { position: 110 } ] }
  zzvatzag;

}
```

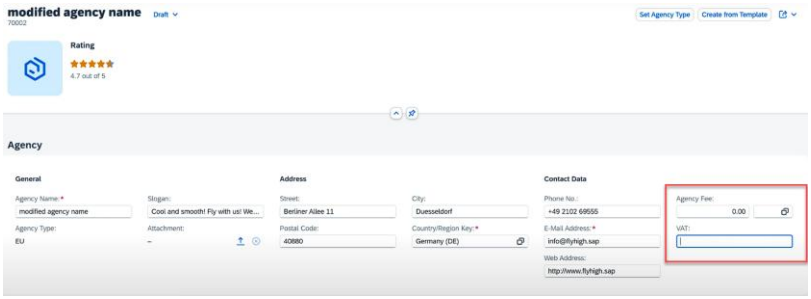
```
  @UI: { fieldGroup:     [ { position: 100, qualifier: 'Contact_FG' } ],
         identification: [ { position: 100 } ] }

  zzagfeezag;

  @UI.hidden: true
  @Consumption.valueHelpDefinition: [{entity: {name: 'I_CurrencyStdVH',
                                               element: 'Currency' },
                                     useForValidation: true }]
  zzcurencyzag;

  @UI: { fieldGroup:     [ { position: 110, qualifier: 'Contact_FG' } ],
         identification: [ { position: 110 } ] }
  zzvatzag;

}
```

Los datos de la entidad utilizada en la ayuda de búsqueda se pueden consultar en la entidad **I_CurrencyStdVH,** al ejecutarla.

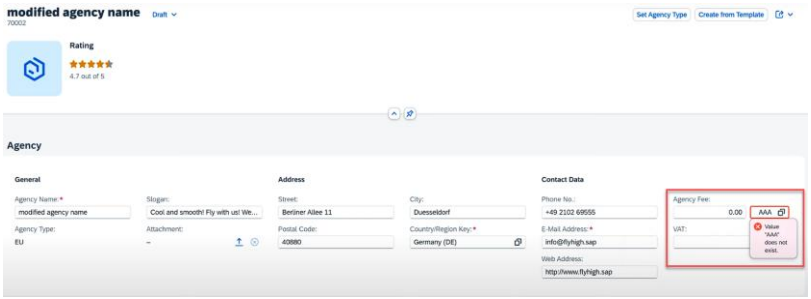
```
@AbapCatalog.sqlViewName: 'ICURRSTDVH'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey:true
@ClientHandling.algorithm: #SESSION_VARIABLE
@VDM.viewType: #COMPOSITE
@VDM.lifecycle.contract.type: #PUBLIC_LOCAL_API
@ObjectModel: { dataCategory: #VALUE_HELP,
                representativeKey: 'Currency',
                usageType.sizeCategory: #S,
                usageType.dataClass: #CUSTOMIZING,
                usageType.serviceQuality: #A,
                supportedCapabilities: [#VALUE_HELP_PROVIDER, #SEARCHABLE_ENTITY],
                modelingPattern: #VALUE_HELP_PROVIDER }
@AccessControl.authorizationCheck: #NOT_REQUIRED
@Metadata.ignorePropagatedAnnotations: true
@EndUserText.label: 'Currency'
@Search.searchable: true
@Consumption.ranked: true
define view I_CurrencyStdVH as select from I_Currency {
```



Al recargar la página con la aplicación RAP estándar a modificar se puede observar que los campos fueron agregados correctamente.



Además que las ayudas de búsqueda añadidas funcionan correctamente.

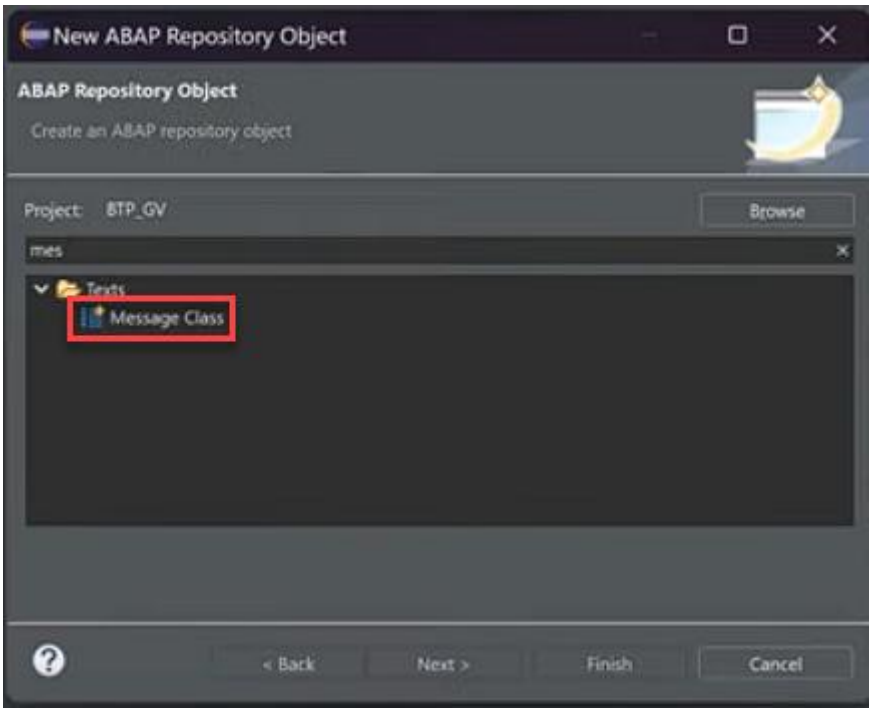


1.3. Clase de Mensajes

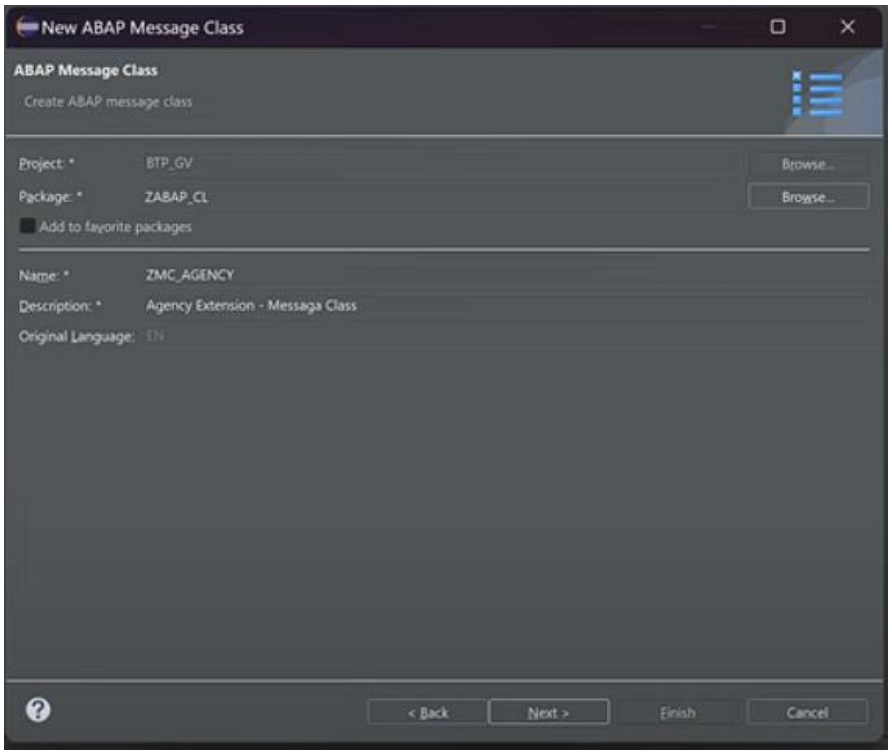
La clase de mensajes es una herramienta utilizada para validar y determinar la aplicación de RAP estándar como una extensión. Este proceso implica varias actividades cruciales que deben llevarse a cabo para satisfacer el requerimiento de validación. En este documento, se detallan los pasos necesarios para crear y vincular textos específicos a los usuarios finales, así como para gestionar la traducción de estos mensajes. Los textos que se mostrarán a los usuarios finales pueden ser textos preexistentes o nuevos, dependiendo de los requerimientos. En esta lección, se describe cómo crear una nueva clase de mensaje en el paquete de desarrollo.

Creación de la Clase de Mensaje:

Al hacer clic derecho en el nombre del paquete de desarrollo y utilizar la opción 'New' se debe seleccionar la opción “Message Class” para crear un nuevo objeto de clase de mensaje.



Colocar un nombre, descripción y asignar el nuevo objeto a una orden de transporte.



Se deben agregar nuevos registros para vincular un identificador con un texto de mensaje. En este caso se asignó el identificador “001”.

Y se agregó un texto específico que indica los parámetros de validación para el usuario final. En este caso "The agency fee must be higher than Zero and lower than 100."

Este mensaje se mostrará cuando el usuario introduzca un valor no esperado según la validación solicitada (valores entre 0 y 100).

Además la traducción de estos textos se puede realizar a través de la aplicación Maintain Translation disponible en el Launchpad. Permitiendo aplicar las traducciones correspondientes a los textos vinculados.

1.4. Clase de excepción – Gestión textos Mensajes

La clase de excepción en la gestión de textos de mensajes es fundamental para asegurar la correcta vinculación y validación de textos en la aplicación RAP. A través de las actividades y definiciones descritas, se asegura una implementación coherente y efectiva de mensajes de usuario, permitiendo una experiencia de usuario fluida y clara.

Definición de la Clase de excepción:

La definición de la clase de excepción inicia heredando de la clase estándar **CX_STATIC_CHECK**, la cual, a su vez, hereda de la clase **CX_ROOT**. Esta herencia en cadena permite acceder de manera genérica a cualquier objeto de excepción, garantizando así una verificación de sintaxis y una estabilidad del programa óptimas.

Luego en la encapsulación pública, se define la implementación de las interfaces **if_t100_dyn_msg**, **if_t100_message** y if_abap_behv_message. Estas interfaces serán necesarias para la vinculación de textos de mensajes.

Constantes y Mensajes de Validación:

Se define una constante que contendrá el nombre de la clase de mensaje que se desea vincular.

```
class zcm_agency definition
  public
  inheriting from cx_static_check
  final
  create public .

  public section.
    interfaces if_t100_dyn_msg.
    interfaces if_t100_message.
    interfaces if_abap_behv_message.

  constants gc_msgid type symsgid value 'ZMC_AGENCY'.

  protected section.
  private section.
endclass.



class zcm_agency implementation.
endclass.
```

A continuación, se define un grupo de constantes que se utilizará para establecer las propiedades necesarias para acceder a un mensaje específico de la clase de mensajes. En este grupo de constantes, se configuran parámetros como el nombre de la clase de mensajes, el identificador del mensaje y hasta cuatro atributos adicionales para valores dinámicos que pueden utilizarse para personalizar el texto del mensaje según sea necesario. Es importante tener presente que se debe crear un nuevo grupo de constantes por cada mensaje que se desee utilizar.

```
  constants gc_msgid type symsgid value 'ZMC_AGENCY'

  constants:
      begin of invalid_fee,
        msgid type symsgid value 'ZMC_AGENCY',
        msgno type symsgno value '001',
        attr1 type scx_attrname value '',
        attr2 type scx_attrname value '',
        attr3 type scx_attrname value '',
        attr4 type scx_attrname value '',
      end of invalid_fee.

  protected section.
  private section.
endclass.


class zcm_agency implementation.
endclass.
```

Esta estructura del grupo de constantes se obtiene de la interfaz **if_t100_message**, contiene el mensaje por defecto que se utilizará cuando se ejecute la clase de excepción y no se especifique alguno de los mensaje personalizados, vinculados a una clase de mensajes.

```
*"* components of interface IF_T100_MESSAGE
interface IF_T100_MESSAGE
  public .


  interfaces IF_MESSAGE .

  constants:
    begin of default_textid,
      msgid type symsgid value 'SY',
      msgno type symsgno value '530',
      attr1 type scx_attrname value '',
      attr2 type scx_attrname value '',
      attr3 type scx_attrname value '',
      attr4 type scx_attrname value '',
    end of default_textid.

  data T100KEY type SCX_T100KEY .
endinterface.
```

Definición del Método Constructor:

En la encapsulación pública, se define el método constructor, y agregando varios parámetros que se utilizarán para especificar la clase de mensaje (**textid** con el tipo **SCX_T100KEY,** de la variable **T100KEY,** de la interfaz **if_t100_message**), los atributos la severidad (**severity,** con el tipo de estructura **t_severity** de la interfaz **IF_ABAP_BEHV_MESSAGE.** Donde se puede establecer con los valores: **E** para error, **W** para advertencia, **I** para información, **S** para exitoso).

```
methods constructor
    importing
       textid   type SCX_T100KEY optional
       attr1    type string optional
       attr2    type string optional
       attr3    type string optional
       attr4    type string optional
       previous like previous optional
       severity type if_abap_behv_message=>t_severity optional.
```

Al implementar el método se generará el siguiente código.

```
class zcm_agency implementation.
  method constructor.

    super->constructor( textid = textid previous = previous ).

  endmethod.

endclass.
```

Donde el parámetro **PREVIOUS** se utiliza en la construcción de excepciones encadenadas. Permite mantener una referencia a una clase de excepción anterior que está definida a la herencia con la clase **CX_STATIC_CHECK** y está a su vez hereda de la clase **CX_ROOT.** Por lo que cuando se lanza una nueva excepción, es útil para rastrear la causa raíz de un error cuando se han producido múltiples excepciones a lo largo de un proceso.

```
class zcm_agency definition
  public
  inheriting from cx_static_check
  final
  create public .
```

```
class CX_STATIC_CHECK definition
  public
  inheriting from CX_ROOT
  abstract
  create public .

*"* public components of class CX_STATIC_CHECK
*"* do not include other source files here!!!
public section.

  methods CONSTRUCTOR
    importing
      !TEXTID like TEXTID optional
      !PREVIOUS like PREVIOUS optional .
protected section.
*"* protected components of class CX_STATIC_CHECK
*"* do not include other source files here!!!
private section.

  data START_AUTHORIZATION_FAILED type SOTR_CONC .
ENDCLASS.
```

```
class cx_root definition
  public
  abstract
  create public.

  public section.
    interfaces if_message.
    interfaces if_serializable_object.

    aliases get_longtext
      for if_message~get_longtext.
    aliases get_text
      for if_message~get_text.

    constants cx_root type sotr_conc value '16AA9A3937A9BB56E10000000A11447B'. "#EC NOTEXT
    data textid type sotr_conc read-only.
    data previous type ref to cx_root read-only.
    data kernel_errid type s380errid read-only.
    data is_resumable type abap_bool read-only.

    methods constructor
```

Luego es necesario declarar unas variables globales que permitirán por medio del constructor, vincular los parámetros incluidos en la instancia de la clase con los atributos dinámicos de la clase de mensajes especificados en el grupo de constantes.

```
private section.

data: mv_attr1    type string,
      mv_attr2    type string,
      mv_attr3    type string,
      mv_attr4    type string.

endclass.
```

En el constructor se vinculan las variables globales con los atributos del grupo de constantes que se desea utilizar para personalizar el mensaje, también se especifica la severidad o el tipo de mensaje a mostrar.

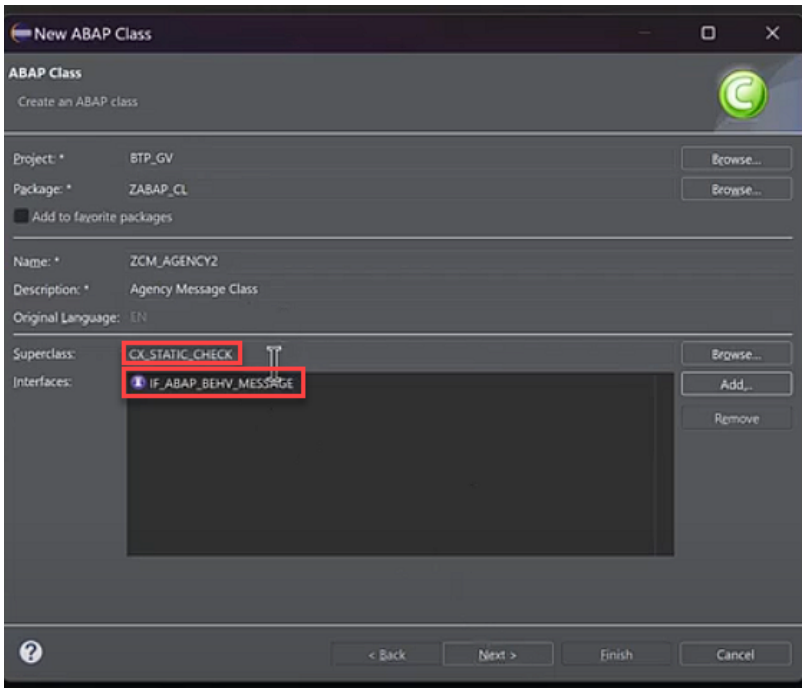
```
class zcm_agency implementation.
  method constructor.

    super->constructor( previous = previous ).

    me->mv_attr1 = attr1.
    me->mv_attr2 = attr2.
    me->mv_attr3 = attr3.
    me->mv_attr4 = attr4.

    if_abap_behv_message~m_severity = severity.

    clear me->textid.

  endmethod.
```

Luego se limpia la variable textid para su utilización. Esta variable permite establecer el mensaje a utilizar al especificar el grupo de constante con las propiedades del mensaje.

```
class zcm_agency implementation.
  method constructor ##ADT_SUPPRESS_GENERATION.

    super->constructor( previous = previous ).

    me->mv_attr1 = attr1.
    me->mv_attr2 = attr2.
    me->mv_attr3 = attr3.
    me->mv_attr4 = attr4.

    if_abap_behv_message~m_severity = severity.

    clear me->textid.

    if textid is initial.
      if_t100_message~t100key = if_t100_message=>default_textid.
    else.
      if_t100_message~t100key = textid.
    endif.
```

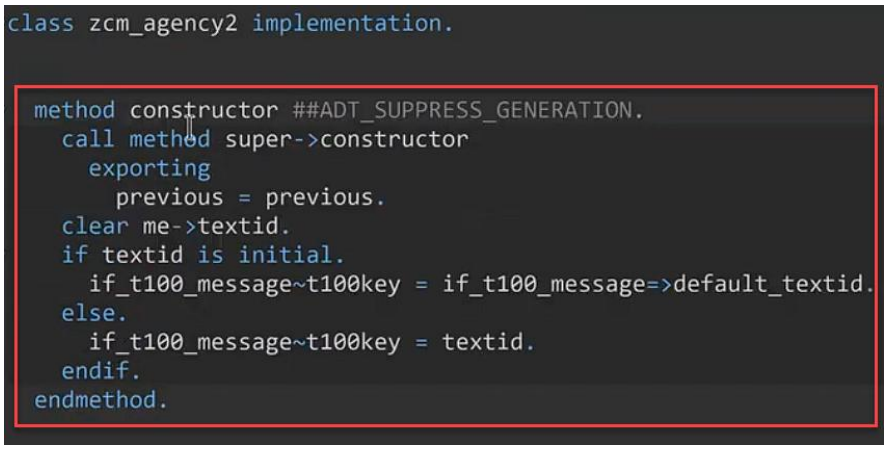
Una alternativa más rápida para la implementación de una clase de mensaje se realiza de la siguiente manera:

Al crear una nueva clase se coloca la clase **CX_STATIC_CHECK**, en el campo “Superclass”, de manera que se pueda generar en la clase creada una herencia directa a dicha clase de excepción principal. Además de Implementar la interfaz **IF_ABAP_BEHV_MESSAGE**.



De esta manera la clase generada tendrá además de la herencia requerida las interfaces que se deben utilizar y la creación del método constructor con la lógica implementada que se debe utilizar para vincular los mensajes de las clases de mensajes con la clase de excepción.

```
class zcm_agency2 definition
  public
  inheriting from cx_static_check
  final
  create public .

  public section.

    interfaces if_abap_behv_message .
    interfaces if_t100_message .
    interfaces if_t100_dyn_msg .

    methods constructor
      importing
        !textid   like if_t100_message=>t100key optional
        !previous like previous optional .
  protected section.
  private section.
endclass.
```



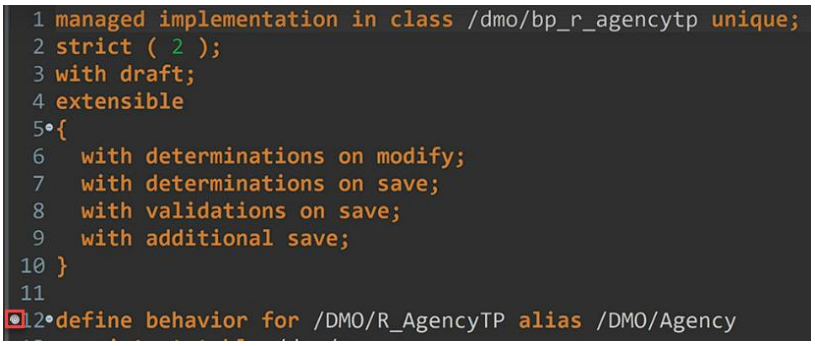
```
class zcm_agency2 implementation.

  method constructor ##ADT_SUPPRESS_GENERATION.
    call method super->constructor
      exporting
        previous = previous.
    clear me->textid.
    if textid is initial.
      if_t100_message~t100key = if_t100_message=>default_textid.
    else.
      if_t100_message~t100key = textid.
    endif.
  endmethod.
```

Existen métodos adicionales, como **get_text** y **get_longtext**, que pueden ser redefinidos en la clase de excepción a través de su generación. Sin embargo, es importante tener en cuenta que estos métodos no son necesarios y pueden ser eliminados si no se requieren. Para finalizar, se deben crear los elementos faltantes para que la clase de excepción sea funcional. Esto incluye definir un grupo de constantes con sus atributos para especificar los mensajes que aparecerán en la interfaz de usuario, así como declarar las variables globales y otros parámetros opcionales en el método constructor, entre otros.

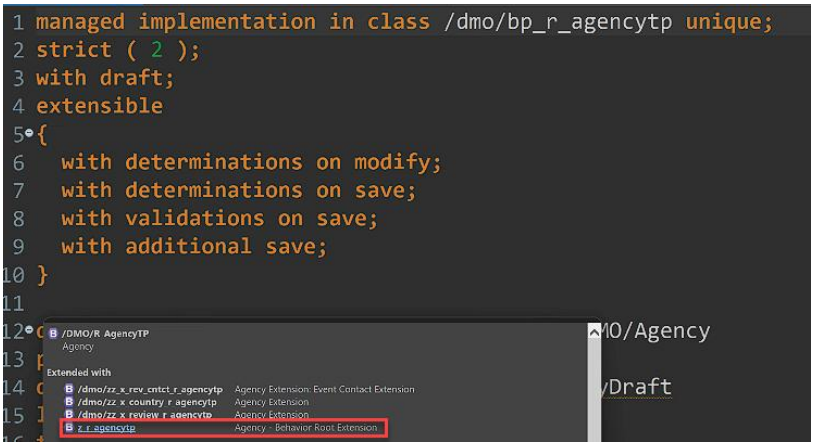
1.5. Behavior Definition Root – Extensión Validación

El proceso de Extensión de Validación en la Definición de Comportamiento Raíz (Behavior Definition Root) es un método utilizado en aplicaciones RAP estándar para añadir nuevas validaciones. Antes de aplicar una extensión, es crucial investigar si la aplicación admite dicho tipo de extensión. La implementación implica crear objetos específicos y definir campos a validar. Estas validaciones se realizaron para crear la extensión del Behavior Definition de la entidad raíz por medio de su interfaz para poder añadir características adicionales a la interfaz de usuario.

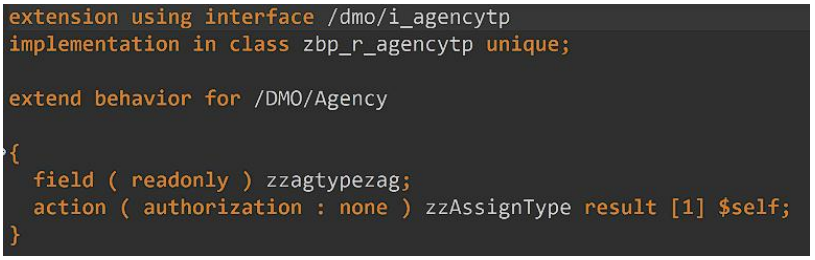
Se puede navegar a dicha extensión del Behavior Definition de la entidad raíz. Para hacerlo, selecciona el símbolo  , ubicado en la línea donde se encuentra la sentencia **define behavior for**. al desplegar el menú de relaciones.



```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}

define behavior for /DMO/R_AgencyTP alias /DMO/Agency
```



```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}
```



```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  field ( readonly ) zzagtypezag;
  action ( authorization : none ) zzAssignType result [1] $self;
}
```

Para incluir las validaciones y determinaciones correspondientes:

Validación:

Añadir una nueva validación sobre el campo especificado en la instrucción field al momento de guardar los cambios en la interfaz de usuario. Esto se realiza por medio de la sentencia:

validation validationName **on save {create; field** fieldname**;}.**

Determinación:

Añadir una determinación de acción para asegurar que esta validación también se ejecute durante el proceso de preparación del borrador,

manteniendo la integridad de los datos en ambas etapas. Esto se realiza por medio de la sentencia:

extend draft determine action DeterminationActionName

{validation validationName **;}**

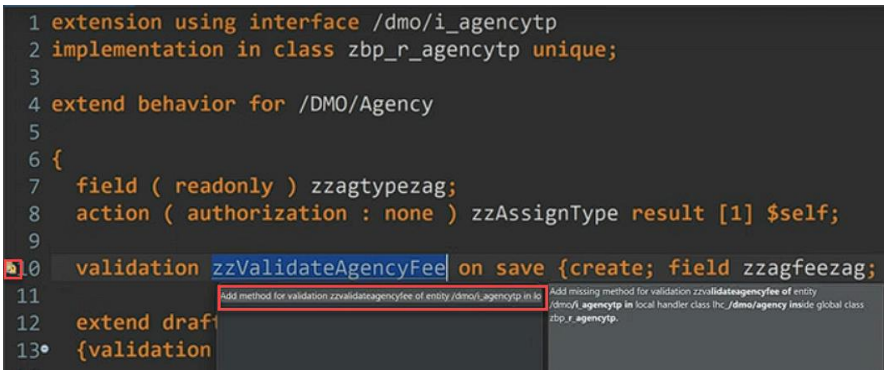
```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  field ( readonly ) zzagtypezag;
  action ( authorization : none ) zzAssignType result [1] $self;

  validation zzValidateAgencyFee on save {create; field zzagfeezag; }

  extend draft determine action Prepare
  {validation zzValidateAgencyFee;}

}
```

1.6. Behavior Pool – Implementación Validación

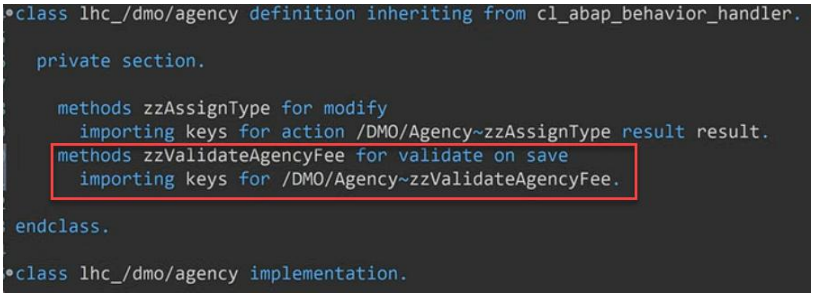
La implementación de validaciones en un Behavior Pool es un proceso clave en el desarrollo de aplicaciones RAP. Este proceso implica la creación y definición de métodos dentro de una clase de extensión de comportamiento, para asegurar que los datos ingresados cumplan con los criterios especificados.

Definición del Método:

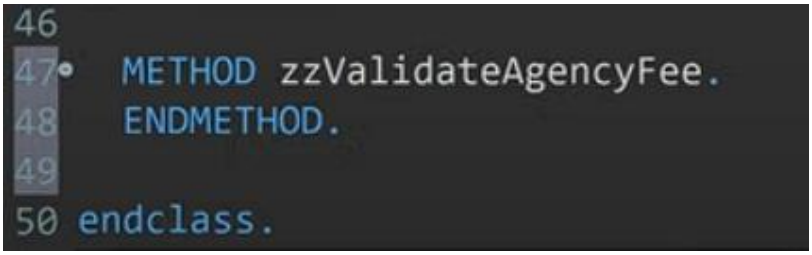
Al navegar a la extensión del Behavior Definition de la entidad raíz. Se puede generar la definición e implementación del método que contendrá la lógica de la acción de la validación. Para esto se debe presionar sobre el mensaje de advertencia para crear la clase por la opción propuesta por el editor. Luego seleccionar la opción “Add method for validation…” mostrada por el editor de código Eclipse para generar el método correspondiente. El nombre de la clase de extensión de comportamiento se puede ubicar fácilmente al lado de la sentencia **implementation in class.**



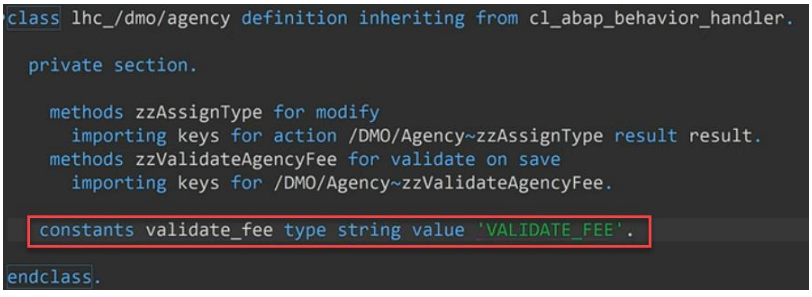
El método generado se agrega en la sección privada definición de la clase relacionada con la extensión de comportamiento.



Posteriormente agregar la lógica de la validación en la implementación del método.



Se agrega una constante para el uso de un texto en la lógica de la implementación.



Por lo cual en este caso, se realiza una lectura detallada de las entidades raíz y para el campo de cantidad seleccionado se realiza una validación para garantizar que se encuentren en un rango de 0 a 100, gestionando

los mensajes de error por medio de la estructura **failed** y el estado y tipo de mensaje en la interfaz de usuario por la estructura **reported.**

```
METHOD zzValidateAgencyFee.

    read entities of /dmo/i_agencytp in local mode
         entity /DMO/Agency
         fields ( zzagfeezag )
         with corresponding #( keys )
         result data(agenicies).

  loop at agenicies into data(agency).

    append value #( %tky        = agency-%tky
                    %state_area = validate_fee ) to reported-/dmo/agency.

    if not ( agency-zzagfeezag gt 0 and agency-zzagfeezag lt 100 ).

       append value #( %tky = agency-%tky ) to failed-/dmo/agency.

       append value #( %tky                = agency-%tky
                       %state_area         = validate_fee
                       %msg                = new zcm_agency( textid = zcm_agency=>invalid_fee )
                       %element-zzagfeezag = if_abap_behv=>mk-on ) to reported-/dmo/agency.

    endif.

  endloop.
```

En definitiva el objetivo de las validaciones es garantizar la integridad y fiabilidad de la aplicación, permitiendo a los usuarios identificar y corregir errores de manera efectiva.

1.7. Behavior Definition Root – Extensión Determinación

La extensión de determinación en la definición de comportamiento raíz es un proceso en el desarrollo de aplicaciones RAP que permite la configuración automática de campos en función de otros datos de entrada.

```
managed implementation in class /dmo/bp_r_agencytp unique;
strict ( 2 );
with draft;
extensible
{
  with determinations on modify;
  with determinations on save;
  with validations on save;
  with additional save;
}

define behavior for /DMO/R_AgencyTP alias /DMO/Agency
persistent table /dmo/agency
draft table /dmo/agency_d query /DMO/R_AgencyDraft
lock master
total etag LastChangedAt
authorization master ( global )
etag master LocalLastChangedAt
late numbering
extensible
```

En este caso, el objetivo sería determinar o completar automáticamente el valor del campo de Impuesto al Valor Añadido (VAT) basado en el código del país. Por lo que dicho campo debe estar

como no editable o de solo lectura para que este proceso sea realizado únicamente por la aplicación, por medio de la propiedad **readonly**.

Además para implementar esta determinación en la modificación del del código de país se utiliza la sentencia:

determination DeterminationName **on modify {create; field** FieldName**;}**

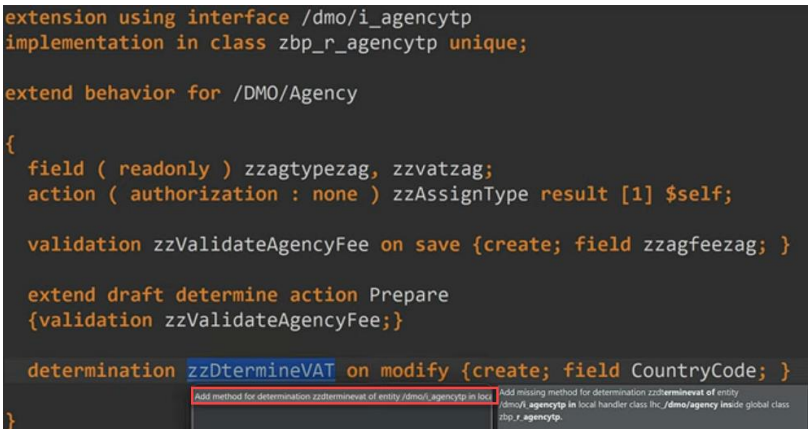
```
extension using interface /dmo/i_agencytp
implementation in class zbp_r_agencytp unique;

extend behavior for /DMO/Agency

{
  field ( readonly ) zzagtypezag, zzvatzag;
  action ( authorization : none ) zzAssignType result [1] $self;

  validation zzValidateAgencyFee on save {create; field zzagfeezag; }

  extend draft determine action Prepare
  {validation zzValidateAgencyFee;}

  determination zzDtermineVAT on modify {create; field CountryCode; }

}
```

La determinación automática de valores asegura que los datos sean precisos y consistentes, mejorando la eficiencia del sistema y facilitando la experiencia del usuario.

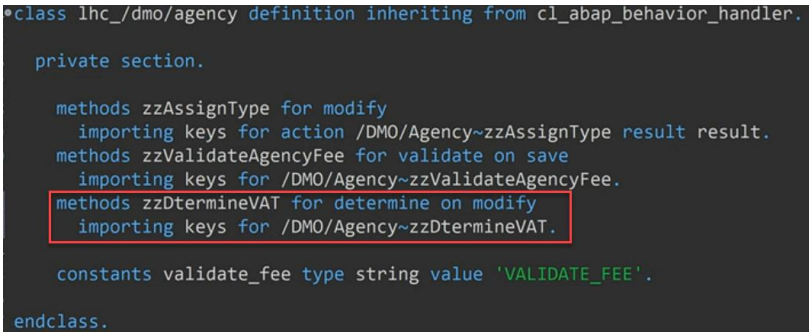
1.8. Behavior Pool – Implementación Determinación

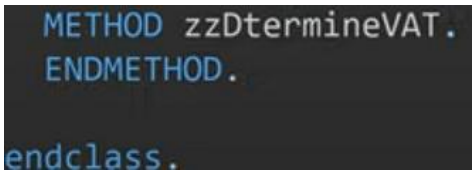
La implementación de determinaciones en un Behavior Pool de una aplicación RAP es un proceso que permite la asignación automática de valores a campos basados en otros datos de entrada. Por esta razón, la lógica de implementación de la determinación del campo VAT consiste en asignar un porcentaje de IVA específico basado en el país de la región en el campo CountryCode. De igual forma que la implementación de las validaciones se debe navegar a la clase de extensión del Behavior Definition de la entidad raíz. Para generar la definición e implementación del método que contendrá la lógica de la acción de la determinación.

Para esto se debe presionar sobre el mensaje de advertencia  para crear la clase por la opción propuesta por el editor. Luego seleccionar la opción “Add method for determination…” mostrada por el editor de código Eclipse para generar el método correspondiente.



Generando el siguiente método en la clase de extensión





Para implementar la lógica, se realiza una lectura detallada de las entidades raíz y para el campo de país CountryCode seleccionado se realiza una validación en donde en el caso de que el país sea Estados Unidos (US), el porcentaje de IVA será 15%. Para cualquier otro país, el porcentaje será 21%. Este proceso asegura que el valor del VAT se determine automáticamente en función del país seleccionado. Para luego modificar el valor del campo de iva en la interfaz de usuario por medio de una modificación a la entidad raíz.

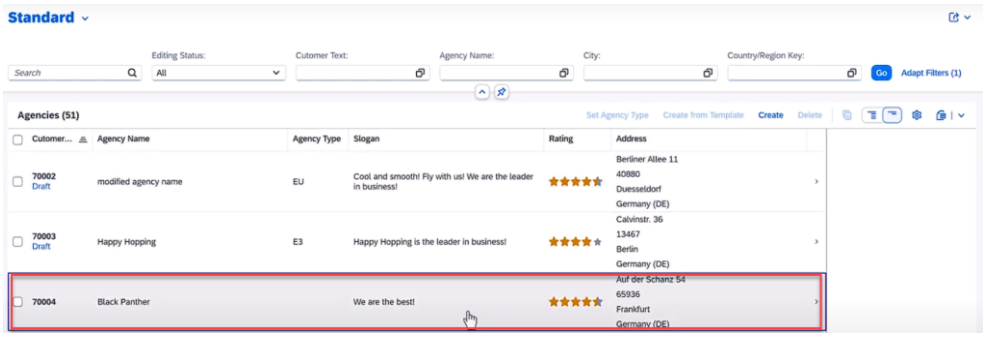
```
method zzDtermineVAT.

  data agencies_update type table for update /DMO/I_AgencyTP.

  read entities of /DMO/I_AgencyTP in local mode
     entity /DMO/Agency
     fields ( CountryCode zzvatzag )
     with corresponding #( keys )
     result data(agencies).

  loop at agencies into data(agency).

    if agency-CountryCode is not initial.

      append initial line to agencies_update assigning field-symbol(<agency_update>).
      <agency_update>-%tky = agency-%tky.

      case agency-CountryCode.
        when 'US'.
          agency-zzvatzag = 15.
        when others.
          agency-zzvatzag = 21.
      endcase.

    endif.

  endloop.

  modify entities of /DMO/I_AgencyTP in local mode
         entity /DMO/Agency
         update fields ( zzvatzag )
         with agencies_update.
```

La implementación asegura la integridad y precisión de los datos en la aplicación, facilitando la gestión eficiente del sistema.

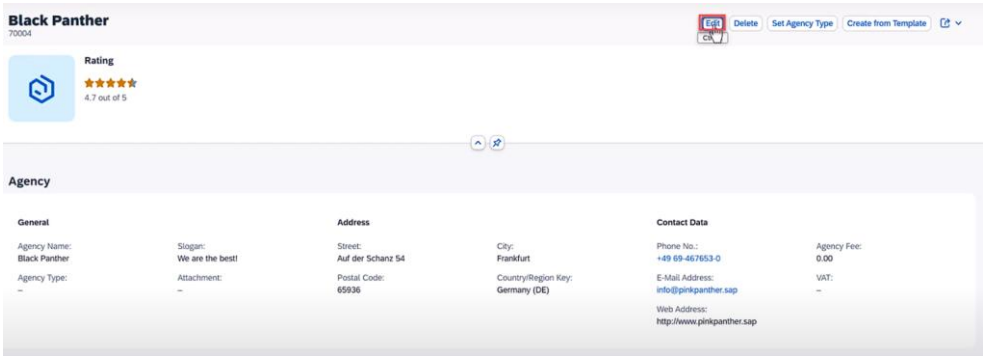
1.9. Prueba Requerimiento Implementado

El objetivo de esta prueba es asegurar que los nuevos campos, incluyendo el campo de IVA (VAT), se comporten según lo esperado al ser modificados. La validación y determinación se centran en verificar los valores ingresados y en asignar automáticamente los valores adecuados en función del país seleccionado.

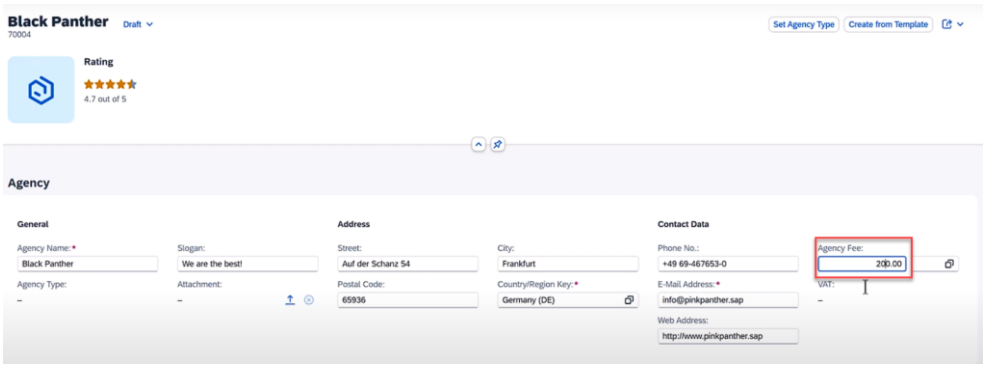
En primera instancia se debe seleccionar un registro dentro de la aplicación.



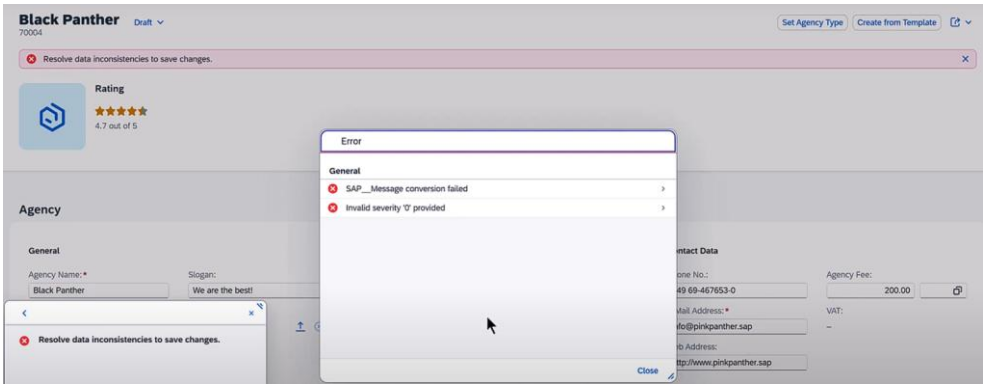
Para posteriormente editarlo.



Al ingresar un monto mayor de 100 en el campo Agency Fee, se puede comprobar que la validación funciona correctamente.



Aunque es necesario realizar un ajuste para agregar el tipo de mensaje. En este caso “error”, utilizando el parámetro severity y especificarlo.



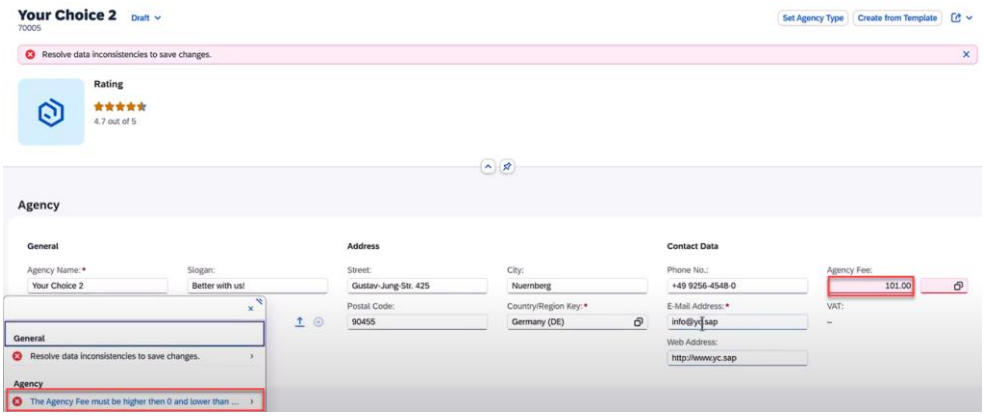
Al navegar a la clase de extensión del Behavior Definition de la entidad raíz para luego acceder al método que contiene la lógica de la validación.

```
class lhc_/dmo/agency definition inheriting from cl_abap_behavior_handler.

  private section.

    methods zzAssignType for modify
      importing keys for action /DMO/Agency~zzAssignType result result.
    methods zzValidateAgencyFee for validate on save
      importing keys for /DMO/Agency~zzValidateAgencyFee.
    methods zzDtermineVAT for determine on modify
      importing keys for /DMO/Agency~zzDtermineVAT.

    constants validate_fee type string value 'VALIDATE_FEE'.

endclass.
```

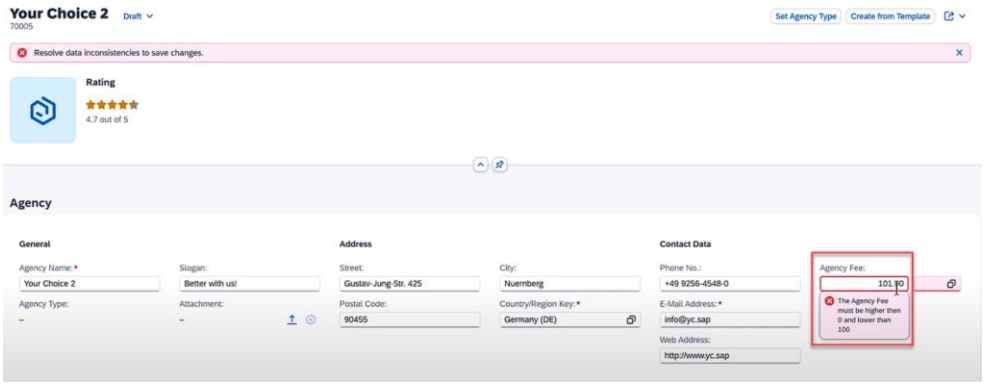
Para luego especificar en el parámetro severity el tipo de mensaje.

```
  loop at agenicies into data(agency).

    append value #( %tky        = agency-%tky
                    %state_area = validate_fee ) to reported-/dmo/agency.

    if not ( agency-zzagfeezag gt 0 and agency-zzagfeezag lt 100 ).

      append value #( %tky = agency-%tky ) to failed-/dmo/agency.

      append value #( %tky                = agency-%tky
                      %state_area         = validate_fee
                      %msg                = new zcm_agency( textid = zcm_agency=>invalid_fee
                                                            severity = if_abap_behv_message=>severity-error )
                      %element-zzagfeezag = if_abap_behv=>mk-on ) to reported-/dmo/agency.

    endif.

  endloop.

endmethod.
```

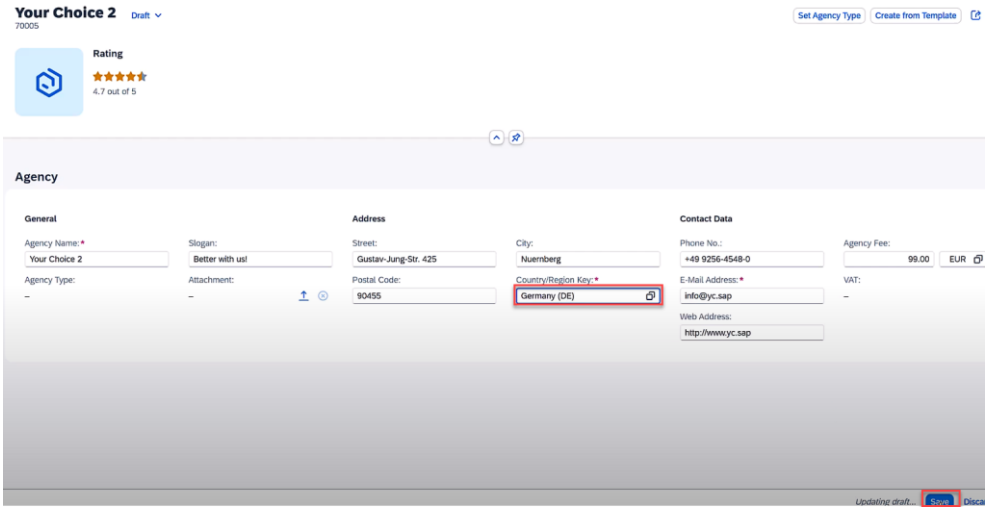
Y al ejecutar nuevamente se puede comprobar que el mensaje se muestra de forma correcta.



Al seleccionar el mensaje de la lista de errores en el menú en la parte inferior izquierda de la interfaz de usuario. El cursor es posicionado al campo con el error para su pronta corrección, además, que muestra el mensaje correspondiente.



Por otro lado al indicar un país.



Se puede comprobar que la determinación del campo no editable VAT se realiza correctamente.