How-to Guide
SAP NetWeaver CE 1.0

Cookbook for customer BAdIs in TPOS application

Version 1.00 – October 2008

Applicable Releases:
SAP NetWeaver CE 1.0

References

[1]. Installation and Configuration Guide for the Composite Application for Taxpayer Online Services from SAP

1. Introduction

1.1 Goal of the document

This cookbook is about how to implement own BAdIs for the Taxpayer Online Services from SAP (TPOS).

The configuration of the TPOS application is not part of this documentation (see the Configuration and Install Guide for TPOS 1.0). The understanding of this documentation assumes WebDynpro of Java knowledge.

1.2 Prerequisites

For developing custom BAdIs you need

- SAP Netweaver Studio (IDE) x.y version or upper
- TPOS ISCETAXBL.SCA

2. Prepare for implementation

2.1 SCA for interfaces

In the Development Infrastructure Perspective. You should create a new SCA (Software Component Archive) with the following attributes.

- Name: ISCETAXBL
- Vendor: sap.com
- Choose Archive SC radio button

Import the ISCETAXBLxy_z.sca into this SCA (xy_z describes the version of the current TPOS SCA file), which you can found in the TPOS package.

2.2 Crate own SCA and DC

Create you own SCA for implementing your own WebDynpro components. This SCA should have a dependency to the newly imported TPOS SCA.

You can create your own WebDynpro DC (Development Componnet) into this SCA.

Of course you can use previously created SCA or WebDynpro DC if you have already.

You should set up the dependencies also for this DCs:

- ISCETAXBL
  - the badi_if public part of is/cmp/etax/bl DC
  - the client public part of is/cmp/etax/bl DC
  - the config_provider public part of is/cmp/etax/bl DC
  - the badi_provider public part of is/cmp/etax/blapp DC
  - the umehelper_provider public part of is/cmp/etax/blapp DC for language determination of the logged in user **(!!! Needs to be added to the patch)**
  - the config_provider public part of is/cmp/etax/blapp DC for role determination of the logged in user

In our example we create an SCA named CUSTOMERBADI and a DC named customer/badi (EJB Module) and we set up the all dependencies mentioned above. We also create the DC names customerbadiapp (Enterpise Application) in the created SCA which has dependency set to customer/badi DC.

Into the SAP specific runtime deployment descriptor from the META-INF directory of customerbadiapp DC we add a hard reference to blapp DC:

```
<reference reference-type="hard">
      <reference-target target-type="application" provider-
name="sap.com">is~cmp~etax~blapp</reference-target>
</reference>
```

3. Implement custom BAdI

For an extension point we can have several BAdIs. If we have only one BAdI this will be the primary BAdI. If there are more BAdIs for an extension point the primary BAdI is a controller which handles the execution of the controlled BAdIs. In this chapter the implementation of primary BAdI, controlled BAdi and of the BAdI controller is presented.

A primary BAdI must implement one of the following interfaces:

- com.sap.is.cmp.etax.badi.IBAdI
- com.sap.is.cmp.etax.badi.IBAdIController

If the primary BAdI implements the IBAdIController interface it can act as a controller.

A controlled BAdI must implement the com.sap.is.cmp.etax.badi.IBAdI interface.

There are several BAdI types, implementing different interfaces. This document presents the implementation of an input validation BAdI and describes the differences in implementation of the other BAdI types.

3.1 Implement primary custom BAdI

In the customer/badi DC we create a stateless session bean which implements the com.sap.is.cmp.etax.badi.interfaces.IBAdIInputValidation interface as business interface which extends the IBAdI interface. The interface has a single method:

public void validateInput(IBAdIContext<ValidationInput, List<ValidationDescriptor>> context);

The IBAdIContext interface is the following:

```
* @param <BAdIInput>
 * @param <BAdIOutput>
 */
public interface IBAdIContext<BAdIInput, BAdIOutput> {

 /**
  * Retrieves the input of the BAdI.
  * It contains the current input value of the BAdI execution,
  * that can be the original value given by the standard code,
  * or the modified one if there were any preceding BAdI implementor
  * which modified it.
  *
  * @return input
  */
 public BAdIInput getInput();

 /**
  * Stores the input of the BAdI into the context.
  * It can be used when one implementor wishes to modify the input.
  *
  * @param input
  */
 public void setInput(BAdIInput input);
```

```
 /**
  * Retrieves the output of the BAdI
  * This is null when no implementor have set it yet,
  * or contains the current value of the output.
  *
  * @return output
  */
 public BAdIOutput getOutput();

 /**
  * Stores the output of the BAdI in the context.
  *
  * @param output
  */
 public void setOutput(BAdIOutput output);

 /**
  * Makes a proposal to terminate the BAdI execution.
  *
  * @param terminationProposal
  */
 public void setTerminationProposal(boolean terminationProposal);

 /**
  * Retrieves the name of the Extension point.
  *
  * @return extension point name
  */
 public String getExtensionPointName();
}
```

A simple example implementation for validation of bank name of the Taxpayer could look like:

```
@Local(value={IBAdIInputValidation.class})
@Stateless
public class BAdIInputValidationTaxPayerBankNameBean implements
        IBAdIInputValidation {

 public void validateInput(
              IBAdIContext<ValidationInput, List<ValidationDescriptor>> context) {
       ValidationInput input = context.getInput();
       List<ValidationDescriptor> output = new ArrayList<ValidationDescriptor>();
       try{
       ValidationAttribute validationAttribute = null;
       if(input != null){

                    Map<String,ValidationAttribute> map = input.getMap();
                    validationAttribute = map.get("bankName");

                    if(output.isEmpty() && Helper.isGiven(validationAttribute) &&
!Helper.getStringValue(validationAttribute).equals("Test Bank")){
                          ValidationDescriptor descriptor = new
ValidationDescriptor("bankName","Invalid Bank Name",Severity.ERROR);
                          output.add(descriptor);
                          context.setTerminationProposal(true);
                    }
       }
```

```
        } catch (Exception e){
              // handle exception
        }
        List<ValidationDescriptor> originalOutput = context.getOutput();
        if (originalOutput != null && !originalOutput.isEmpty()) {
              originalOutput.addAll(output);
        } else {
              context.setOutput(output);
        }
  }
```

The description of the used classes and interfaces can be found in **Table 1**

We have used a Helper class for checking if the validation attribute is available and for getting the value of the attribute.

The Termination Proposal handling can be configured, and it can have the values IGNORE, STOP, EXCEPTION. If in the context the Termination Proposal is set to true the BAdI framework will take care of the proper termination of the BAdI in case an error occurs.

For language determination of the validation messages the *String getLoggedUserLanguage()* method of the *com.sap.is.cmp.etax.util.UMEHelperBean* can be used which implements the *com.sap.is.cmp.etax.util.UMEHelperLocal* business interface.

3.1.1 Configure primary custom BAdI

In Development Infrastructure perspective the DCs we build the DCs and deploy the customerbadiapp DC.

With JNDI Browser we search for the newly implemented BAdI and we determine the application name, the bean name and the interface name we should enter for the Primary BAdI JNDI Name of the extension point we want to change. The configuration of primary custom BAdI is described in [1].

3.2 Implement a controlled BAdI

The implementation of the controlled BAdI is similar as the implementation of the primary BAdI.

3.2.1 Configure controlled BAdI

In Development Infrastructure perspective the DCs we build the DCs and deploy the customerbadiapp DC.

With JNDI Browser we search for the newly implemented BAdI and we determine the application name, the bean name and the interface name we should enter for the controlled BAdI JNDI Name of the extension point we want to change. In case of controlled BAdI a BAdI controller has to be provided, the default controller can be used or a custom controller can be implemented. The configuration of controlled BAdI is described in [1].

3.3 Implement a BAdI controller

The BAdI controller must implement the *com.sap.is.cmp.etax.badi.IBAdIController* interface which has the following definition:

```
* Interface representing a controller for BAdI implementations.
 * The method 'execute' is called by the framework with the appropriate controller
context.
```

```
 *
 * Any controller including custom ones must implement this interface.
 *
 * @param <BAdIInput>
 * @param <BAdIOutput>
 */
public interface IBAdIController<BAdIInput, BAdIOutput> {

 /**
  * Executes the controller logic
  *
  * @param context
  * @throws Exception
  */
 public void execute(ControllerContext<BAdIInput, BAdIOutput> context)
       throws BAdITerminationException, BAdIException;

}
```

The *ControllerContext<BAdIInput, BAdIOutput>* class is the context of the controllers, it contains the execution context and the list of controlled BAdIs and provides the following methods:

- *int sizeOfImplementationList()* – provides the number of controlled BAdIs
- *void executeNextImplementation()* – executes the next controlled BAdI, throws BAdITerminationException and BAdIException
- *void executeNextImplementation(ExecutionContext<BAdIInput, BAdIOutput> eCtx)* – executes the next controlled BAdI with the provided execution context, throws BAdITerminationException and BAdIException
- *boolean hasNextImplementation()* – returns true if the extension point has a next implementation
- *ExecutionContext<BAdIInput, BAdIOutput> getExecutionContext()* – returns the ExecutionContext

The *ExecutionContext<BAdIInput, BAdIOutput>* class implements the *IBAdIContext<BAdIInput, BAdIOutput>* interface described earlier and contains the execution context of the BAdI implementations.

The implementation of the default controller is the following:

```
* @param <BAdIInput>
 * @param <BAdIOutput>
 */
@Stateless(name="RelayIterator")
@Local(IBAdIController.class)
public class RelayIterator<BAdIInput, BAdIOutput> implements
IBAdIController<BAdIInput, BAdIOutput> {

 public void execute(ControllerContext<BAdIInput, BAdIOutput> context)
       throws BAdITerminationException, BAdIException {

       while(context.hasNextImplementation()) {
             context.executeNextImplementation();
       }
 }
```

```
}
```

For implementation of a custom controller create a stateless session bean in customer/badi DC which implements the IBAdIController interface. For example for input validation BAdIs we could have a controller which ends the execution of the controlled BAdIs when a validation problem occurs:

```
@Stateless(name="RelayIteratorInputValidationTaxPayerBankDataBean")
@Local(IBAdIController.class)
public class RelayIteratorInputValidationTaxPayerBankDataBean implements
 IBAdIController<ValidationInput, List<ValidationDescriptor>> {

 public void execute(ControllerContext<ValidationInput,
List<ValidationDescriptor>> context)
 throws BAdITerminationException, BAdIException {

 while(context.hasNextImplementation()) {
       context.executeNextImplementation();
       if (context.getExecutionContext().getOutput().size() > 0){
             break;
       }
 }
}


}
```

3.3.1 Configure custom BAdI controller

In Development Infrastructure perspective the DCs we build the DCs and deploy the customerbadiapp DC.

With JNDI Browser we search for the newly implemented custom controller and we determine the application name, the bean name and the interface name we should enter for the primary BAdI JNDI Name which will be the controller BAdI for the extension point. The configuration of custom BAdI controller is described in [1].

3.4 Interfaces to be implemented by different BAdI types

The interfaces to be implemented are in the *com.sap.is.cmp.etax.bl.badi.interfaces* package.

BAdI Type	Interface to be implemented	Method to be implemented, used interfaces and classes
Input validation	IBAdIInputValidation	void validateInput(IBAdIContext<ValidationInput, List<ValidationDescriptor>> context) The used classes are: • *com.sap.is.cmp.etax.bl.badi.interfaces.ValidationInput* class which provides the following methods: ○ Map<String,ValidationAttribute> getMap() **!!! we have to provide the keys for each validation attribute** ○ String getLoggedUser() ○ RoleData getLoggedUserRole() ○ ValidationInput(Map<String,ValidationAttribute > map, String loggedUser,RoleData loggedUserRole) • *com.sap.is.cmp.etax.bl.badi.interfaces.ValidationDescriptor* class which provides the following methods: ○ String getKey()

		○ String getMessage() ○ Severity getSeverity() –,the Severity is an enum with values INFO, WARNING, ERROR ○ ValidationDescriptor(String key, String message, Severity severity) • *com.sap.is.cmp.etax.bl.badi.interfaces.ValidationAttribute<Type>* class which implements the Serializable interface, always has the type of the attribute which has to be validated and provides the following methods: ○ String getKey() ○ Type getValue() ○ ValidationAttribute(String key, Type value) • UmeHelper **(!!! The dependency has to be added)** • *RoleData* class which provides the following methods: ○ RoleData(EtaxRoles role, String umeRole, String description) where EtaxRoles is an enum class with the following values INDIVIDUAL_TAXPAYER ("ITP"), TAX_ADVISER ("TAD"), TAX_ACCOUNTANT ("TAC") and providing the String getName() method • *com.sap.is.cmp.etax.bl.badi.interfaces. ValidationAttributeKey* class which provides the following methods: ○ ValidationAttributeKey(String key) ○ String getKey()
Field class provider	IBAdiFiledExtClassProvider	void getExtendClass(IBAdIContext<FieldExtInput, FieldExtOutput> context) The used classes are: • *com.sap.is.cmp.etax.bl.badi.interfaces.FieldExtInput* class which provides the following methods: ○ String getWdModelClassName() ○ void setWdModelClassName(String wdModelClassName) • *com.sap.is.cmp.etax.bl.badi.interfaces.FieldExtOutput* class which provides the following methods: ○ Map<String, Class<?>> getEnhancementMap() ○ void setEnhancementMap(Map<String, Class<?>> enhancementMap)
PBO filing	IBAdiProcessBeforeOutput	void processAfterInput(com.sap.is.cmp.etax.badi.IBAdIContext<PBOInput,PBOOutput> context) throws BAdITerminationException The used classes are: • *com.sap.is.cmp.etax.bl.badi.interfaces.PBOInput* extends the *ProcessInput* class which provides the following methods: ○ String getAction() ○ void setAction(String action) ○ String getLoggedUser() ○ void setLoggedUser(String loggedUser) ○ RoleData getLoggedUserRole() ○ void setLoggedUserRole(RoleData loggedUserRole)

		○ String getBpId() ○ void setBpId(String bpId) ○ Map<String, Object> getAdditionalValues() ○ void setAdditionalValues(Map<String, Object> additionalValues) ○ Map<String, FieldDescription> getFields() ○ void setFields(Map<String, FieldDescription> fields) ○ List<MessageDescription> getMessages() ○ void setMessages(List<MessageDescription> messages) ○ String getLanguageCode() ○ void setLanguageCode(String languageCode) • *com.sap.is.cmp.etax.bl.badi.interfaces.PBOOutput* extends the *ProcessOutput* class which provides the following methods: ○ boolean isStopProcess() ○ void setStopProcess(boolean stopProcess) ○ Map<String, ? extends FieldValue> getFields() ○ void setFields(Map<String, ? extends FieldValue> fields) ○ List<MessageDescription> getMessages() ○ void setMessages(List<MessageDescription> messages) • *com.sap.is.cmp.etax.configuration.rolemanagement.RoleData* class which provides the methods described for input validation BAdIs • *com.sap.is.cmp.etax.bl.badi.interfaces.FieldDescription* class extends the *FieldValue* class and provides the following methods: ○ FieldDescription(String fieldId, Object value, Class valueClass ,Set valueSet) ○ Class getValueClass() ○ void setValueClass(Class valueClass) ○ Set getValueSet() ○ void setValueSet(Set valueSet) • *com.sap.is.cmp.etax.bl.badi.interfaces.FieldValue* class provides the following methods: ○ FieldValue(String fieldId, Object value) ○ String getFieldId() ○ void setFieldId(String fieldId) ○ Object getValue() ○ void setValue(Object value) • *com.sap.is.cmp.etax.bl.badi.interfaces.MessageDescription* class which provides the following methods: ○ String getMessage() ○ void setMessage(String message) ○ String getFieldId() ○ void setFieldId(String fieldId) ○ boolean isHighlight() ○ void setHighlight(boolean highlight) ○ SeverityType getSeverity() ○ void setSeverity(SeverityType severity) the SeverityType is an enum with the following values: SUCESS, WARRNING, ERROR, INFO
TRM form scenario	IBAdITrmFormScenario	void findTrmFormScenarioByElement(IBAdIContext<TrmFormScenarioInput, TrmFormScenarioOutput> context)

		The used classes are: • *com.sap.is.cmp.etax.bl.badi.interfaces.TrmFormScenarioInput* class provides the following method: ○ TrmFormScenarioInput(Object request) • *com.sap.is.cmp.etax.bl.badi.interfaces.TrmFormScenarioOutput*

TRM return prefill	IBAdITrmReturnPrefill	void prefillTrmReturn(IBAdIContext<ReturnDTO, ReturnDTO> context)
		
		The used classes are:
		• *com.sap.is.cmp.etax.tr.ReturnDTO* class extends the *com.sap.is.cmp.etax.tr.BaseListDTO* class, implements the *com.sap.is.cmp.etax.tr.IReturn* `and` *com.sap.is.cmp.etax.tr.request.Request* interfaces and provides the following methods:
		  ○ public String getSupplementId()
		  ○ void setSupplementId(String supplementId)
		  ○ Calendar getSupplementDate()
		  ○ void setSupplementDate(Calendar supplementDate)
		  ○ String getAssessmentId()
		  ○ void setAssessmentId(String assessmentId)
		  ○ Calendar getAssessmentDate()
		  ○ void setAssessmentDate(Calendar assessmentDate)
		  ○ Calendar getApplyFrom()
		  ○ void setApplyFrom(Calendar applyFrom)
		  ○ String getBusinessPartner()
		  ○ void setBusinessPartner(String businessPartner)
		  ○ String getContainerNum()
		  ○ void setContainerNum(String containerNum)
		  ○ String getContractAccount()
		  ○ void setContractAccount(String contractAccount)
		  ○ List<FormDataDTO> getFormData()
		  ○ void setFormData(List<FormDataDTO> formData)
		  ○ String getPostmark()
		  ○ void setPostmark(String postmark)
		  ○ Calendar getReceipt()
		  ○ void setReceipt(Calendar receipt)
		  ○ String getRegistrationID()
		  ○ void setRegistrationID(String registrationID)
		  ○ String getRegistrationType()
		  ○ void setRegistrationType(String registrationType)
		  ○ String getReturnID()
		  ○ void setReturnID(String returnID)
		  ○ Calendar getTimeStamp()
		  ○ void setTimeStamp(Calendar timeStamp)
		  ○ String getValidation()
		  ○ void setValidation(String validation)
		  ○ boolean getXChanged()
		  ○ void setXChanged(boolean changed)
		  ○ ScenarioDTO getScenario()
		  ○ void setScenario(ScenarioDTO scenario)
		  ○ boolean isDraft()
		  ○ void setDraft(boolean draft)
		  ○ String getType()

		○ void setType(String type) ○ ReturnDTO() ○ ReturnDTO(String returnID, String businessPartner, String registrationID, String contractAccount, ScenarioDTO scenario, Calendar applyFrom, String registrationType, List<FormDataDTO> formData, String containerNum, boolean changed, String validation, String postmark, Calendar receipt, Calendar timeStamp, String supplementId, Calendar supplementDate, String assessmentId, Calendar assessmentDate, boolean draft, String type) ○ ReturnDTO(IReturn iReturn) ○ List<FormDataDTO> getUiFormData() ○ void setUiFormData(List<FormDataDTO> uiFormData) ○ List<String> getReadOnlyField() ○ void setReadOnlyField(List<String> readOnlyField) ○ void validate() throws InvalidInputException ○ String toString() • *com.sap.is.cmp.etax.tr.BaseListDTO* class provides the following methods: ○ boolean isStandard() ○ void setStandard(boolean standard) ○ boolean isUpdate() ○ void setUpdate(boolean update) • *com.sap.is.cmp.etax.tr.FormDataDTO<E extends Serializable, F extends Serializable>* class extends the *com.sap.is.cmp.etax.tr.ValueHelpDTO* and provides the following methods: ○ FormDataDTO(E key, F value) ○ FormDataDTO() ○ FormDataDTO(E key, F value, boolean standard, boolean update) ○ String toString() • *com.sap.is.cmp.etax.tr.ValueHelpDTO <E extends Serializable, F extends Serializable>* class extends the *com.sap.is.cmp.etax.tr.BaseListDTO* class and provides the following methods: ○ ValueHelpDTO(E key, F value) ○ E getKey() ○ void setKey(E key) ○ F getValue() ○ void setValue(F value) ○ String toString() • *com.sap.is.cmp.etax.tr.ScenarioDTO* class extends the *com.sap.is.cmp.etax.tr.BaseListDTO* class and provides the following methods: ○ String getPeriod() ○ void setPeriod(String period) ○ String getRevenueType() ○ void setRevenueType(String revenueType) ○ String getScenario() ○ void setScenario(String scenario) ○ String getScenarioVersion() ○ void setScenarioVersion(String scenarioVersion) ○ String getLanguage()

		○ void setLanguage(String language) ○ String getAmountFieldName() ○ void setAmountFieldName(String amountFieldName) ○ String getRevenueTypeName() ○ void setRevenueTypeName(String revenueTypeName) ○ String getPeriodName() ○ void setPeriodName(String periodName) ○ String toString() ○ void validate() throws InvalidInputException
UIPrefillPaymentOnAccount	IBAdIUIPrefill	void prefillInput(IBAdIContext<PrefillInput, PrefillOutput> context) The used classes are: • *com.sap.is.cmp.etax.bl.badi.interfaces.PrefillInput* class provides the following methods: ○ String getBpId() ○ void setBpId(String bpId) ○ Map<String,PrefillAttribute> getMap() ○ PrefillInput() ○ PrefillInput(Map<String,PrefillAttribute> map, String action, Object data,String bpId) ○ Object getData() ○ void setData(Object data) ○ String getAction() ○ void setAction(String action) ○ void setMap(Map<String, PrefillAttribute> map) • *com.sap.is.cmp.etax.bl.badi.interfaces. PrefillOutput* class provides the following methods: ○ Map<String,PrefillAttribute> getMap() ○ PrefillOutput() ○ PrefillOutput(Map<String,PrefillAttribute> map) ○ void setMap(Map<String, PrefillAttribute> map) • *com.sap.is.cmp.etax.bl.badi.interfaces.PrefillAttribute* class provides the following methods: ○ String getKey() ○ Object getValue() ○ PrefillAttribute(String key, String fieldId,Object value, Class valueType)

Table 1 Classes and interfaces used for custom BAdI implementation