How-to Guide
Taxpayer Online Services from SAP 2.0

Cookbook for Implementing BC switch in TPOS 2.0 application

Version 1.00 – August 2010

Applicable Releases:
Taxpayer Online Services from SAP 2.0

Copyright

1. INTRODUCTION **2**
1.1 GOAL OF THE DOCUMENT 2
1.2 PREREQUISITES 2
2. PREPARE FOR IMPLEMENTATION **3**
2.1 SCA FOR INTERFACES 3
2.2 CREATE OWN SC AND DC 3
3. IMPLEMENT THE NEW BC BEAN **4**
3.1 THE BASE INTERFACES 4
3.1.1 The gen interface *4*
3.1.2 The business interface *4*
3.2 THE EXTERNALSERVICE CLASS 5
3.2.1 Declaration *5*
3.2.2 Parameters *5*
3.2.3 The execute method *5*
3.2.4 Abstract methods *6*
3.3 CREATE A NEW EJB SESSIONBEAN 7
3.3.1 In your EJB Module DC create a new EJB SessionBean. *7*
3.3.2 Add methods to the class *7*
3.3.3 Deploy your SCA into the CE Engine *8*
3.4 CREATE THE CONFIGURATION ENTRY 8
3.4.1 Determine the JNDI name of the bean *9*
3.4.2 Enter the new configuration key *10*

1. Introduction

1.1 Goal of the document

This cookbook is about how to implement a brand-new Business Connectivity (BC) EJB beans for Taxpayer Online Service from SAP 2.0 and re-direct an existing Business Layer (BL) call from the SAP provided BC to the customer BC. This procedure is called “BC switch”.

The configuration of the TPOS application is not part of this documentation (see the Configuration and Install Guide for TPOS 2.0). The understanding of this documentation assumes TPOS architecture (i.e. Enterprise Services) knowledge.

1.2 Prerequisites

For developing BC extension beans you need

- SAP Netweaver Studio (IDE) 7.11 SP04 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.

Define the Attributes for the SCA:

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 among the TPOS software archives.

2.2 Create own SC and DC

Create you own SC for implementing your own BC extension beans. This SCA should have a dependency to the newly imported TPOS SCAs. You can create your own Enterprise Application DC (Development Component) and EJB Module DC into this SCA. Of course you can use previously created SCA or Enterprise Application DC and EJB Module DC if you have already.

You should set up the dependencies also for this DCs:

ISCETAXBL

is/cmp/etax/bl **external_service** Public Part (PP) this describes the base ExternalService interface (see later)

is/cmp/etax/bl **client** PP this describes the TPOS DTOs (see later)

is/cmp/etax/exceptions **exceptions** PP

ENGFACADE

This SCA provided by the IDE

tc/bl/logging/api **api** PP for logging (used by the ExternalService)

In our example we create an SCA named BCSWITCH (vendor: com.cust) and Enterprise Application DC named bcsw_app, EJB Module DC named bcsw and we set up the all dependencies mentioned above.

Also a runtime reference has to be added to the EJB Application DC's application-j2ee-engine.xml what points to the TPOS BC App DC.

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

For example:

```
<application-j2ee-engine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="application-j2ee-engine.xsd">
<reference reference-type="hard">
      <reference-target target-type="application" provider-
name="sap.com">is~cmp~etax~blapp</reference-target>
</reference>
</application-j2ee-engine>
```

3. Implement the new BC bean

All BC beans that are shipped with TPOS extends ExternalService class. It is advised the you also derive your own BC bean from this class.

Also, the BC beans implement an EJB Local interface that are defined in the is/cmp/etax/bl. The communication is done between the BL-BC via these business interfaces.

In order to create a BC extension bean, a new EJB bean has to be created that must implement the proper BC extension interface what is defined in the ISCETAXBC.

3.1 The base interfaces

3.1.1 The gen interface

This interfaces are defined in the TPOS BL.

The execute method defined here the generic “technical” version of the business

```
public interface IExternalService<DTOIn, DTOOut> {

      IServiceResponse<DTOOut> execute(DTOIn inputDTO)
            throws AmbiguousUserAdministrationException, ResourceProblemException,
BCException;

}
```

3.1.2 The business interface

The BL calls the BC via the business interfaces where the types are defined according the business requirements.

For example: the business interface for creating payment card is:

```
public interface ICreatePaymentCard {
      public ServiceResponse<CreditCardDTO> execute(CreditCardDTO paymentCard)
      throws AmbiguousUserAdministrationException, ResourceProblemException,
BCException;
}
```

3.2 The ExternalService class

The ExternalService class implements the IExternalService interface BL calls the BC via the business interfaces where the types are defined according the business requirements.

3.2.1 Declaration

```
public abstract class ExternalService<DTOIn, DTOOut, ServiceIn, ServiceOut> implements
IExternalService<DTOIn, DTOOut> {
```

3.2.2 Parameters

- DTOIn: TPOS specific DTO that contains input data
- DTOOut: TPOS specific DTO that contains output data
- ServiceIn: Generated proxy class
- ServiceOut: Generated proxy class

3.2.3 The execute method

The execute() method controls the external service calls step-by-step. However your task will be add the business logic to the BC bean what is done via the abscract methods declared in the class.

```
      public ServiceResponse<DTOOut> execute(DTOIn inputDTO) throws
AmbiguousUserAdministrationException, ResourceProblemException, BCException {
            SERV_NAME = getSERV_NAME();
            ServiceIn serviceIn = null;
            ServiceOut serviceOut = null;
            ServiceResponse<DTOOut> serviceResponse  = null;

            String method = "execute";
            SimpleLoggerDecorator.entering(method, location, null);

            try {

                  String loggingString = null;

                  //** Preparing the service input
                  dtoIn = inputDTO;
                  if (dtoIn != null){
                        loggingString = dtoIn.toString();
                  }
```

```
                  SimpleLoggerDecorator.trace(Severity.DEBUG, location, "The
processInput() method for service {0} was called with parameters {1}", new Object[]
{SERV_NAME, loggingString});
                  serviceIn = this.processInput(inputDTO);
                  //** calling service
                  SimpleLoggerDecorator.log(Severity.INFO, location, "The
callService() method for service {0} was called", new Object[] {SERV_NAME});
                  serviceOut = this.callService(serviceIn);
                  //** processing the service output
                  SimpleLoggerDecorator.trace(Severity.DEBUG, location, "The
processOutput() method for service {0} was called", new Object[] {SERV_NAME});
                  DTOOut dtoOut = this.processOutput(serviceOut);
                  if (dtoOut != null){
                        loggingString = dtoOut.toString();
                  }
                  SimpleLoggerDecorator.trace(Severity.DEBUG, location, "The  results
of the processOutput() method for service {0} are {1}", new Object[] {SERV_NAME,
loggingString});
                  //** populating the GDT log.
                  SimpleLoggerDecorator.trace(Severity.DEBUG, location, "The
populateGDTLog() method for service {0} was called", new Object[] {SERV_NAME});
                  GDTLog log = populateGDTLog(serviceOut);
                  //** creating the response
                  serviceResponse = new ServiceResponse<DTOOut>(dtoOut,log);

                  return serviceResponse;

                  } catch (RuntimeException rt) {
                        throw new BCException("Error in External Service
('"+SERV_NAME+"') processing",rt);
                  } finally {
                        SimpleLoggerDecorator.exiting(location);
                  }

      }
```

3.2.4 Abstract methods

- `protected abstract ServiceIn processInput(DTOIn dtoIn) throws ResourceProblemException, AmbiguousUserAdministrationException, BCException;`

  This method is used for map the input DTO to service input.

- `protected abstract DTOOut processOutput(ServiceOut serviceOut) throws BCException;`

  This method is used for map the service result to DTO.

- `protected abstract ServiceOut callService(ServiceIn serviceIn) throws AmbiguousUserAdministrationException, ResourceProblemException, BCException;`

This method is used for call the service.

- `protected abstract String getSERV_NAME();`

  This method is used for return the name of the service. The value is used for logging.

- `protected abstract GDTLog populateGDTLog(ServiceOut serviceOut) throws BCException;`

  This method is used for error handling.

3.3 Create a new EJB SessionBean

3.3.1 In your EJB Module DC create a new EJB SessionBean.

In our example we are going to create a new BC bean for creating payment card. The corresponding business interface is the ICreatePaymantCard. Therefore, the dtoIn and dtoOut has type CreditCardDTO. The service related serviceIn and serviseOut now are only mock types in this example.

```
package com.cust.bcsw;

import javax.ejb.Stateless;
import com.cust.bcsw.mock.MockWSInput;
import com.cust.bcsw.mock.MockWSOutput;
import com.sap.is.cmp.etax.bc.iface.pcard.ICreatePaymentCard;
import com.sap.is.cmp.etax.exceptions.AmbiguousUserAdministrationException;
import com.sap.is.cmp.etax.exceptions.BCException;
import com.sap.is.cmp.etax.exceptions.ResourceProblemException;
import com.sap.is.cmp.etax.tr.CreditCardDTO;
import com.sap.is.cmp.etax.tr.GDTLog;
import com.sap.is.cmp.etax.tr.externalservice.ExternalService;


@Local(value = { ICreatePaymentCard.class })
@Stateless
public class NewBCean
      extends
      ExternalService<
            CreditCardDTO,
            CreditCardDTO,
            MockWSInput,
            MockWSOutput>
      implements ICreatePaymentCard {

}
```

3.3.2 Add methods to the class

For example: the extension of CreatePaymentCardManagerBean would look like:

```
..
@Override
protected MockWSOutput callService(MockWSInput serviceIn)
            throws AmbiguousUserAdministrationException,
            ResourceProblemException, BCException {
      // generated method stub

      ...
}

@Override
protected String getSERV_NAME() {
      // generated method stub

      ...
}

@Override
protected GDTLog populateGDTLog(MockWSOutput serviceOut) throws BCException {
      // generated method stub

      ...
}

@Override
protected MockWSInput processInput(CreditCardDTO dtoIn)
            throws ResourceProblemException,
            AmbiguousUserAdministrationException, BCException {
      // generated method stub

      ...
}

@Override
protected CreditCardDTO processOutput(MockWSOutput serviceOut)
            throws BCException {
      // generated method stub

      ...
}

...
```

3.3.3 Deploy your SCA into the CE Engine

If the implementation is done, deploy the EJB Application (e.g. bcsw_app) DC into the engine.

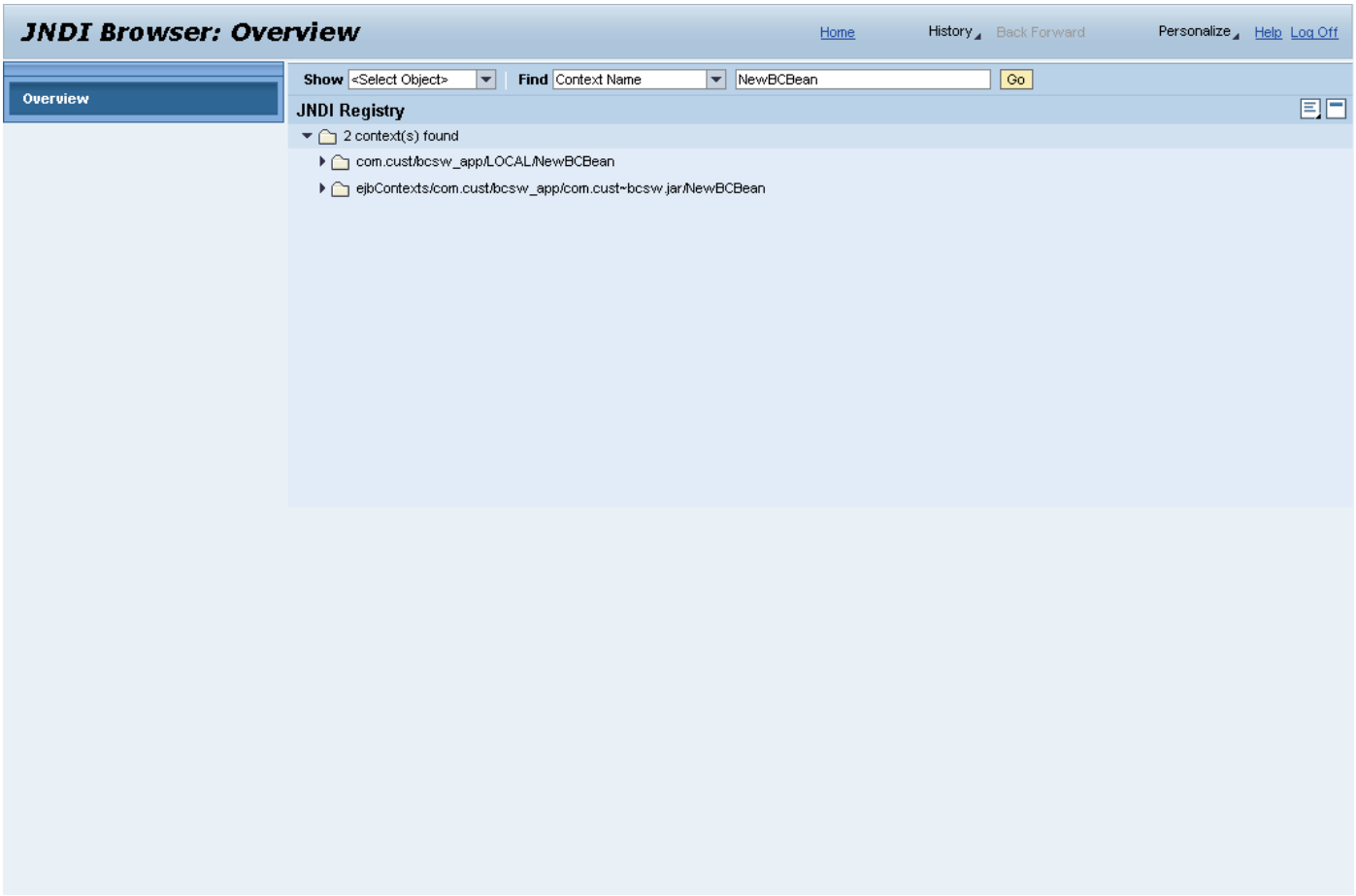
3.4 Create the configuration entry

When the new BC bean is done, you have to inform the TPOS framework that there is a new BC bean has to be called instead the SAP provided one. This is done with a TPOS AdminUI

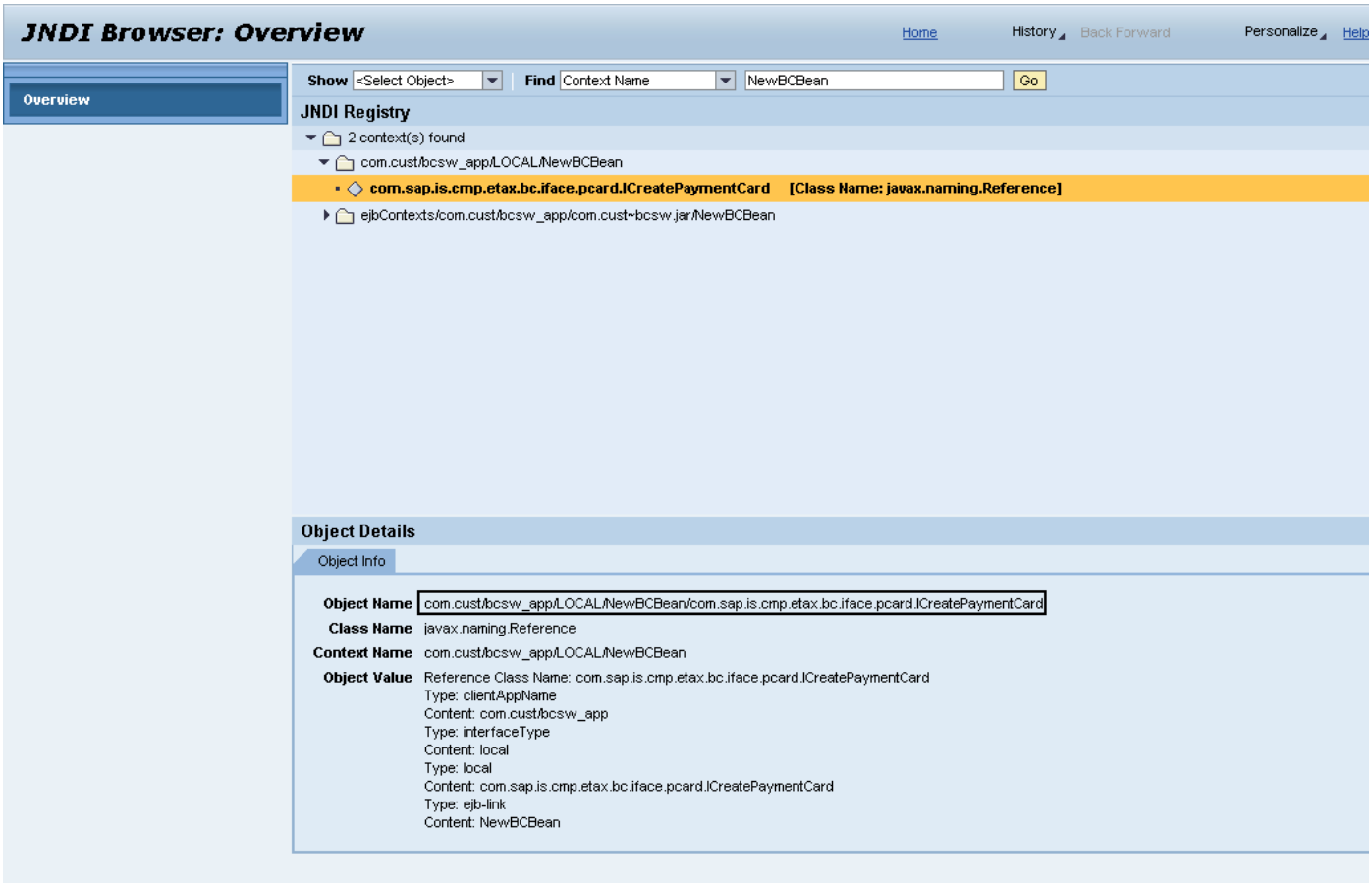
3.4.1 Determine the JNDI name of the bean

You can use the CE NWA -> Problem Management -> JNDI Browser to determine a proper JNDI name.

Step 1: enter the Context Name <name of the bean>. Press “Go”.



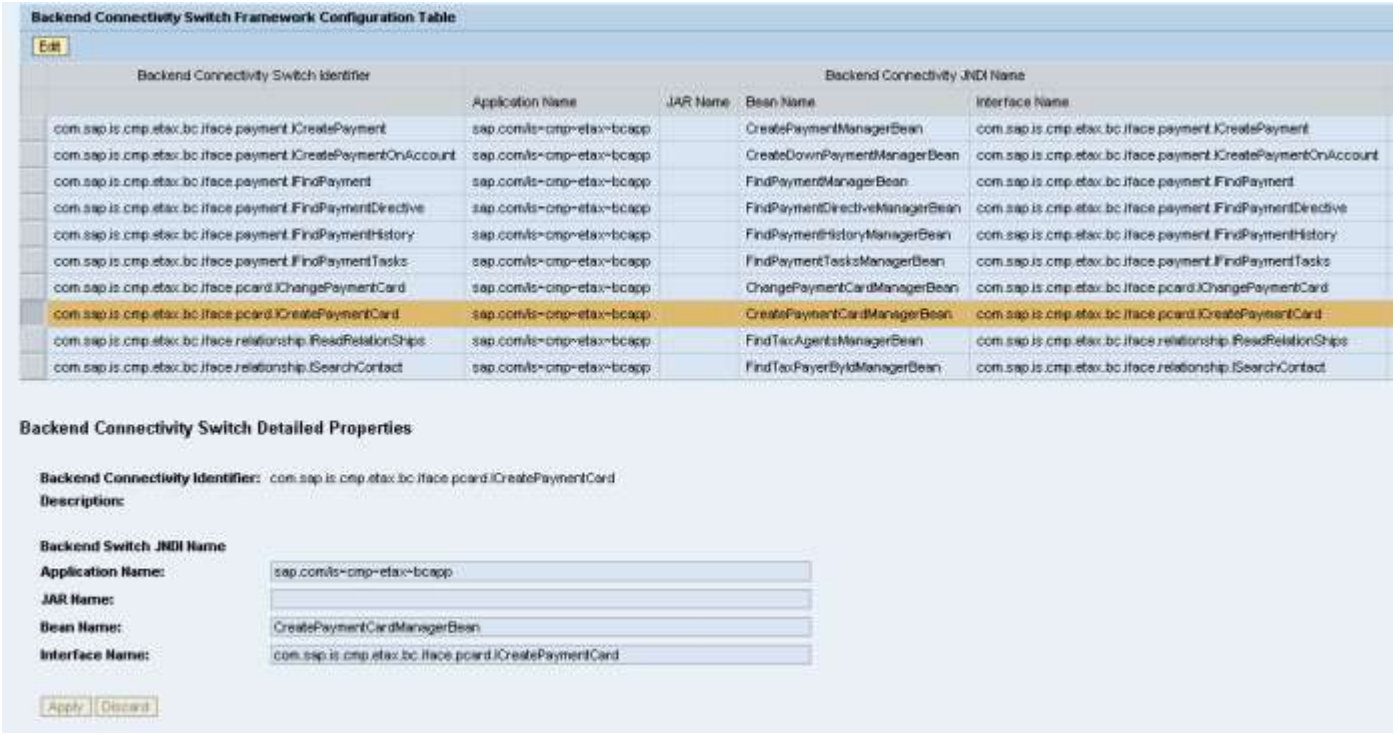
Step2: Select the Local interface <EJB Application DC>/LOCAL/<your Bean name>:



- The “Object Name” contains the fully qualified JNDI name of your Bean.

3.4.2 Enter the new configuration key

Go to TPOS Admin UIs, Backend Abstraction Layer. Choose your interface, press Edit.



Enter the name of your xyz_app for “Application Name” and your BC bean name to “Bean Name:”

For example: in this example the configuration looks like: