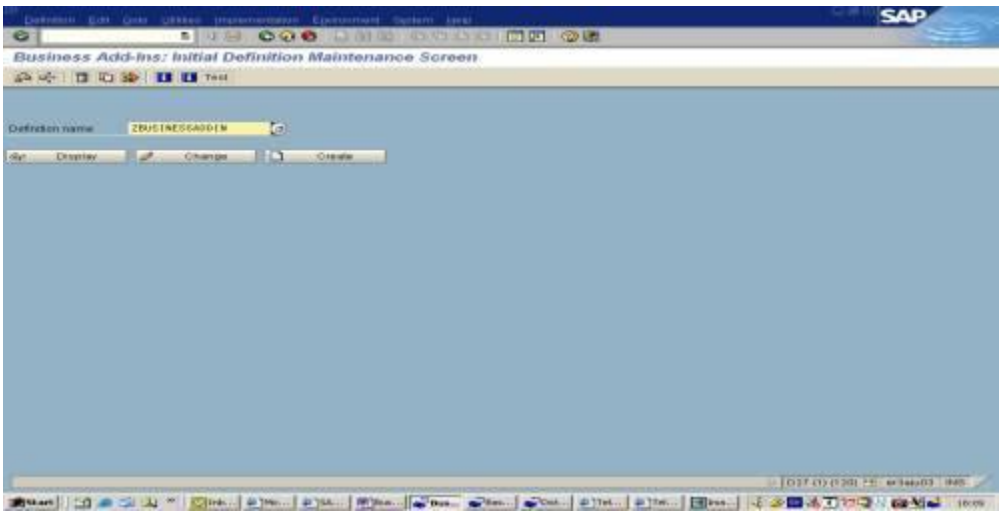
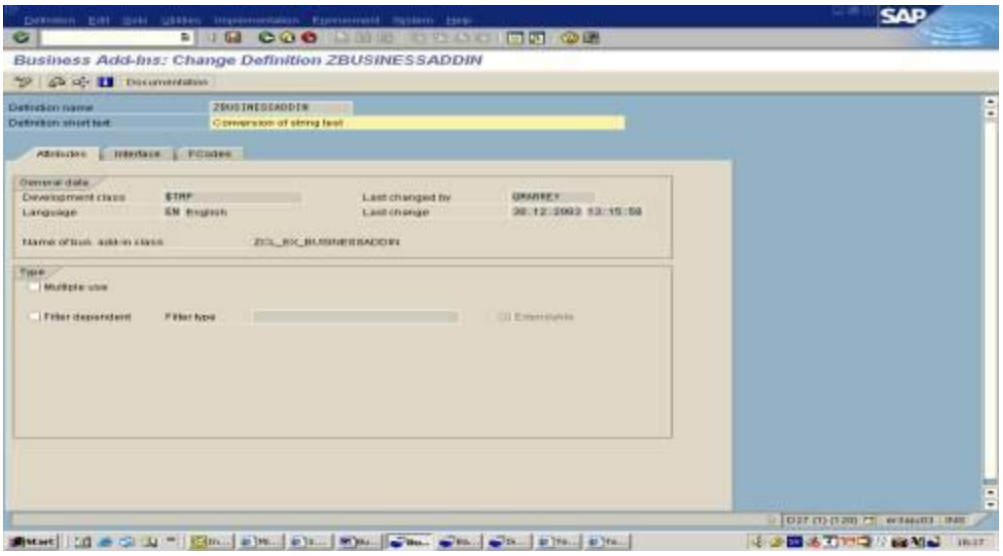
Defining a Business Add-in

SAP provides the BADI's where are applicable in the standard applications. Application programmer whoever wishes to have a Business Add-ins in a particular program can define the interface for an enhancement in the Business Add-in builder. Programmer has to program the interface call in the program at the appropriate place. Customers can select the add-in and implement it accordingly to their business needs.

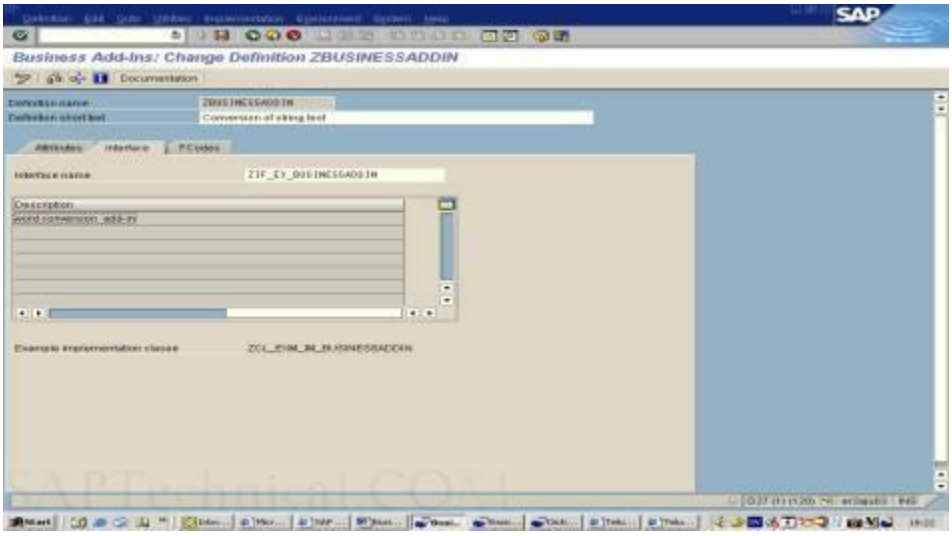
1. From SAP menu, choose Tools -> ABAP Workbench -> Utilities -> Business Add-ins or transaction code SE18. The example, which is illustrated, is the string conversion in the program. And giving the provision to the users to determine themselves how their strings are to be converted. Application developer define an enhancement, consists of interface with a method with changing parameter used to pass the string.
2. Enter the BADI name and choose create



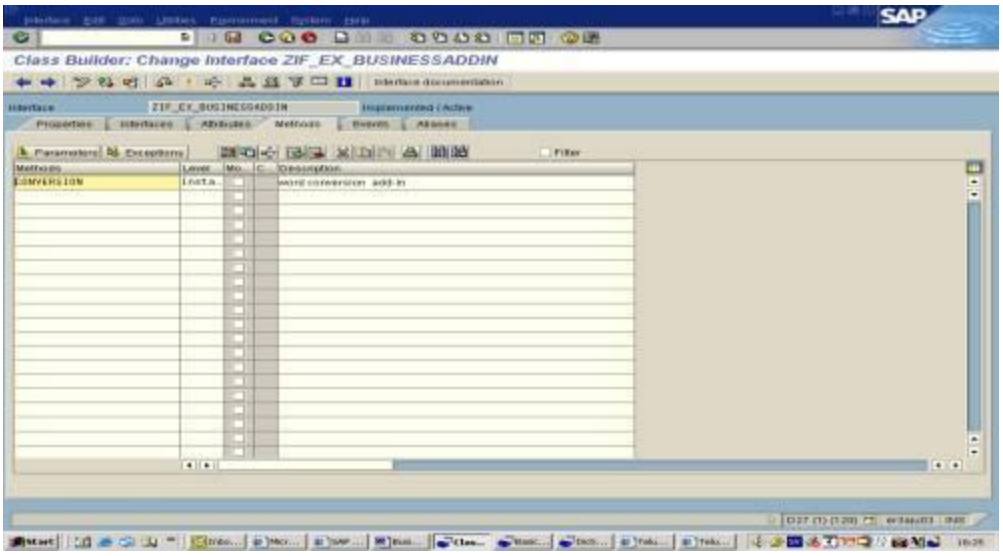
3. Enter the short text, choose the interface tab.



4. Double click on the interface name field. The system branches to the class builder.



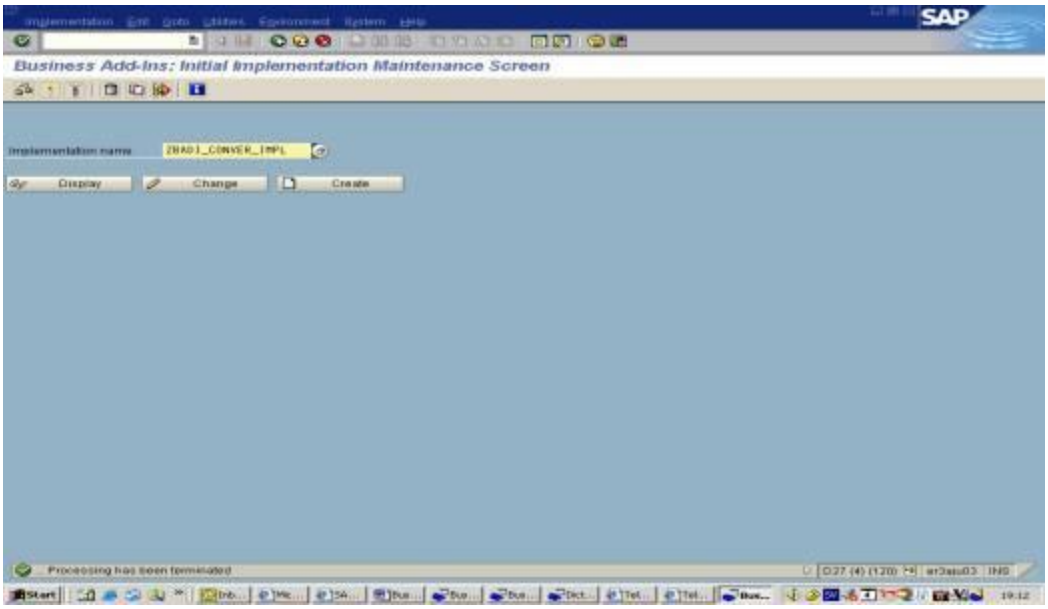
5. In the class builder assign a method to the interface and define a parameter with the attributes.



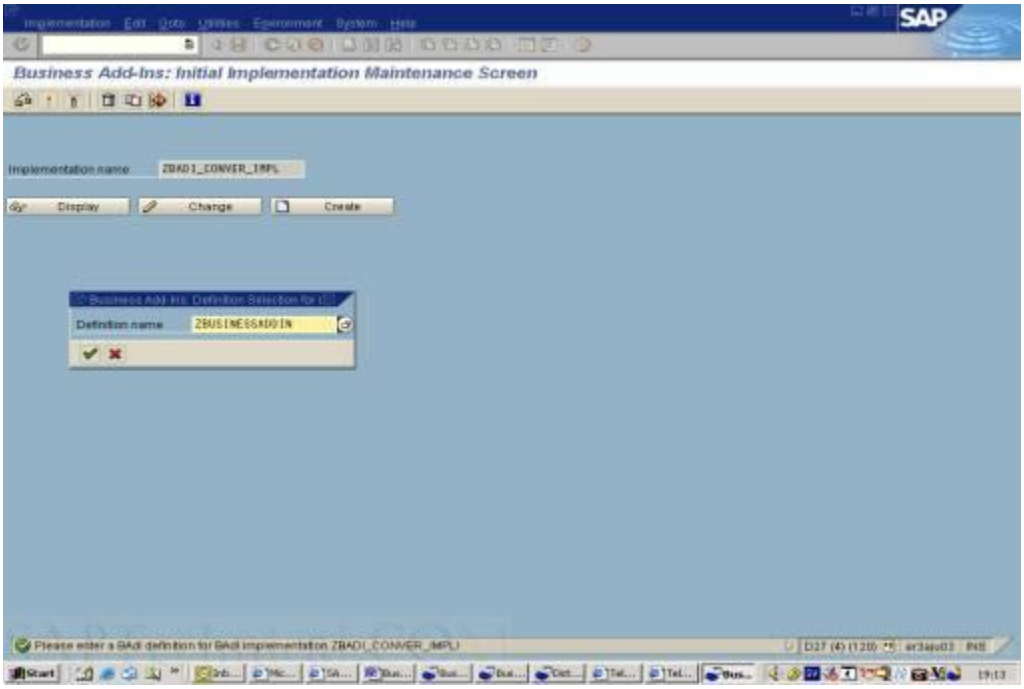
6. Save and activate the interface and navigate back to the Business Add-in definition. Now in the BADI screen, displays the method you have created for the interface. When you maintain the interface methods, corresponding executing class (Adapter class) is generated.
7. Save your entries and document the description of the Business Add-in. Documentation is important for the users to understand the purpose of the Add-in.

Implementation of BADI

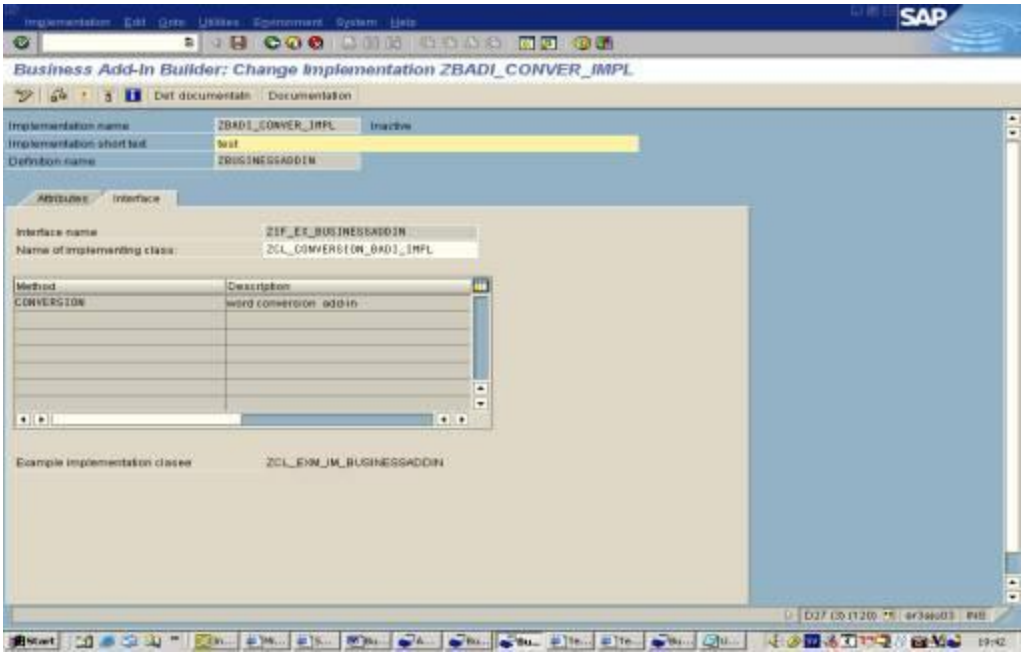
1. The list of Business Add-ins available in the system can be found through SAP Reference Implementation guide (IMG) or in component hierarchy. BADI's definition is included in IMG so that the customer/partner can create suitable, company-specific implementations
2. In the SAP menu, choose ABAP Workbench -> Utilities -> Business Add-ins or transaction code SE19.
3. Enter the implementation name and click on the create button.



4. Enter the BADI name



5. Enter the short description for the BADI implementation and implement the interface in the class appearing in the BADI implementation screen.



6. Double click on the implementation class and insert the desired source code for the implementation between the **method ZIF_EX_BUSINESSADDIN~CONVERSION.** And **Method**. In this particular example enter the statement **translate parameter to upper case**. Save and activate your entries and return to the change implemention screen.
7. Choose Activate, now you can use this implementation when the application program is executed. Several implementations may exist for a Business Add-in but that is not used in multiple use basis. However only one implementation can be activate at any one time. But in case of **multiple use** of the BADI, we can have multiple implementations activate. The instance generation of the implementing class must set the attribute as public and not as private, protected or abstract. System will give dump if you do so.
8. Following is the code for the class ZCL_CONVERSION_BADI_IMPL method CONVERSION

```
Method ZIF_EX_BUSINESSADDIN~CONVERSION.
translate parameter to upper case.
endmethod.
```

Calling BADI in the Application

1. When we define BADI, enhancement management generates a class that implements the interface. The application developer uses a factory method to create an instance of adapter class in the application program and calls corresponding method. The adapter class method generated by the enhancement management decides by checking the entries in the table whether one or several active implementations need to be called. If required, the implementations are subsequently executed. The application program ensures only the adapter class method is called. The application program doesn't know which implementations are called.

2. Call the string conversion Business Add-in, the program calling the Business Add-in. Following is the ABAP source code

```
REPORT ZMPTEST_BADI.
* Declaring the handler
class: cl_exithandler definition load.
* Interface Reference
data: badi_interface type ref to ZIF_EX_BUSINESSADDIN.
* String
data: w_str(15) type c value 'baddi test'.
*********************************************
****Start of Selection Event.....................
*********************************************
start-of-selection.
  call method cl_exithandler=>get_instance
            changing
              instance = badi_interface.
   write: / 'Please click here'.
*********************************************
****At line-selection Event.....................
*********************************************
at line-selection.
 write: / 'original word', w_str.
 if not badi_interface is initial.
  call method badi_interface->conversion
      changing parameter = w_str.
 endif.
  write: / 'Converted word', w_str.
```

Filter dependent Badi

1. Business Add-in definition level (for example a country, industry sector) we can have filter dependent option. If an enhancement for country specific versions then it is likely that different partners can implement this enhancement. The individual countries can create and activate their own implementation.
2. In the enhancement definition, all the methods created in the enhancement's interface need to have filter value as their importing parameter. The application program provides the filter values for the implementation method.
3. Filter dependent BAdi is called using one filter value only, it is possible to check active implementation for the filter value using the function module SXC_EXIT_CHECK_ACTIVE.

Multiple use Badi

1. There are multiple use and single use Business Add-ins. This option can be choose at Business Add-in definition.
2. The distinction is base on the procedure or event character of an enhancement. In the first case the program waits for the enhancement to return a return code. Typical example is benefit calculation in HR depending on the implementation, alternative calculations can be executed. In case of multiple use add-ins, an event that may be interest to other components in program flow. Any number of components could use this event as a hook to hang their own additional actions on to.
3. There is no sequence control for multiple-use implementations of BAdi's. Sequence control is technically impossible, at the time of the definition the

interface does not know which implementations parameters will be change the implementations.

4. The concept of multiple use of the Business Add-in is that has been implemented once already can be implemented again by right of the software chain.