Tips and Techniques

This section contains information you can refer to while developing business services. The following topics are covered:

- Wrap Multiple Subprograms into a Business Service
- Use *ISN as the Unique Primary Key for Maintenance
- Access the Value of +METHOD
- Determine Where a Transaction is Completed

Wrap Multiple Subprograms into a Business Service

When creating business services based on existing subprograms, the real power of this type of business service comes from a developer's customizations. (For examples of these customizations, refer to the BNUM and BSTRINGN subprograms in the SYSBIZDE library.)

One of your goals as a business service developer is to simplify the business service for a consumer by exposing as little as possible and by categorizing the information that is expected of the business service. To do this, determine which subprogram parameters must be exposed by the business service and categorize them into four categories: Input, Input and Output, State, and Output. This is done for the following reasons:

1. It helps the business service consumer easily identify what the service is expecting (input), what is passed back (output), what is necessary to maintain the state of the service on the server (but not necessary to expose on a GUI) (state), and what attributes can be considered for both input and output (input and output).

   Note:
   Even if data is passed to the service in the output attributes, these attributes will be reset before processing on the server takes place. It is also assumed that even though values can change on the server for the input attributes, these values will not be reflected back to the client (but you can override these principals).

2. Some level 1 groupings or individual parameter variables do not need to be exposed on the client. Although they are required for the lower-level subprogram(s), the server understands the requirements and, with your help, can derive the appropriate values. In the BNUM subprogram, for example, the two internal programs require two operands, but the first subprogram called them #FIRST-NUM and #SECOND-NUM and the second subprogram called them #OPERAND-1 and #OPERAND-2. The business service developer decided to expose #FIRST-NUM and #SECOND-NUM and handle the correct population of these variables in the **SAG DEFINE EXIT MOVE-TO and **SAG DEFINE EXIT MOVE-BACK exits.

If two subprograms use the same PDA containing the same level 1 structure, the first level 1 encountered will be processed. An example of this situation is the CDPDA-M messaging PDA used in many Natural Construct-generated subprograms. If two subprograms have the same level 1 name, but the content is different, you must analyze this information and determine how it should be processed. But what happens if duplicate variable names get placed under the same category? For example, in the scenario below both

EMPL and VEH can be selected for the INPUT and OUTPUT category:

```
1 EMPL
   2 PERSONNEL-ID (A8)
   2 NAME (A30)
1 VEH
  2 PERSONNEL-ID (A8)
  2 MODEL (A30)
```

This creates a problem because attributes are uniquely identified within level 1 groupings. When the wrapper subprogram places these two groups under the #BIZ-INPUT-OUTPUTS grouping, it creates the following data structure:

```
01 #BIZ-INPUT-OUTPUTS
  02 EMPL
      03 PERSONNEL-ID (A8)
      03 NAME (A30)
  02 VEH
      03 PERSONNEL-ID (A8)
      03 MODEL
```

A Natural developer will recognize that this code will not compile and a business service developer will recognize the potential of exposing redundant attributes. It is important that the business service consumer not have more attributes than required as that causes larger messages to be transported across the network than are necessary and adds to the complexity of the business service by exposing more attributes than are required. Unless the data should be different between the data structures, the business service expects only one input/output attribute called PERSONNEL-ID in the example above. If the data should be different, the attributes must clearly define the difference as you do not want the consumer providing PERSONNEL-ID in the EMPL structure when it was needed in the VEH structure. What really matters is that this business service has an attribute called PERSONNEL-ID and the business service developer decides how to handle the internal structures.

One solution to this scenario is to take PERSONNEL-ID out of EMPL and VEH and place it at a common level within the set category. For example:

```
01 #BIZ-INPUT-OUTPUTS
  02 EMPL
*      03 PERSONNEL-ID (A8)
       03 NAME (A30)
  02 VEH
 *     03 PERSONNEL-ID (A8)
       03 MODEL
   02 PERSONNEL-ID (A8)
```

With this solution, nothing has to change in the MOVE user exits. The only down side to this solution is if there are different formats and lengths for variables with the same name. For duplicate level 1 structures and duplicate fields, only the first field will be processed with the Business Service wizard. The duplicate fields will still exist in the structures in their original formats and lengths, but they will be commented out (see above). This code, however, is generated into user exits so you can modify the solution.

Note:
If a parameter data area is changed in a business service subprogram, you must regenerate the business service proxy.

In the MOVE exits above, there are typically MOVE BY NAME statements. Ensure that you do not accidently overwrite these assignments. For example, the following code was added to the MOVE-BACK exit in the BSTRINGN subprogram to prevent the MOVE BY CSACASE statement from overwriting the data in #BIZ-INPUT-OUTPUTS. For the ReverseString method, CSACASE.INPUT-STRING would be blank:

```
IF +METHOD = 'ReverseString' THEN
   MOVE BY NAME FLIPSTRA TO #BIZ-INPUT-OUTPUTS
   #BIZ-OUTPUTS.MSG := ##MSG
   #BIZ-OUTPUTS.MSG-NR := ##MSG-NR
 ELSE
   MOVE BY NAME CSACASE TO #BIZ-INPUT-OUTPUTS
 END-IF
```

Use *ISN as the Unique Primary Key for Maintenance

When using the Business Service wizard to generate new subprograms for data access, one of the options is to generate a single view data access service. This option allows the service to browse by a non-unique key while uniquely maintaining the data object using Adabas's internal sequence number (*ISN).

To take advantage of this feature, the business service must be generated:

- With both the browse and maintenance functions
- For a single view
- With the GET-BY-ISN option enabled

This section covers the following topics:

- Enable the GET-BY-ISN Option
- Test the *ISN Feature
- Generate a Single View Service that Maintains Data by ISN

Enable the GET-BY-ISN Option

To enable the GET-BY-ISN option:

1. Logon to the SYSCSTX library.
2. Edit the CSXDEFLT subprogram and uncomment the following code:

   ```
   VALUE 'GET-BY-ISN'
       CSADEFLT.PARM-VALUE := TRUE
   ```

3. Stow CSXDEFLT.
4. Use the SYSMAIN utility to copy CSXDEFLT to SYSLIBS.

Tip:
To implement this functionality for an existing business service, regenerate the business service after specifying the GET-BY-ISN option. (The entire service must be regenerated, not just the proxy.)

Test the *ISN Feature

To test that this feature is working:

1. Open the context menu for the business service in the **NBS Repositories** view.

   For information, see:

   - Eclipse plug-in: Test a Business Service
   - Natural plug-in: Test a Business Service

2. Test a FindBy method that returns non-unique keys.
3. Modify non-key data in the rows that have non-unique keys.
4. Enter "U" in the State field for each row that you modified.

   This will update the rows.

5. Use the MultiMaint method to modify the data (after all data is entered).

   The data was successfully committed to the database if the State field is now US.

Notes:

1. For an object maintenance subprogram, this feature works with the GET, UPDATE, and DELETE methods. If the feature is enabled, the object PDA contains an extra field called OBJECT-ISN. To allow the object maintenance subprogram to use this value, OBJECT-ISN must be populated and the #USE-ISN value in the CDAOBJ2 data area must be set to True.
2. This option is not currently available for the NEXT and FORMER actions.

Generate a Single View Service that Maintains Data by ISN

If the GET-BY-ISN option has been defined in the CSXDEFLT subprogram, the Business Service wizard can use an existing object browse and object maintenance subprogram to generate a single view data access business service that browses by a non-unique key and maintains data by *ISN. To do this, the existing subprograms must:

- Access the same file (for example, the ACUSTN and MCUSTN subprograms)
- Not have intra-object relationships

The wizard uses the Object-Browse-Subp model to generate an object browse-select subprogram that uses the FindBy methods. For more information, see:

- Eclipse plug-in: By Generating New Subprograms for Data Access.
- Natural plug-in: By Generating New Subprograms for Data Access.

Access the Value of +METHOD

The +METHOD variable contains the method that was used by a subprogram. For information on accessing the value in +METHOD within a subprogram, see +METHOD.

Determine Where a Transaction is Completed

The ##TRANSACTION variable determines where a transaction is completed. For information on this variable, see Additional Standard PDA Variables.