Documentation

- Tools: Storable Class editor and BRM SDK Opcodes
- Storable Class Editor(via Developer Center):
    + Define a /nexai class as an subclass of /service
    + Adds fields to them
    + Should inherit some important fields (primary POID and other foreign class POID)
    + Add new field, dependent on the product, for our use case
    + Note: we should consider adding all fields we need, and depending on the product, we only use certain necessary fields

I. Setup the CM for writing a new class

1. Modify the pin.conf
    - ☐ Open the Oracle DM configuration file (BRM_Home/sys/dm_oracle/pin.conf) in a text editor. (BRM_Home is the directory in which you installed BRM components.)
    - ☐ To enable field creation in the data dictionary, set the following entry to 1:

        - dm dd_write_enable_fields 1
    - ☐ To enable the creation, editing, or deletion of custom storable classes in the data dictionary, set the following entry to 1:

        - dm dd_write_enable_objects 1

2. Increase the size of the CM Cache for the Data Dictionary
    - ☐ Open the Connection Manager (CM) configuration file (BRM_Home/sys/cm/pin.conf)
    - ☐ Increase the cache_size in the following entries:

        - cm_cache cm_data_dictionary_cache number_of_entries, cache_size, hash_size

        - cm_cache fm_utils_data_dictionary_cache number_of_entries, cache_size, hash_size
    - ☐ Save the file.
    - ☐ Stop and restart the CM. See "Starting and Stopping the BRM System" in BRM System Administrator's Guide.

3. Use DDL when updating data dictionary tables

You can configure the DM to execute Data Definition Language (DDL) when updating object types in the data dictionary tables. This ensures that database objects are mapped to the correct tables.

To specify whether DDL is used when updating the data dictionary tables:

- ☐ Open the Oracle DM configuration file (BRM_Home/sys/dm_oracle/pin.conf) in a text editor.
- ☐ Set the sm_oracle_ddl entry to one of the following:

  0 to not execute DDLs when updating object types in the data dictionary.

  1 to execute DDLs when updating object types in the data dictionary.

  - dm sm_oracle_ddl 1

- ☐ Save and close the file.

II. Create Custom Fields

When you create a new field in Storable Class Editor, it is committed to the data dictionary when you click OK in the New Field dialog box. You can't delete a field after it has been committed. For more information, see Storable Class Editor Help.

- ☐ In the Storable Class Editor, choose File - New - Field.
- ☐ In the Field Name box, enter a unique name for the new field.
- ☐ In the Type list, choose a field type in the list.
- ☐ In the Description box, enter text to define the purpose of the field.
- ☐ In the Field ID field, change the automatically assigned ID number.

The following table lists the field ID ranges for Oracle-only use and for customer use:

Table 14-1 BRM Field ID Restrictions

Field ID Range	Reserved For
0 through 9999	Oracle use only
10,000 through 999,999	Customer use
1,000,000 through 9,999,999	Oracle use only
Over 10,000,000	Customer use

- ☐ Click OK.

III. Create Custom Storable Classes

- ☐ In Storable Class Editor, choose File - New - Class.
- ☐ In the Class Name field, enter the class name in the format /classname for base classes and /classname/subclass for subclasses.
- ☐ In the Label field, enter a name for the storable class.
- ☐ In the Description field, enter text to define the purpose of the storable class.
- ☐ Click OK.

The storable class opens in a Class Definition window. The class name appears in the title bar.

- ☐ Select the root icon of the new class.
- ☐ In the Properties window, choose values for the Read Access and Write Access properties.
- ☐ Add fields to the class by dragging field icons from the Class Browser or Field Browser.
  Note:
  Each storable class must have the PIN_FLD_ACCOUNT_OBJ field. If you do not include this field, it is automatically inserted when the storable class is created.
- ☐ Choose File - Commit New Class to commit the class to the data dictionary.
- ☐ If you created a new subclass and if the base class is partitioned, you must run the partition_utils script with the -n parameter to ensure that the new subclass uses the same partitioning layout as its base class.

  See "Partitioning Tables" in BRM System Administrator's Guide.

IV. Making Custom Fields Available

- ☐ In Storable Class Editor, choose File - Generate Custom Fields Source to create source files for your custom fields. See the Storable Class Editor Help system for detailed instructions.
  Storable Class Editor creates a C header file called cust_flds.h, a Java properties file called InfranetPropertiesAdditions.properties, and a Java source file for each custom field.

  For applications written in PCM C or PCM C++, perform these steps:

  - ☐ Run the parse_custom_ops_fields.pl Perl script on the cust_flds.h file created by Storable Class Editor. Use this syntax:

    parse_custom_ops_fields -L language -I input -O output

  - ☐ For information on the parameters of parse_custom_ops_fields and their valid values, see "parse_custom_ops_fields".

  - ☐ In the pin.conf file for applications that need to access these fields, including testnap and other utilities, create an entry using the format shown below. Replace cust_flds with the file name and location of the memory-mapped extension file that the parse_custom_ops_fields script created.

    - - ops_fields_extension_file cust_flds

    Note:
    Do not add more than one ops_fields_extension_file entry. The custom fields source file and the extension file that results from it contain information about all the custom fields in the data dictionary, so a single reference to that file is sufficient.

  - ☐ Include the cust_flds.h header file in the applications and in the FMs that use the fields.

Note:

Default BRM fields are defined with their numbers in the pin_flds.h file in the BRM_Home/include directory. While it is possible to add custom fields directly to pin_flds.h, you should not do so. Placing custom field definitions in the separate cust_flds.h file allows you to upgrade to new releases without having to edit pin_flds.h.