Using SQL in Barista

March 2009

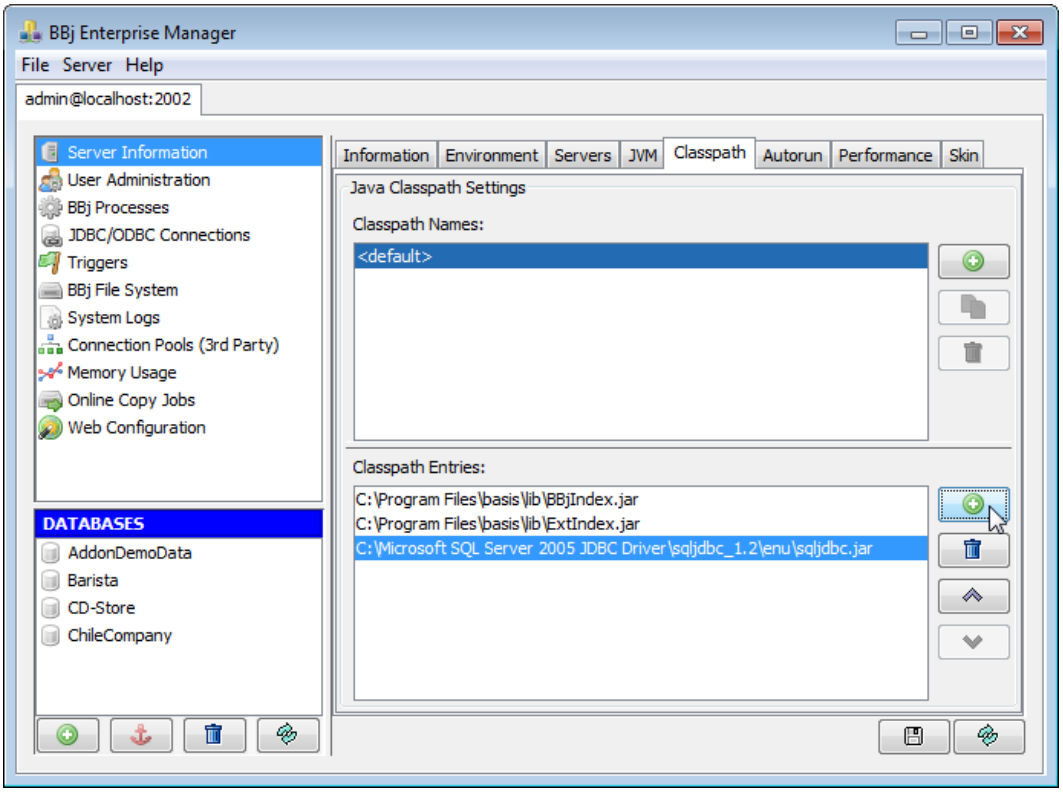
By

Ralph Lance
BASIS Europe Distribution GmbH
rlance@basis-europe.eu

Introduction

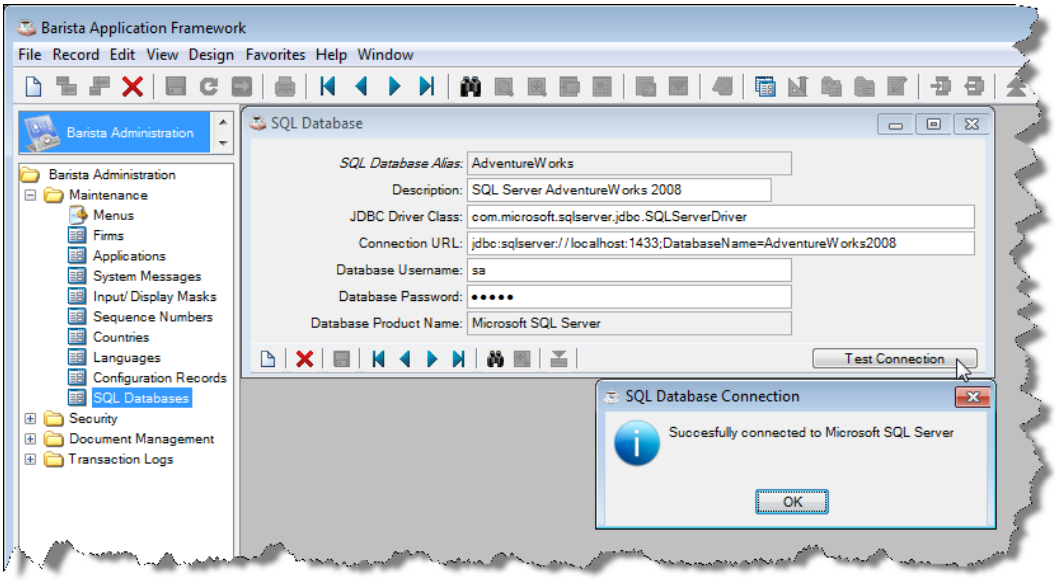
(back to top)

The following steps show how easy it is to incorporate a SQL database into Barista® Application Framework. Although Microsoft's SQL Server 2008 AdventureWorks sample database is used for this example, you can use any JDBC 2.0 compatible database, including the BASIS ESQL database that comes with BBj®. When working with an external database, ensure that the JDBC driver for that database is in the default classpath of BBj Services. You can add the jar file(s) from the Classpath tab of the Server Information section of Enterprise Manager. Restart BBj Services after making any changes to the default classpath.



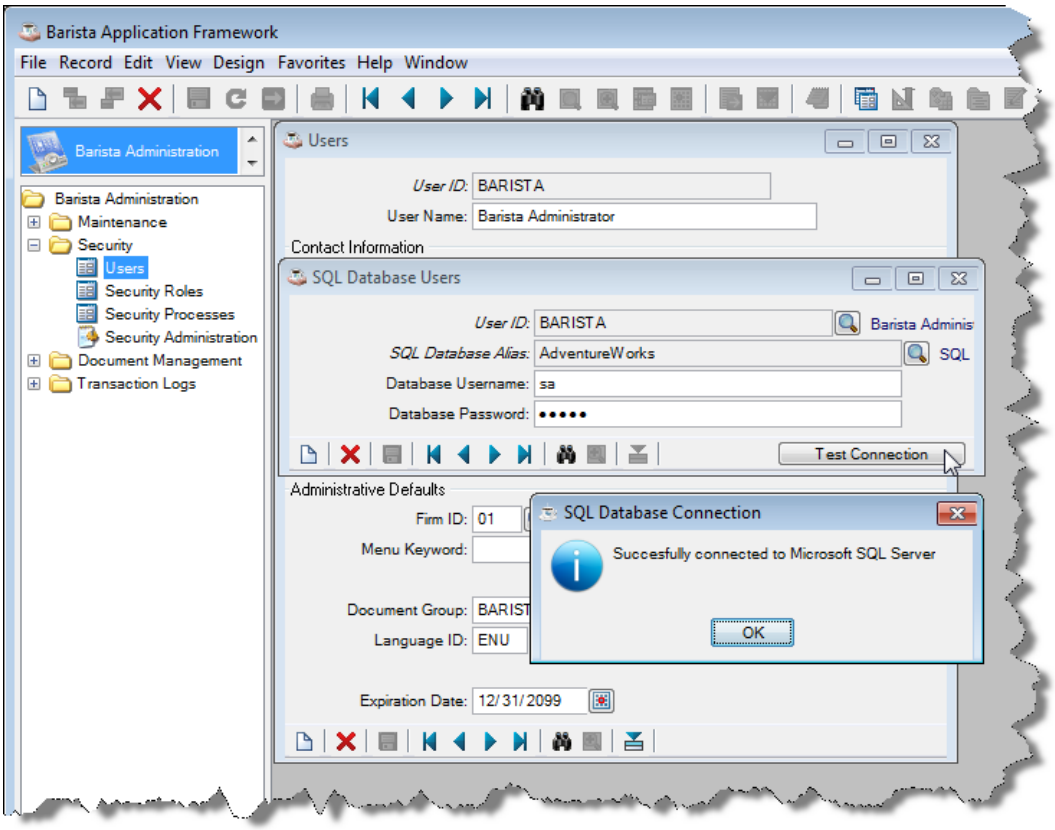
Barista Administration → Maintenance → SQL Databases

Enter an SQL Database Alias to be used in Barista, and then enter the JDBC connection information for that database. After saving, you should use [Test Connection] to verify that the connection is correct. A successful connection returns the database product name; save once more to save the product name. The username and password in this form are only used here to test the connection. The user credentials entered in the next step are used to make any further connections in Barista.

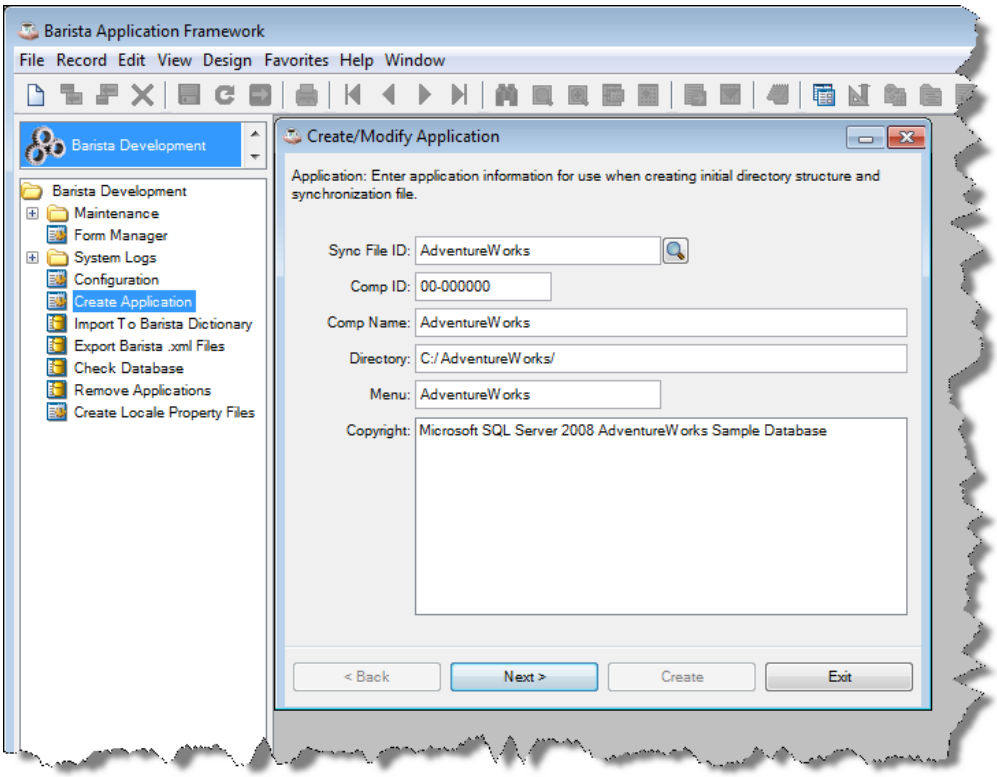


Barista Administration → Security → Users

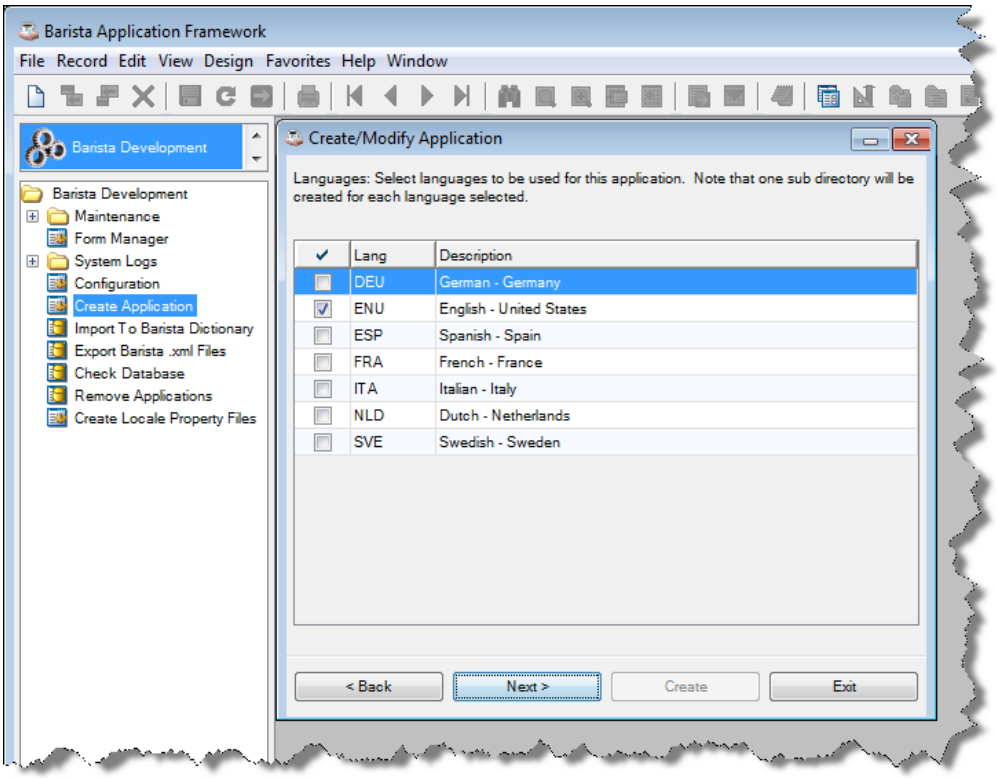
For each user who will be using this SQL database, choose "SQL Database Users" from the Option menu. Choose the database alias and enter the database username and password. You should verify the credentials by pressing the [Test Connection] button after saving.



Barista Development → Create Application



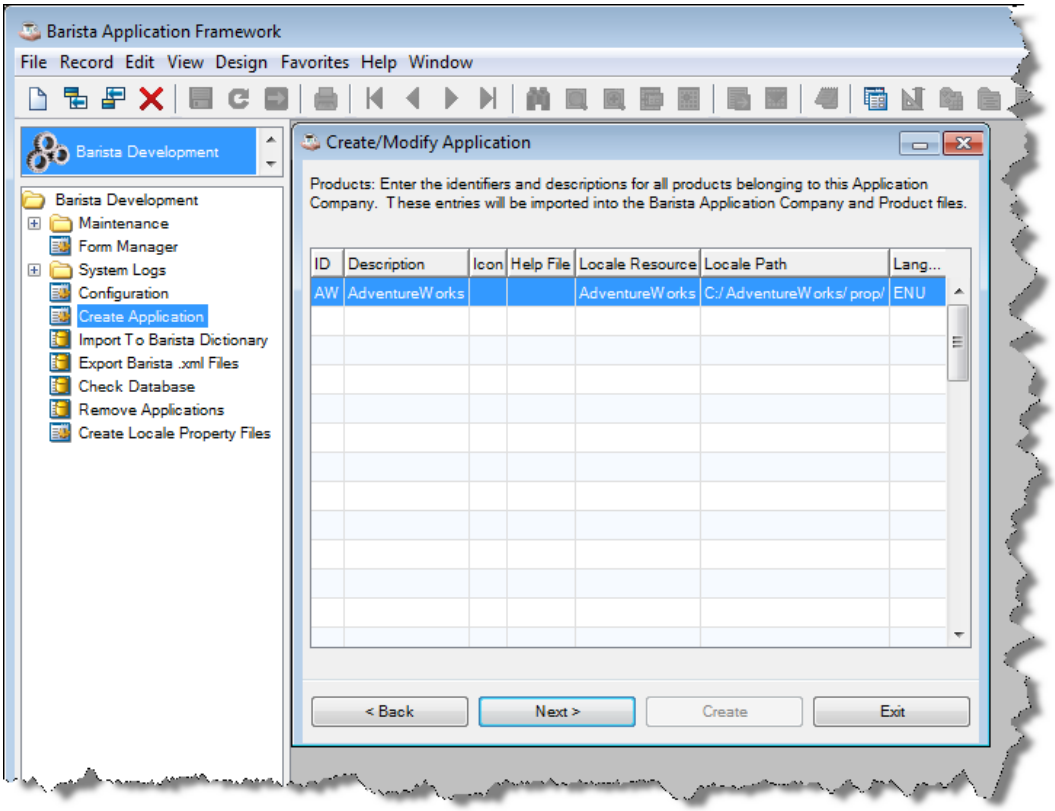
Click [Next].



Click [Next].

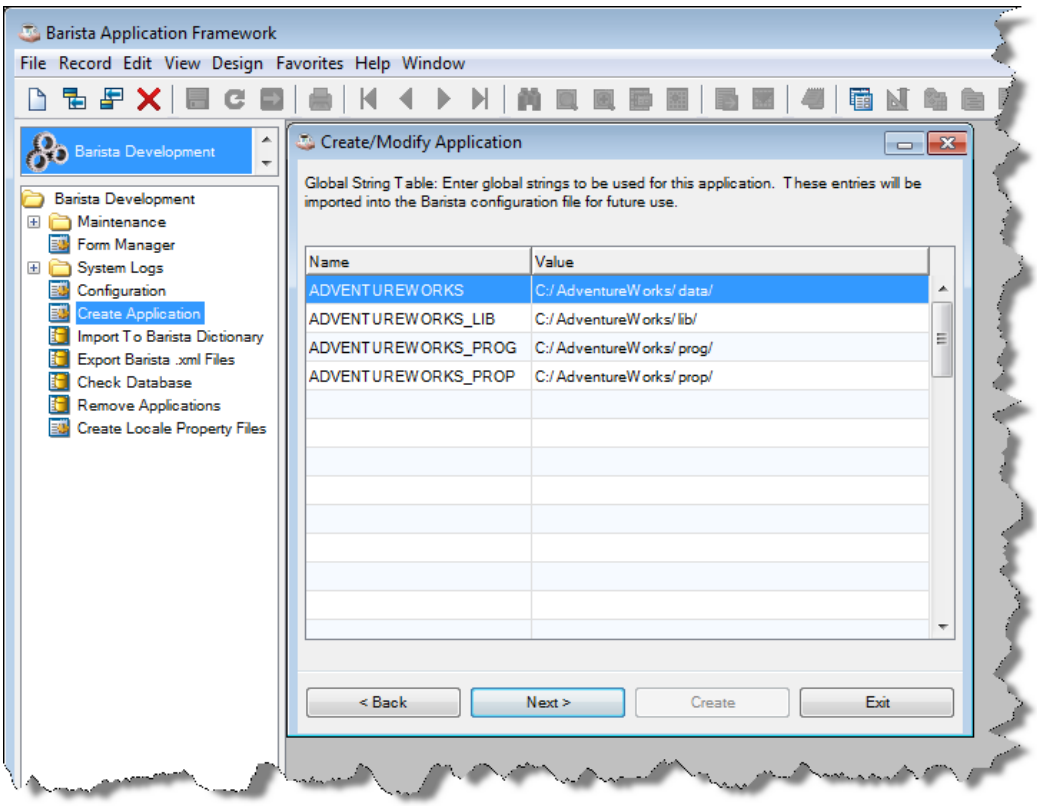
The following application product panel is used to identify related items, mainly data elements and tables, in Barista as belonging to a particular company and product or module. Enter a two or three character code, AW in our example. (To be consistent, we'll use this code later as the "application prefix" that will be prepended to table and data element names.)

Enter a description and leave the rest of the product row with its defaulted values.

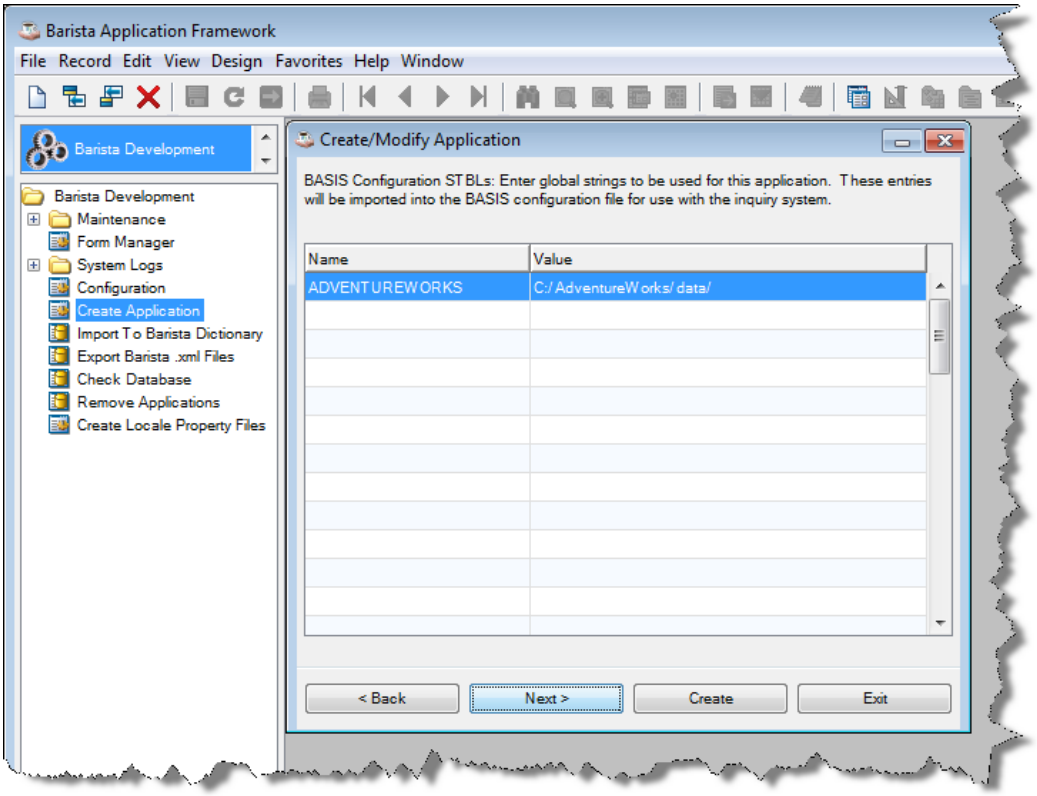


Click [Next].

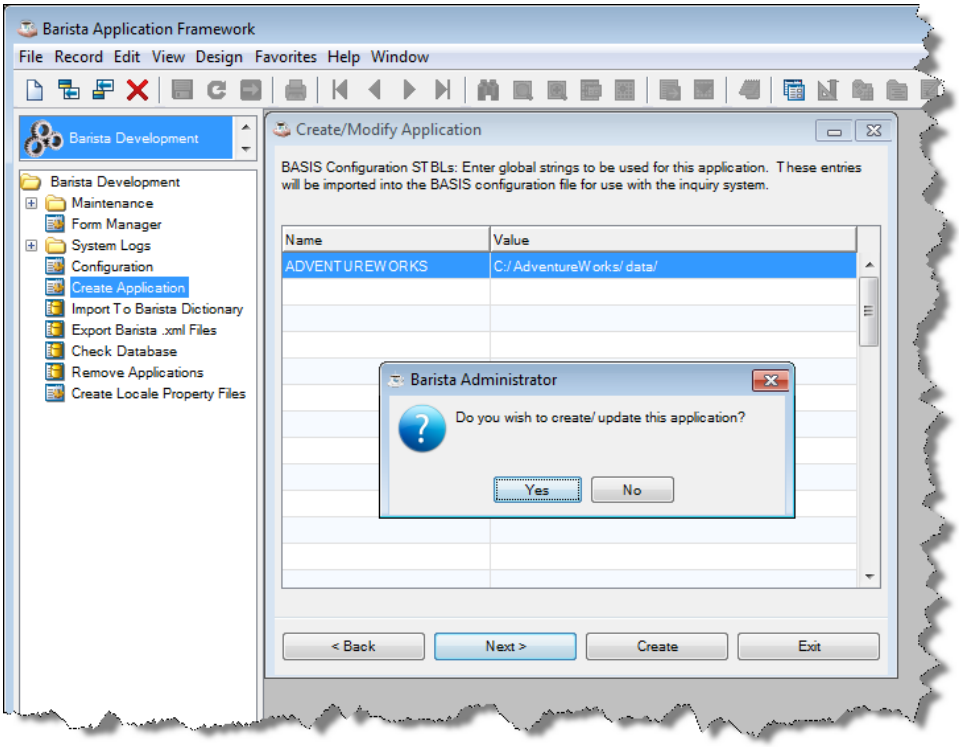
Note the global name that points to the application data (defaults to the name of the "Sync File ID," in capital letters). You will enter it later when importing table definitions from the SQL database.



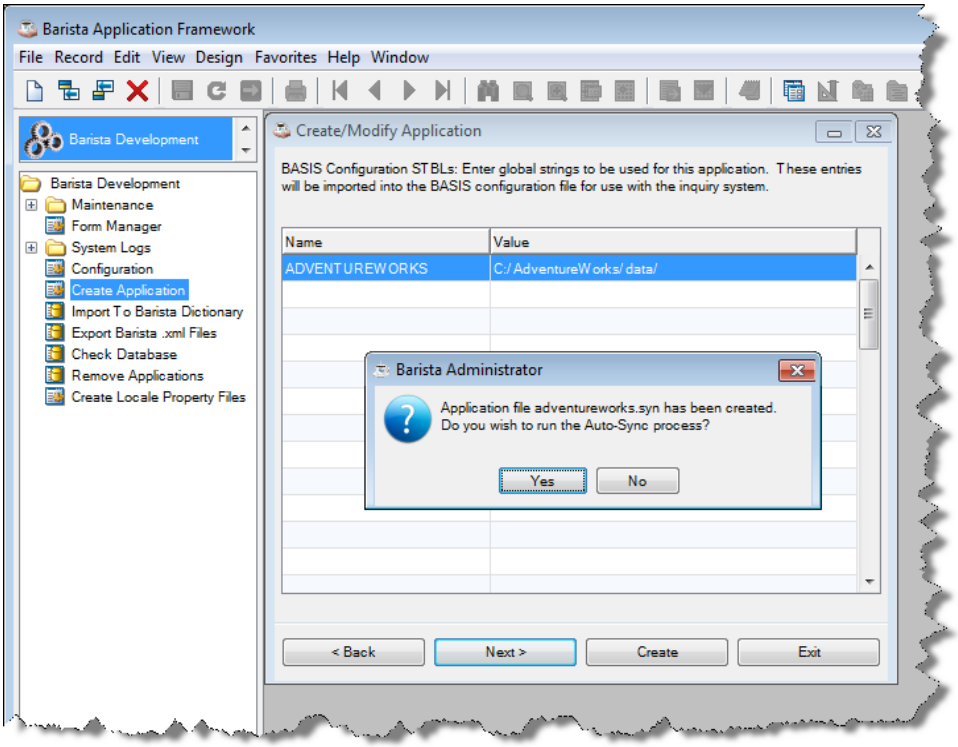
Click [Next].



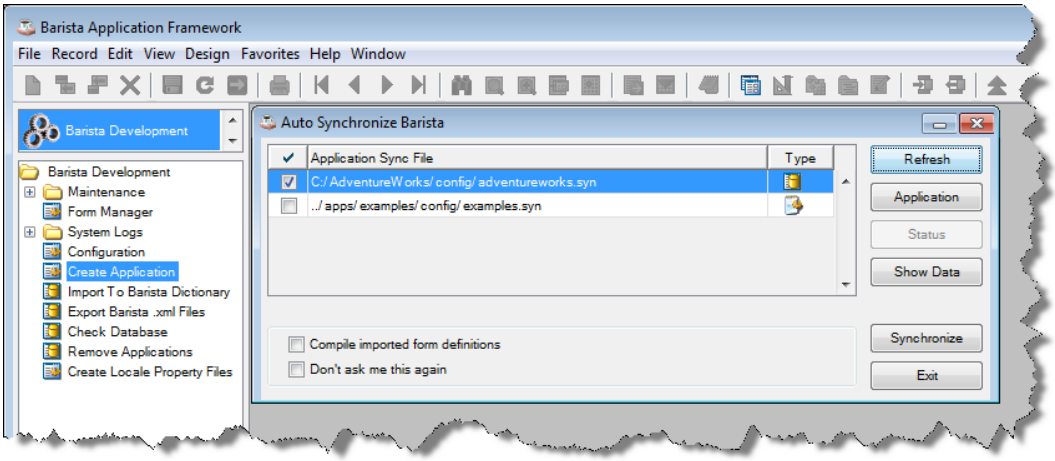
Click [Create].



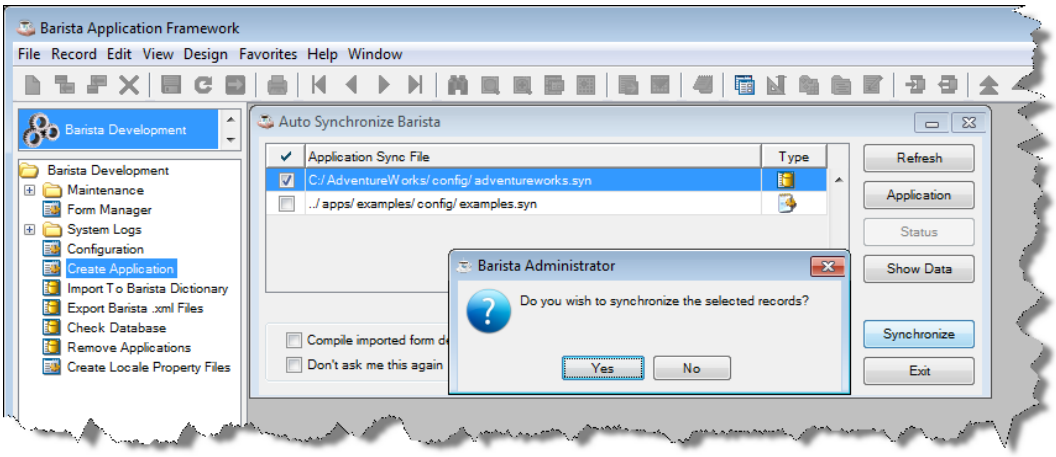
Click [Yes].



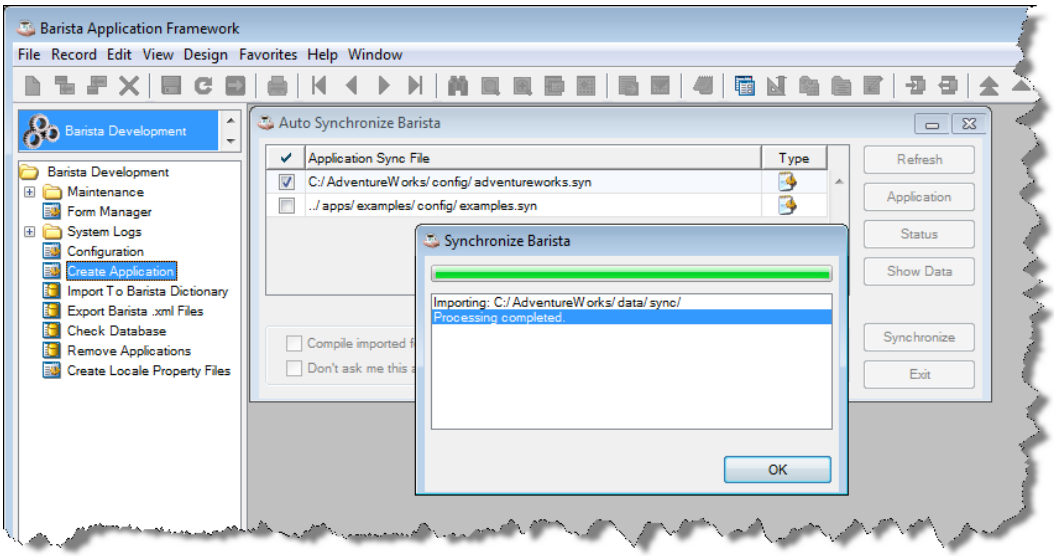
Click [Yes].



Click [Synchronize].



Click [Yes].



Click [OK].

Now we have an application configured that we'll use to import our SQL tables into Barista.

Importing Table Definitions from a SQL Database

(back to top)

Run the menu item "Import to Barista Dictionary" under "Barista Development."

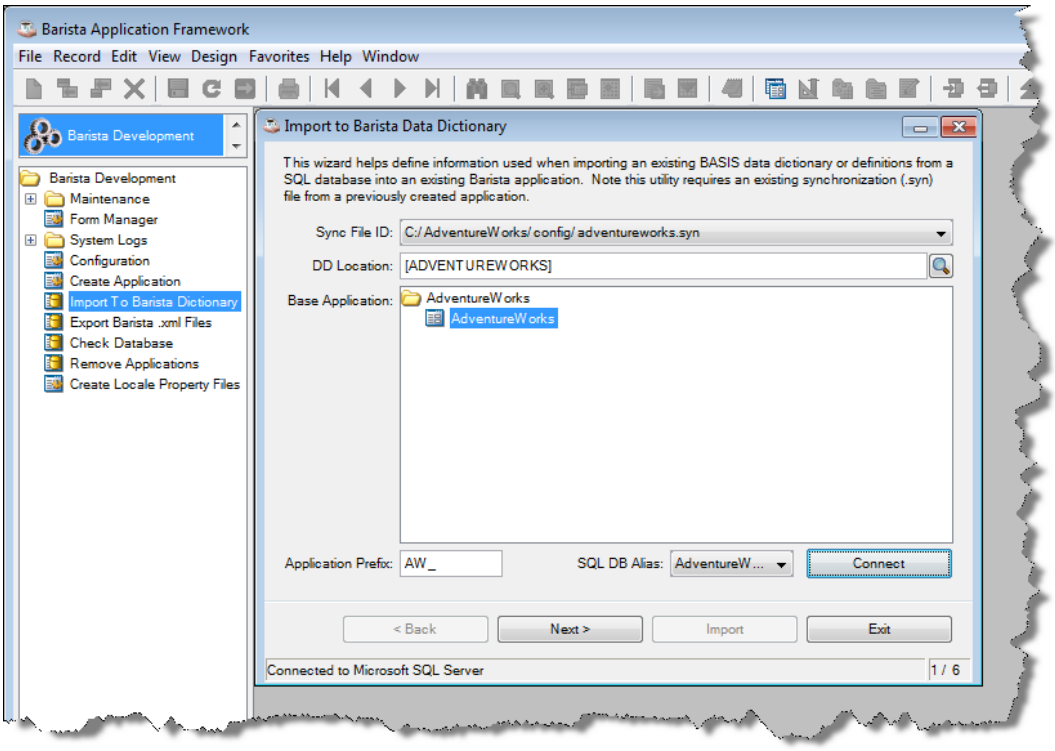
Choose your application's `.syn` file found in the `config` directory under your application's directory.

There is no BASIS Data Dictionary directory for SQL databases, so the field "DD Location" must contain the global variable, enclosed in brackets ("[..]"), that represents your application's data directory. It's the name used in the "Create Application" noted earlier and in our case `[ADVENTUREWORKS]`. This directory global will be used to point to the application's resource (`.arc`) and form definition (`.def`) files used by Barista.

Ensure the application (usually the lowest branch) in the "Base Application" tree is selected.

Enter the two-character application code `AW` in the "Application Prefix" field. (An underscore will be automatically added to the prefix, if it doesn't end with one.) This prefix is an enormous help when finding and filtering your application's own data elements, tables, forms, etc.

Choose the SQL Database Alias that we setup in the beginning and then click the [Connect] button. The Barista user's "SQL Database User" name and password will be used to make a JDBC connection to the database. If the connection is successful, then you will see a "Connected to..." message in the form status line below.



Click [Next].

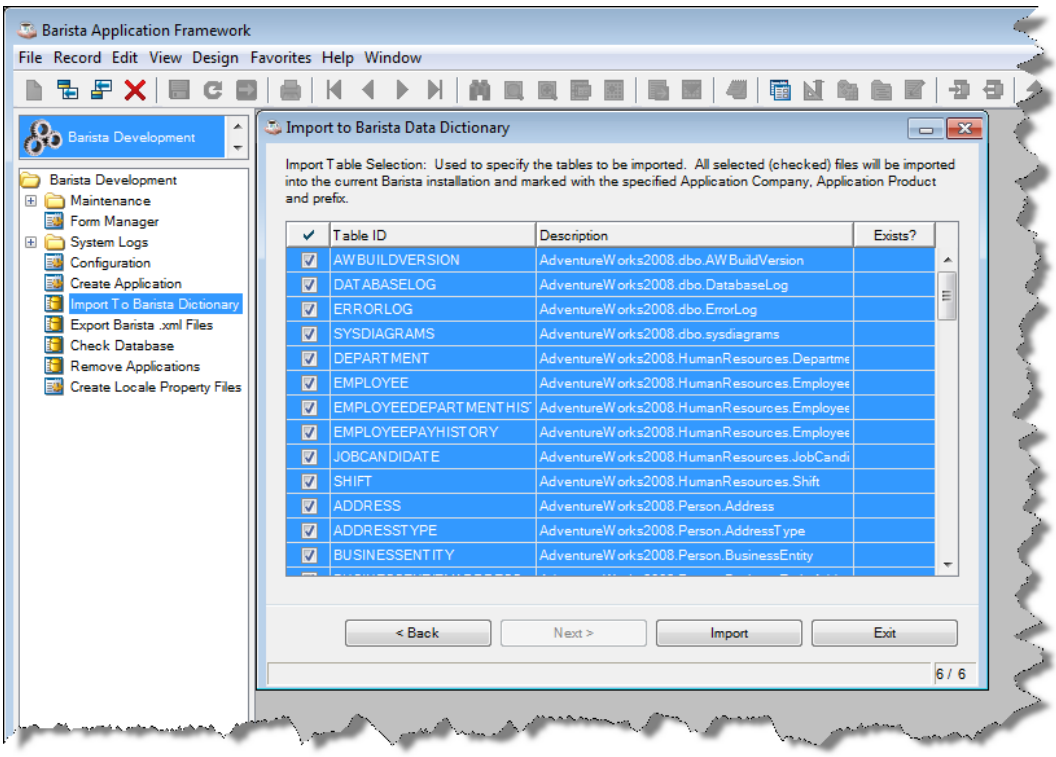
For our purposes, skip over the next four wizard pages by just pressing the [Next] button until the "Import Table Selection" page is displayed with the list of tables found in the SQL database.

The "Table ID" is the simple name of the table (without the application prefix) that is constructed for Barista's use. The "Description" is the path to the table in the database consisting of <catalog>.<schema>.<table>. An unused catalog or schema will display as "<->".

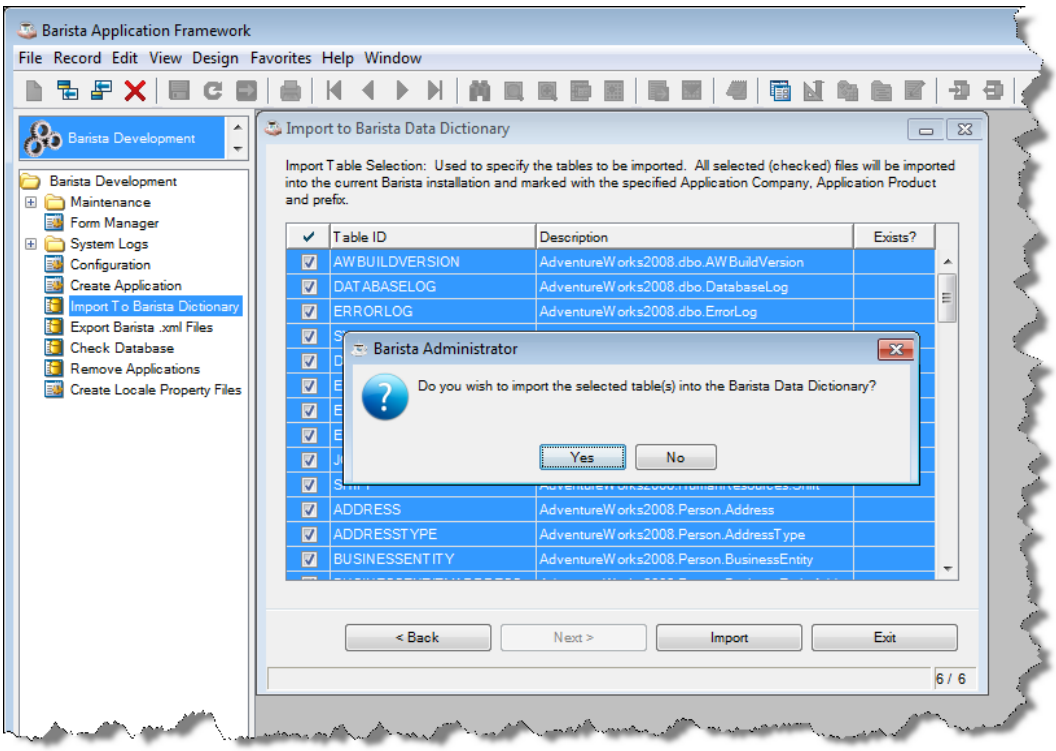
Whether a table already exists in Barista is determined by appending the application prefix to the simple table name and checking to see if there is a table record with this name.

Note: If a table already exists, then a re-import will completely replace the previous contents in the table and table column records. Data elements that already exist will be updated.

Subsequent table changes must be made in the SQL database and re-imported to Barista. It is not currently possible to create or change the structure of SQL tables from within Barista.

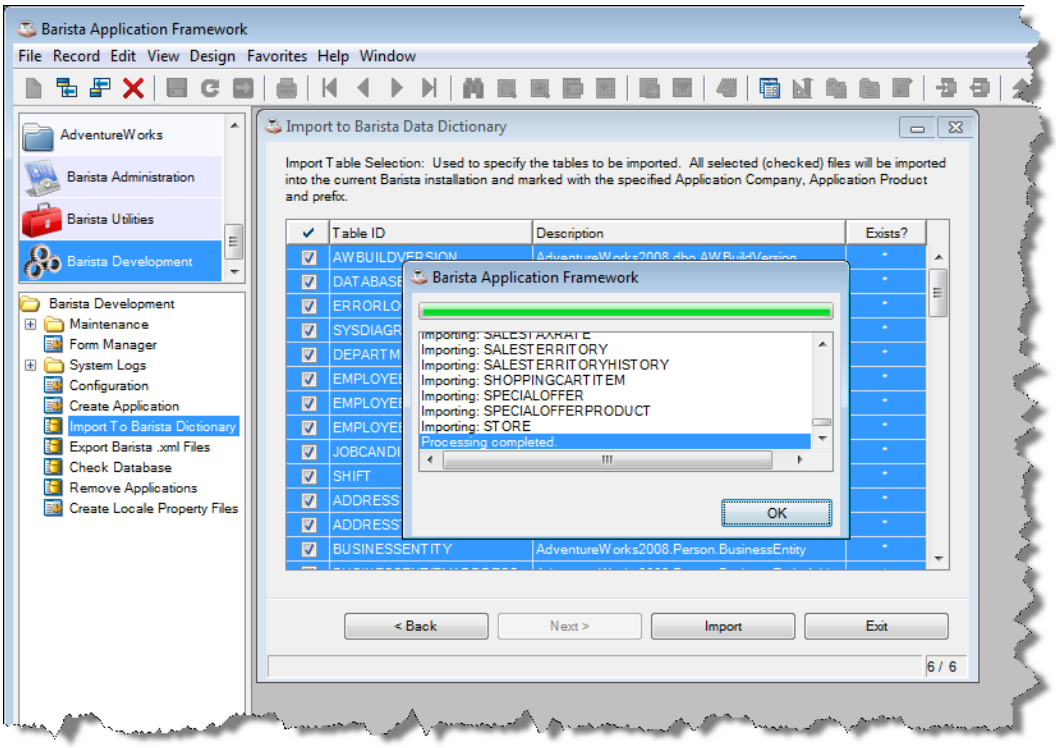


Select the table(s) you wish to import and press [Import].



Click [Yes].

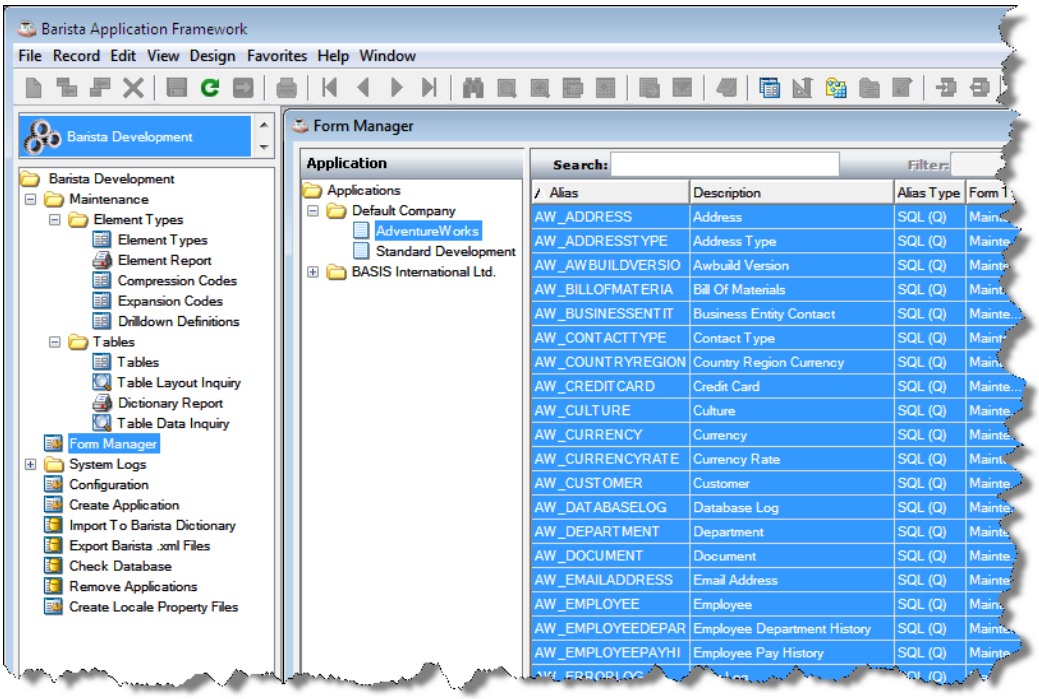
The tables will now be imported, meaning data elements, tables, and table keys (indices) are built. Also, the application menu file is automatically updated. Press [OK] to end the import wizard when the processing is complete.



Barista will automatically build forms as needed, so it is okay to skip the next step and go directly to testing your application from the generated menu system.

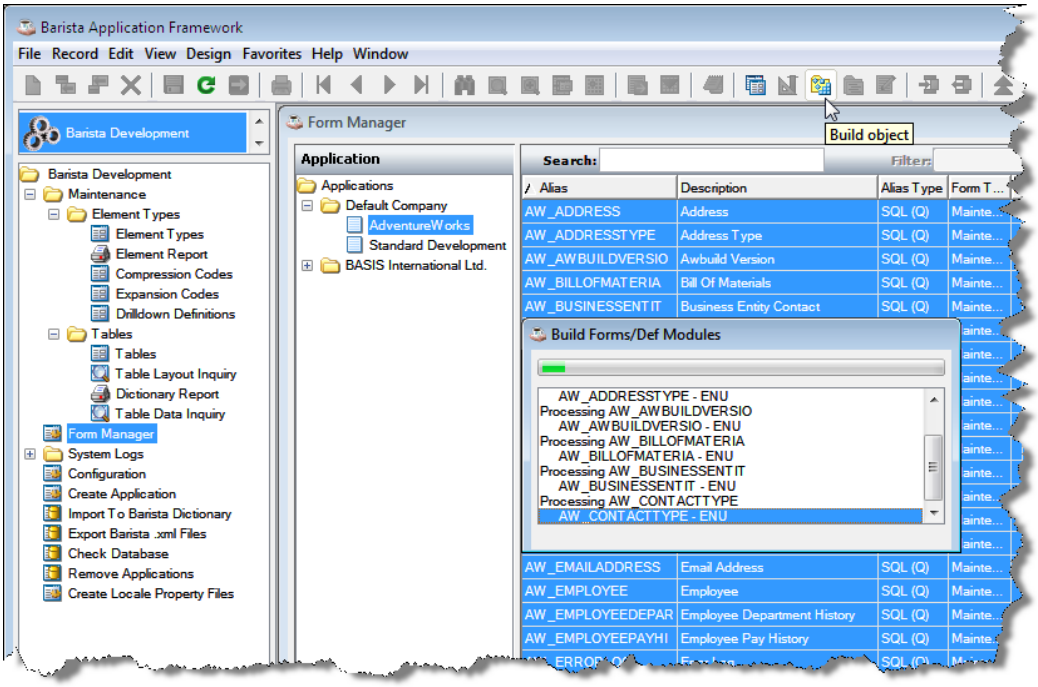
Barista Development → Form Manager

Select your application.

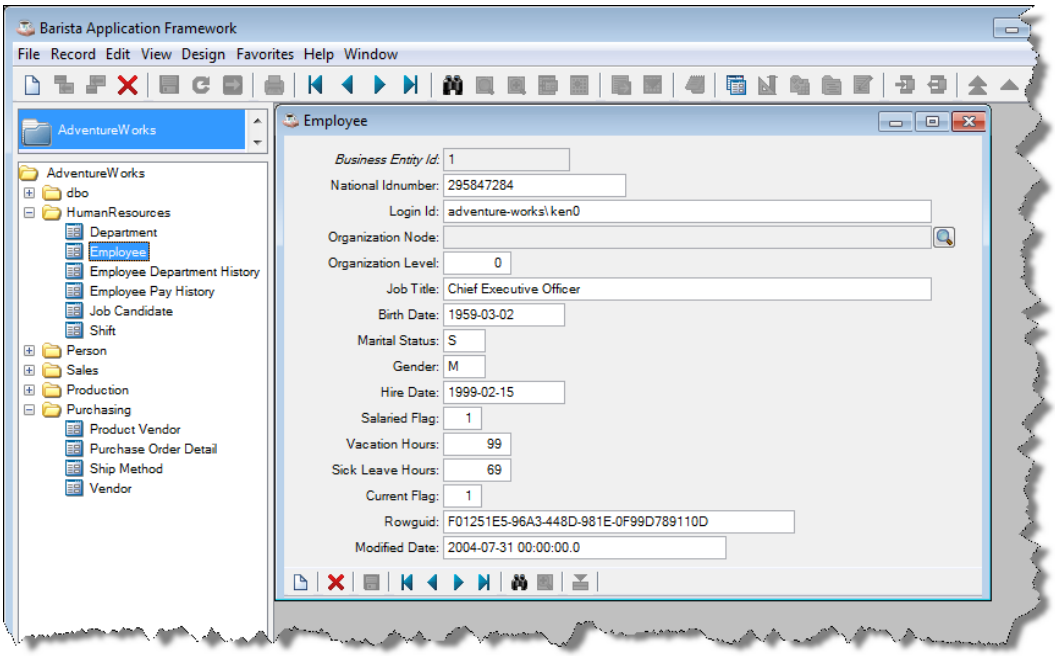


Select all the tables and press the "Build Object" tool button or choose "Design/Build Object" (Ctrl+B) from the Barista menu.

A small progress window will appear showing the build process and disappear when it's finished.



After the forms are built, you can run them from the generated menu system.



Inside the SQL Integration

(back to top)

The BBjdbo (BBj Database Object) class in `bsq_bbjdbo.bbj` can work with any JDBC 2.0 compatible database, including the handling of connections, interrogation of database metadata, and reading and writing data in the database. It wraps many of the `java.sql.*` classes and does not use any of the regular BBj SQL commands, like SQLOPEN() or SQL related BBj infrastructure like the `sql.ini` file.

To manage and reuse JDBC connections, the "+SQL_DBO_MAP" namespace variable contains a BBjdbo object for each active SQL table. The Barista file open programs handle the instantiation as needed. Because each BBjdbo object requires its own result set space, there is no use of the same file channel among multiple processes, as there can be when using the BBj File System.

The SQL database table and column names are stored under the respective Barista table and column for use in all methods that need to communicate with the SQL database. SQL tables that have been imported have a table alias type "Q". Barista programs check this type when I/O needs to occur. If `rd_table_chans$[rd_maint_chan,2]="Q"`, then SQL-related processing occurs instead of the regular BBj I/O processing.

The BBjdbo methods make heavy use of the BBjTemplatedString class. Special methods handle conversions between the BBj/Barista record template and the SQL template. The conversion to/from Barista's character-only keys is also handled in the template conversion methods.

Record navigation occurring in Barista is realized though corresponding SQL result set methods – next(), previous(), first(), last(). (See restrictions below.)

Getting of single record data uses the getRecordByKey() method, returning a BBjTemplatedString that is, in turn, converted into the record as defined in the Barista template. The converse occurs when updating record data.

The simple key value string used in BBj I/O verbs is broken down in the Barista program into its components and passed as a vector of <key column name><operator><value> pairs to BBjdbo methods that need a full or partial key. This key vector is also passed between the main maintenance and inquiry processes as either the selected key from an inquiry or as a filter going in to the inquiry.

At this point, there is very little database-specific (exception) processing. Information from the database metadata has been used as often as possible.

Additional Notes and Restrictions

(back to top)

- The Record navigation for SQL tables is only enabled for tables containing fewer records than the value contained in the configuration variable +DEFAULT_MAXROWS. Otherwise, record selection has to occur via Barista's inquiry function.

- SQL database and table creation and modification from Barista is not yet implemented. SQL tables have to be imported via "Import to Barista Dictionary." With the exception of existing data elements, tables and their column information will be completely replaced when importing.

- All key segment and field values pertaining to a record must be contained within named columns. This means there can be no keys based on an offset and length to data in the record, like there can be when using the BBj File System.

- Barista's table based "Auto-Completion" when entering field values is not yet implemented.

- Barista requires character key types. Numeric key columns in the SQL table will be created as character type keys in Barista with a calculated length and appropriate masking for numeric values during the import process.

- The handling of BLOB, CLOB, XML, and other large database-specific data types is still a work in progress. These columns, although partially visible, cannot be displayed and edited in their entirety in Barista's standard processing.

- SQL database views, stored procedures, triggers and constraints are NOT imported into Barista. Constraint violations not covered by Barista's validation concept will, however, be reported in message dialogs.

- SQL database connections need to be setup for Auto-Commit. Although, the JDBC driver's transaction isolation is implicitly reflected, there is no explicit transaction processing with COMMITs.

- The Barista SQL integration has been tested with the following databases and JDBC drivers:

  ☑ BASIS ESQL (built in to BBj; no additional JDBC driver required when working with BBj/Barista)

  ☑ MySQL 5.0.67-community-nt / MySQL-AB JDBC driver mysql-connector-java-3.1.14

- ☑ MySQL 5.1.31-community-nt / MySQL-AB JDBC driver mysql-connector-java-5.1.7
- ☑ Microsoft SQL Server 9.00.4035 / Microsoft SQL Server 2005 JDBC driver 1.2.2828.100
- ☑ Microsoft SQL Server 10.0.1600.22 / Microsoft SQL Server 2005 JDBC driver 1.2.2828.100
- ☑ Oracle Database 10g Express Edition Release 10.2.0.1.0 / Oracle JDBC driver 10.2.0.1.0XE
- ☑ Apache Derby 10.2.2.1 / Apache Derby Network Client JDBC driver 10.1.2.1

(back to top)