Genero Interface Programming - Class Exercises

Chapter 1 – Genero Clients

- Review the GDC configuration in GST.
- Execute the query application
  - The query program, is the main program we will be working on during the training, it manages data about a group of stores.
- Note the appearance of the query program in it's current state, which is no Genero enhancements.
- Query for a store. Note that querying without providing any search criteria returns all stores, one store at a time.
- Add a store. Make a note of the store number that you assigned to your fictitious store.
- Update details regarding the store you added.
- Exit the query program.
- Switch your client config to 'Web' and run the query program again – test the program in the browser.
- Put the client config back to the 'Desktop'

Chapter 2 – Forms

Chapter 2 - Exercise 1 - Containers

In the query application node, open custform.per

1) Change the legacy syntax of SCREEN to **LAYOUT** and add **GRID** below that. Eg:

```
LAYOUT
GRID
{
…
{
END -- LAYOUT
```

2) Add a **TEXT** attribute to set the title of the top window.

Execute the program again and you should see your title in the title bar.

3) Split the form into two **GRID**s , Using a **VBOX**, the 1st **GRID** should contain just the store number and name, the 2nd will contain the other fields. Finally put a **GROUP** around the first **GRID.**

Now execute the query application, you should see a group box outline around the fields in the first grid

4) Change the **VBOX** container to an **HBOX** container.

You should see the form laid out horizontally instead of vertically.

5) Add the **HIDDEN** attribute to the second **GRID** container.

Recompile the form and run the application. You should NOT see the fields from the second **GRID** container.

6) Change the **HBOX** container back to a **VBOX** container. Remove the **HIDDEN** attribute. Confirm your changes took effect.

Chapter 2 - Exercise 2 - Tables

In this exercise, you will examine and work with tables.

1) Execute the arrays application.
2) Observe how the TABLE container is rendered by the application.

a. Reorganize the table by dragging the **Name** column (store name) to precede the **Id** column (store number).
b. Click on the **Name** column heading to sort the list by the store name.
c. Click the **Cancel** button to close the application without making a selection.

3) Open the manycust.per file for editing.
   a. Locate the TABLE container.
   b. Change the column headings.
   c. Add the UNMOVABLECOLUMNS attribute to the table container.
4) Rerun the application.
   a. Do you see your new headings?
   b. Are you able to rearrange the columns?
   c. Did the GDC preserve the order of the columns from your prior interaction with the program?
   d. Click the **Cancel** button to exit the program.

Chapter 2 - Exercise 3 – Layout Tags

For this exercise return to the query application.

1) Move the State and Zip fields onto a new line under the Address fields.
2) Change the *State* and *Zip* fields so they use a hbox tags and have a spacer between them.
3) Finally, remove the **GROUP** and merge the two **GRID**s so you have only one **GRID** with both sections in, then add a Group around each section using the <G shorthand syntax.

Chapter 2 - Exercise 4 - Widgets

1) Find and review the documentation on the form item types (EDIT item type, BUTTON item type, and so on) located in the **Form Specification File – Attributes** section.

If you are viewing the HTML version of the documentation, click on **Form Specification Files** located under the section **User Interface (1)**. From the resulting **Form Specification Files** page, click **Form file structure** >> **Attributes**.

2) Change the *created_on* field to be a **DATEEDIT** and the *state* field to be a **COMBOBOX** widget. The values for the combobox are IL, IA, and WI.
3) Change the form so it uses labels defined in the attributes section instead of static text in the grid and make the text right justified using the appropriate attribute.
4) Recompile the form and run the application. Add a new store. Use the store number 9xx where xx is your student number. What happens when your cursor enters the state field?

Chapter 2 - Exercise 5 – More Complex Forms

1) Execute the orders application.
2) Enter a new order.
    a. Click the **Order** button.
    b. Click the "zoom button" located in the **Store #** field and select the store you added in the previous exercise.
    c. Make sure the 'Promotions' checkBox is NOT null!
    d. Click the **OK** button.

    A new order is saved in the database. At this time, there are no items associated with your order.
3) Add an item to your new order.
    a. Click the "zoom button" located in the **Stock#** field and select an item from the list.
4) Look at orderform.per to see how this more complex form is defined.
5) Add a total for the Qty column in the items array.
6) Ask your instructor about the default 'demo' that comes with Genero.
    a. In the **Directory** column, find entries identified as **Widgets**. Run the demos for the form item types that interest you.

Chapter 3 – DUI

Chapter 3 - Exercise 1 - Styles

In this exercise, you modify the query application by creating and applying a new window style.

1. In the Studio project file create a new 'library' node called 'etc', in the 'UserInterface' sub project. Add to this the default.4st file from the etc folder on disk.
2. Set the dependencies on the arrays, orders & query program to be dependent on this new library
3. Add a new style to the default.4st file that places both the menu and all action buttons on the bottom of the window.
4. Apply your style in the .per file on the LAYOUT.
5. Compile and run the query application.
6. The menu and action buttons should appear along the bottom of the window.
7. Now make all LABELS have bold text, use the manual to find the specific styles.

Chapter 3 - Exercise 2 - Actions

In this exercise, you define the default actions for your application.

1) Add the default.4ad from etc folder on disk to the 'etc' library node.
2) Look the default actions in the default.4ad file.
   - Add an entry for each action referenced by ON ACTION statements in the query application but not defined in the default action file. For example, while a default action was defined for "insert" and "delete", no default action has been defined for "add" or "quit".

     Also put and image on the 'delete' and 'update' actions which already exists.

     Add text and image attributes to the four new actions.
     NOTE: Genero is using fontAwesume, see the cheatsheet for a list of icons and their names: http://fontawesome.io/cheatsheet/
   - Find the delete action and add an image attribute to it.

NOTE: Setting as many attributes as possible in the default actions file saves you from having to add them later when defining TopMenus and Toolbars.

3) Execute the program.

   Your menu buttons should appear along the bottom of the window (as a result of your style file), each displaying the appropriate image and text.

   If the appropriate image and/or text does not display, check that your action default file uses lowercase names and that you have loaded the correct default action file.

   Note: You can modify the .4ad file and not recompile. Just edit and rerun your program to see your changes. The .4ad file is loaded at runtime!

Chapter 3 - Exercise 3 - ToolBar

Create a Toolbar for the query application by creating a .4tb file and load it in a new function called ui_init().

1) Create your .4tb file as a new file in the 'etc' library node and save it to the etc folder on disk:

   Add an entry for each action listed in the MENU statement of your custmain.4gl module. You do not need to add any other attribute values as they will come from the action default settings.

2) Create a new function called ui_init() at the bottom of the custmain.4gl file.

3) In the ui_init() function, load your Toolbar.

4) Call ui_init() immediately **before** the OPEN WINDOW statement in the MAIN block.

5) Execute the query program and see the new toolbar.

Chapter 3 - Exercise 4 - TopMenu

Create a TopMenu for the query application by creating a .4tm file and loading the file in ui_init() function.

6) Create your .4tm file as a new file in the 'etc' library node and save it to the etc folder on disk :
   - Your TopMenu should have one group with these actions: find, add, update, delete, quit. Use separators to divide them. Make sure all name attribute values are lowercase.
   - You do not need to add additional attributes since you have already defined them in the action default file. However, you might change one to see how the .4tm values take precedence over the defaults.
7) Load the TopMenu:

   In the ui_init() function, add a line of code to load your TopMenu.
8) Compile and run the query program.

   By adding a top menu, the **Find**, **Add**, **Update**, **Delete**, and **Quit** menu buttons have been replaced by the TopMenu options. If you still see any of these buttons, check your .4tm file to ensure that the name of the action in the MENU block is the same as the name attribute in the .4tm file. Again, make sure your name attributes are within quotes and in lowercase.

   In the menu, you still have **Next** and **Previous** buttons.

   In the action panel, you still have the **OK** and **Cancel** buttons.

Exercise 5 - StartMenu

Create a Start Menu that launches the Orders program and the default Genero demos ($FGLDIR/demo/demo.42r).

1) Create the .4sm file as a new file in the 'etc' library node and save it to the etc folder on disk .
2) Load the Start Menu in the ui_init() function.
3) Edit your .4st file to include an entry for startMenuPosition as "tree" and startMenuSize as "small".
4) Change the style of your Start Menu to "poptree" and add a "startMenuShortcut" entry (such as "control-shift-p") to your style. Compare the visual difference between the "tree" and "poptree" styles.

Chapter 4 – Built-In Classes

Chapter 4 - Exercise 1 – Window Title

Change the title of the arrays application using ui.window methods.

What happens if you try this with the query application, which has the title defined in the LAYOUT section of the form?

Chapter 4 - Exercise 2 - ComboBox

Use an initializer function to fill a Combobox with values at runtime:

1) In the query application, change the custform.per file so that the combobox is filled at runtime by an initializer function named set_combo_list.
2) Create a library node of 'uilib' in the project and add a new file of uilib.4gl, this will hold common library functions for your application. Change the dependencies on the query program it's using this library
3) Cut-n-paste the ui_init function from the custmain.4gl into this new library.
4) Add the function, set_combo_list, to the library file. Use ui.Combobox methods in the function to add items to the combobox. Valid values are IL (Illinois), IA (Iowa), WI (Wisconsin)
5) Now change the code to actually fetch the values from the database and populate the combobox with all the states.
   - Create an SQL statement to select the values from the *state* table.
   - Declare a cursor to retrieve the values from the state table.
   - Add the values to the Combobox as items using a FOREACH statement.

Chapter 4 - Exercise 3 – Hide Field

Pinpoint a node and modify one of its attributes (hide the contact_name field and label).

Write a function in uilib.4gl called hide_field(), it should be passed the name of the field to hide and a boolean for hide/unhide.

Use the ui.Form method of setFieldHidden. *Don't forget to call your function!!*

Next, to hide the 'Contact:' label you have to use label widgets instead of just static text in the form. Once it's a label widget it will have a 'name' attribute. We did this change in a previous exercise.

1) Add code to the hide_field function so you can pass the name of the label to hide and use the setElementHidden to hide that label.
2) Recompile the program and run it. You should no longer have a contact name field or label on the form.

Chapter 4 - Exercise 4 – Setup Actions

Change the MENU statement of the query application so that the next, previous, update and delete actions are all enabled/disabled correctly depending on if there is a current data set.

1) Add a function setup_actions to the library file.

   There will be two parameters passed to the function:

   - a ui.Dialog object
   - a variable indicating whether the actions should be disabled or enabled.

   Use the ui.Dialog.setActionActive to enable or disable the next and previous actions based on the value of the variable passed to the function.

2) Call the setup_actions function from the MENU statement in custmain.4gl.

   Disable the actions in a BEFORE MENU statement.

   Add code to the find action to enable the actions if the find is successful (query_ok is TRUE).

Check the documentation for the differences between disabling an action and hiding it.

Chapter 5 – Programming the AUI

Chapter 5 – Exercise 1 – Change Labels

Part A

Create a nodeList and modify the attributes of the nodes in that list (change all the labels to the color RED).

1) Write a new function in uilib.4gl called change_labels().

   Use the om.DomNode method selectByTagName to create a node list of Label nodes. (Since this method will find all the form Labels, both static text and those defined in the Attributes section, you do not have to assign names to all the labels.)

   Then, process each node in the list changing its "color" attribute to "red".

2) Call this function from a new menu option in your custmain.4gl module.

Part B

Change all LABELS to uppercase.

Chapter 5 - Exercise 2 – Extend Form

Part A

Set a default initializer for all forms in the query application to call a new function of forminit.

Part B

Add code to the forminit function to test whether the form being initialized is custform. If so, extend the form by adding a new GROUP to the form.

In the new GROUP add a GRID and in that a LABEL with some text. Look for the first container which may be a VBOX or HBOX, using the method getFirstChild. Once you have found this, add a child "Group" or "Grid" node and then add a "Label" node to that.