Neil J Martin
neilm@4js.com

Genero User Interface Built-in Classes

Genero BDL 3.10
Genero Studio 3.10

FOUR Js
The Power of Simplicity

January 2018

Goals

Learn about the built-in Packages:

- base
- om
- ui

Use the built-in classes and their methods in your programs.

What is a Built-in Class in Genero?

A built-in class is a object type implemented in the runtime system. Example: The channel class

A built-in class belongs to a Package.

A class implements a set of *methods* to manage a specific domain. For example, the DomNode class provides methods to manipulate DOM nodes. Methods can be invoked like global functions, by passing parameters and/or returning values.

Syntax:

package.classname

where *package* is the name of the module comes from
classname is the name of the class

FOUR Js
The Power of Simplicity

Usage

Using a built-in class with a class method:
(No object instantiated)

```
MAIN
    DISPLAY base.Application.getArgumentCount()
END MAIN
```

What is a Built-in Class in Genero?

To use object methods, instantiate a variable of that built-in class with a new object:

- Define a variable.
- Create the object with a class method.

```
MAIN
    DEFINE  d om.DomDocument
    LET d = om.DomDocument.create("Stock")
END MAIN
```

FOUR Js
The Power of Simplicity

The BASE Package

FOUR Js
The Power of Simplicity

The 'base' Package

This package contains base classes of the runtime system.

- **Application**: provides a set of utility functions related to the program environment.
- **Channel**: is a built-in class providing basic input/output functions.
- **SqlHandle**: is a built-in class providing an API to execute parameterized SQL statements, with or without result sets.
- **StringBuffer**: is a built-in class designed to manipulate character strings.
- **StringTokenizer**: is designed to parse a string to extract tokens based on delimiters.
- **TypeInfo**: creates a DOM node from a structured program variable.
- **MessageServer**: allows a program to send a key action over the network to other programs using this service.

Base.Application – No Object Example

The "Application" class is a built-in class providing an interface to the application internals.

This class does not have to be instantiated; it provides class methods for the current program.

Example: Display the command line arguments

```
DEFINE i INTEGER
FOR i=1 TO base.Application.getArgumentCount()
   DISPLAY base.Application.getArgument(i)
END FOR
```

FOUR Js
The Power of Simplicity

Base.Channel – Object Usage Example

The "Channel" class provides basic read/write functionality to access files, pipes or tcp sockets.

Example: open a channel to a file for reading.

```
DEFINE l_mychannel  base.Channel
LET l_mychannel = base.Channel.create()
CALL l_mychannel.openfile(l_fname,"r")
```

The OM Package

FOUR Js
The Power of Simplicity

The 'om' Package

This package provides basic DOM, SAX utilities

Package Classes

- om.DomDocument – An XML document handler
- om.DomNode – An XML node
- om.NodeList – List of DomNodes
- om.SaxAttributes – Attribute list
- om.SaxDocumentHandler – interface to write an XML filter following the SAX standards
- om.XmlReader – Stream reading
- om.XmlWriter – Stream writing

More on XML handling later in the course.

The UI Package

FOUR Js
The Power of Simplicity

The 'ui' Package

This package contains User Interface classes for interacting with the GUI Clieants.

Package Classes

- ui.Interface – The root to the user interface
- ui.Window – Window manipulation
- ui.ComboBox – To handle the ComboBox form item
- ui.Form – Interface to the forms used by a program
- ui.Dialog – Interface to the interactive instructions in a program
- ui.DragDrop – Drag and Drop handling

FOUR Js
The Power of Simplicity

ui.interface

A built-in class to manipulate the user interface.

No object needs to be instantiated; all the methods are class methods.

Example, to show the type and version of the front end client:

```
DISPLAY "Type = " || ui.Interface.getFrontEndName()
DISPLAY "Version = " || ui.Interface.getFrontEndVersion()
```

ui.interface

To load interface elements defined in XML files into the AUI tree:

- loadStartMenu(file STRING) Loads your start menu
- loadToolBar(file STRING) Loads your default toolbar
- loadTopMenu(file STRING) Loads your default topmenu
- loadActionDefaults(file STRING) Loads your default decoration for actions
- loadStyles (file STRING) Loads your default presentation style

Example:

```
CALL ui.Interface.loadToolbar("standard")
```

ui.interface.frontCall

This method of ui.Interface allows you to call functions defined in the front end client. The function will be executed locally on the workstation where the front end client resides.

Syntax:

```
ui.Interface.frontCall(module,func,in-list,out-list)
```

module : name of the module where the function is defined
func : function to be called
in-list : list of the parameters to be passed
Out-list : list of the parameters to be returned

FOUR Js
The Power of Simplicity

ui.interface.frontCall

If you use "**standard**" as the module name, you can call built-in functions defined by default in the front end client:

```
CALL ui.Interface.frontCall(
    "standard",
    "cbset",
    “This is a test”,
    l_ret)
```

cbset is the built-in function to set the contents of the clipboard.
The text to be set is passed.
The result is returned in the variable l_ret.

FOUR Js
The Power of Simplicity

ui.interface.frontCall

Some other built-in functions defined by default in the front end client:

General:

execute, feinfo, shellexec, getenv, mdclose

Dialogs:

opendir, openfile, savefile

Clipboard functions:

cbclear, cbset, cbget, cbadd, cbpaste

ui.interface.frontCall

Calling your own client-side functions:

```
 LET key =
“SOFTWARE\\Microsoft\\WindowsNT\\CurrentVersion“
 LET item = “CurrentVersion“

 CALL ui.Interface.frontCall(
  “getreg",
  “getreg",
  [ key, item ]
  [ val ])
```

NOTE: Only supported the Windows version of the GDC!

FOUR Js
The Power of Simplicity

ui.interface.* More methods

setName(*name* STRING) : sets the name to identify the program on the front-end.
getName() RETURNING STRING : returns the identifier of the program.
setType(*type* STRING) : defines the type of program.
getType() RETURNING STRING : returns the type of the program.
setContainer(*name* STRING) : defines the name of the parent container
getContainer() RETURNING STRING : returns the name of the parent container
getChildCount() RETURNING INTEGER : returns the number of children in this container
getChildInstances(*name* STRING) : RETURNING INTEGER returns the number of children identified by *name*
setSize(x INT, y INT) : Specify the initial size of the parent container window.

setText(*title* STRING) : defines a title for the program.
getText() RETURNING STRING : returns the title of the program.
setImage(*name* STRING) : sets the name of the icon to be used for this program.
getImage() RETURNING STRING : returns the name of the icon.

refresh() : synchronizes the front end with the current AUI tree

ui.Window

Provides an interface for window objects.

Class Methods:

forName(name STRING) RETURNING ui.Window
    Returns the Window object named by OPEN WINDOW statement
getCurrent() RETURNING ui.Window
    Returns the Window object for current window

Object Methods:

findNode(t STRING, n STRING) RETURNING om.DomNode
    Returns Node for tag type of *t* and name of *n*
getNode() RETURING om.DomNode
    Returns node for current window.
setText(t STRING) getText() RETURNING STRING
    Set/Get current window title.
getText() RETURNINNG STRING
    Returns the title of the window object
createForm(n String) RETURNING ui.Form
    Creates an empty form and returns the new form object
getForm() RETURNING ui.Form
    Returns a form object to handle the current form

ui.Window

This example gets the current window, and changes its title:

```
MAIN
  DEFINE l_win ui.Window
  OPEN WINDOW w1 WITH FORM "customer"
  …
  LET l_win = ui.Window.getCurrent()
  CALL l_win.setText("IL Customers")
  …
```

Do Chapter 4 – Exercise 1

ui.ComboBox

The ui.ComboBox class provides methods for populating and interrogating ComboBox objects.

Class Methods:

forName(fieldname STRING) RETURNING ui.ComboBox
setDefaultInitializer(funcname STRING)

Object Methods:

clear()
addItem(name STRING, text STRING)
getColumnName() RETURNING STRING
getItemCount() RETURNING INTEGER
getItemName(index INTEGER) RETURNING STRING
getItemText(index INTEGER) RETURNING STRING
getTableName() RETURNING STRING
getTag() RETURNING STRING
removeItem(name STRING)

FOUR Js
The Power of Simplicity

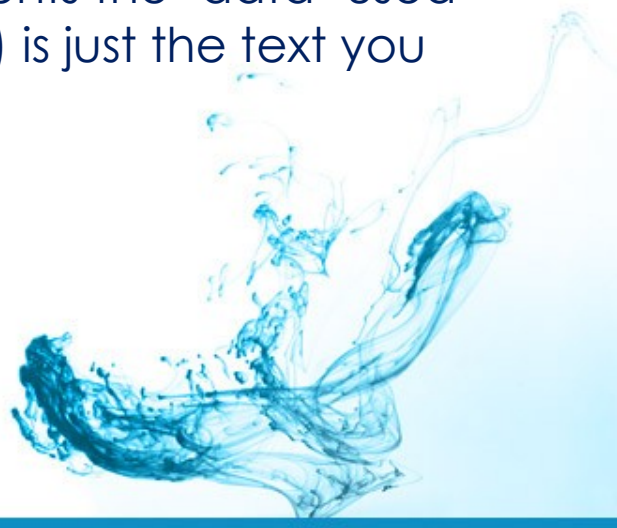
ui.ComboBox

To define an initializer function for a ComboBox:

```
OPEN WINDOW WITH FORM "custform"
CALL cbinit( ui.ComboBox.forName("customer.state") )
…

FUNCTION cbinit(l_cb ui.ComboBox )
  CALL l_cb.clear()
  CALL l_cb.addItem(1,"Paris")
  CALL l_cb.addItem(2,"London")
  CALL l_cb.addItem(3,"Madrid")
END FUNCTION
```

Note: The first value (1,2,3) the 'key' and represents the 'data' used in the 4gl variable. The second value ("Paris" …) is just the text you wish the end users to see.



Do Chapter 4 – Exercise 2

ui.Form

The ui.Form class provides an interface to forms opened with an OPEN WINDOW WITH FORM or DISPLAY FORM instruction in the program.

Class methods:

setDefaultInitializer(fn STRING)

Object methods:

findNode(t STRING, n STRING) RETURNING om.DomNode
getNode( ) RETURNING om.DomNode
getTag() RETURNING STRING
loadActionDefaults(fn STRING)
loadToolBar(fn STRING)
loadTopMenu(fn STRING)
setElementHidden(name STRING, v INTEGER)
setElementText(name STRING, v STRING)
setElementImage(name STRING, v STRING)
setElementStyle(name STRING, v STRING)
setFieldHidden(name STRING, v INTEGER)
setFieldStyle(name STRING, v STRING)

Example: Hiding a Form Element

```
MAIN
    DEFINE w ui.Window
    DEFINE f ui.Form
    DEFINE store_num INTEGER
    DEFINE store_name CHAR(10)
    OPEN WINDOW w1 WITH FORM "nameform"
    LET w = ui.Window.getCurrent()
    LET f = w.getForm()
    INPUT BY NAME  store_num, store_name
      ON ACTION hide
        CALL f.setElementHidden("customer.store_name",TRUE)
      ON ACTION show
        CALL f.setElementHidden("customer.store_name",FALSE)
    END INPUT
 END MAIN
```

ui.Form.setDefaultInitializer()

The **setDefaultInitializer**() class method specifies a default Initialization function for all forms.

```
MAIN
  CALL ui.Form.setDefaultInitializer("my_form_init")
  ...
END MAIN

FUNCTION my_form_init(l_f ui.Form )
   CALL l_f.loadTopMenu("mymenu")
END FUNCTION
```

This function binds a callback function for Form AUI node creation during an **OPEN WINDOW WITH FORM** or **DISPLAY FORM** instruction.

FOUR Js
The Power of Simplicity

Do Chapter 4 – Exercise 3

ui.Dialog

The ui.Dialog class provides an interface to interactive instructions in the program.

To get an instance of this class, use the predefined object variable DIALOG within an interactive instruction in the program or use the ui.Dialog.getCurrrent() method.
The dialog object is only valid during the execution of the an interactive instruction.

Some of the Object methods are:

```
getForm()  RETURNING ui.Form
setActionHidden( name STRING, v INTEGER )
setActionActive( name STRING, v INTEGER )
setFieldActive( name STRING, v INTEGER )
setDefaultUnbuffered( v INTEGER )
setCellAttributes( attarr ARRAY OF RECORD )
```

ui.Dialog

Example: enabling and disabling actions

```
  OPEN WINDOW w1 WITH FORM "custform"
  MENU
    BEFORE MENU
      CALL DIALOG.setActionActive("next",FALSE)
      CALL DIALOG.setActionActive("previous",FALSE)
    ON ACTION find
      CALL query_cust() RETURNING query_ok
      CALL DIALOG.setActionActive("next", query_ok)
      CALL DIALOG.setActionActive("previous", query_ok)
    ON ACTION next
       CALL fetch_rel_cust(1)
...
    ON ACTION quit
      EXIT MENU
  END MENU
```

Do Chapter 4 – Exercise 4

ui.Dialog

The ui.Dialog class also now has methods to create dynamic dialogs:

ui.Dialog.createInputByName
ui.Dialog.createConstructByName
ui.Dialog.createDisplayArrayTo
ui.Dialog.createInputArrayFrom
ui.Dialog.createMultipleDialog
ui.Dialog.addConstructByName
ui.Dialog.addDisplayArrayTo
ui.Dialog.addInputArrayFrom
ui.Dialog.addInputByName
ui.Dialog.addTrigger

```
DEFINE fields DYNAMIC ARRAY OF RECORD
                          name STRING,
                          type STRING
              END RECORD
DEFINE d ui.Dialog
...
LET d = ui.Dialog.createInputByName(fields)
...
```

Summary

A built-in class is a predefined object template that is provided by the runtime system.

Classes are grouped into packages:

- ui - provides user interface classes.
- base - provides the base classes of the runtime system.
- om - provides DOM and SAX utilities.

A class may have two kind of methods:

- Class Methods
- Object Methods

Methods can be invoked in programs like global functions, passing parameters and/or returning values

Q&A

Intelligent Business Application Infrastructure