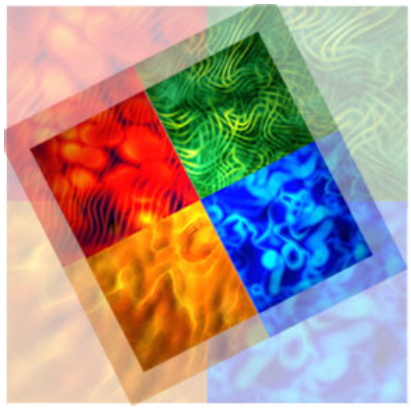
Part IV - Exercises

Jacada Studio for iSeries



Jacada Studio for iSeries

Jacada Studio for iSeries
October 2002

Jacada Studio for iSeries

Tutorial Exercise Overview

This overview will introduce you to the main objectives of each of the exercises in this tutorial. You will then be introduced to the Jacada Studio workflow and the development and runtime architecture models. You will be introduced to the files created by the development process and the connection between the GUI and host development environments. This section acts as a stand-alone reference and glossary of Jacada Studio for iSeries development terminology and exercise objectives. It is not necessary that you learn all of the information in this section by heart before going on to do the exercises. Just remember that you can always use these pages as a reference, when you need to.

* *Note :* This status bar shows your position in the workflow of the tutorial exercises. It is very important that you do the exercises in this tutorial in the order of their appearance in the status bar, in order for you to navigate through your runtime successfully.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

1. Tutorial Exercise Objectives

In the beginning of each exercise, you will find a gray box with a bulleted list of exercise objectives. The following is list of the objectives for all of the exercises in this tutorial.

IDK Walk-Through Objectives

- To introduce the reader to the Jacada Studio for iSeries Interface Development Kit (IDK)
- To familiarize the reader with the basic structure and navigation of the IDK
- To expose the reader to the fundamental components of the IDK including toolbars, palettes, and basic terminology

Your First Application Exercise Objectives

- To introduce the reader to the development process workflow
- To familiarize the reader with the mechanics of constructing and executing of a very simple Studio for iSeries graphical program
- To expose the reader to the wizard-driven design and deployment features and automated code generation capabilities of Jacada Studio

Main Menu Exercise Objectives

- To provide an exercise that replicates the menu application window in the prepackaged iTutor application
- To show one possible way to build a menu window and to explain how it works
- To introduce the reader to the power of pre-built, reusable graphical components
- To explain the relationships between client-side graphical components and host-side navigation and process logic
- Briefly examine host code to understand what was generated and what process logic must be added to complete the program

Add/Edit Resource Exercise Objectives

- To provide an exercise that replicates the Add/Edit Resource window in the pre-packaged iTutor application
- To build a window in which new data can be Added or existing data can be retrieved and Updated
- To examine dynamic GUI display alternatives to Indicator driven Display File behaviors
    - Conditional runtime display driven by client events (based on program mode of Add or Update)
    - Resource # show/hide depending on mode
    - Update or Add button displayed depending on mode
- Passing variable text display values from a host program
- To gain an understanding of the relationship between fields and their graphical representations
- To gain a better understanding of client-controlled application activity as an alternative to host-controlled application activity
    - Data validation at the client or server instead of the host
- Using a graphical link control to launch an external URL in a separate browser frame
- Briefly examine the host code to understand what was generated and what process logic must be added to complete the program

Add/Edit Project Exercise Objectives

- To provide an exercise that replicates the Add/Edit Project window in the pre-packaged iTutor application
- To build a window in which new data can be Added or existing data can be retrieved and Updated
- To examine dynamic GUI display alternatives to Indicator driven Display File behaviors
    - Conditional runtime display driven by client events (based on program mode of Add or Update)
    - Project # show/hide depending on mode
    - Variable Text and behavior (Update or Add) of a single Action button displayed depending on mode
    - Disabling an input capable field at the client based on program mode
- Using methods to communicate the results of data validation performed at the host to the client
- Using code extensions to add advanced GUI controls in XHTML (date control)

Work With Projects Exercise Objectives

- To provide an exercise that replicates the Work with Project window in the pre-packaged iTutor application
- To build a window that illustrates the graphical alternative to Subfile behavior through the use of a graphical table control
- To provide a brief explanation on how to construct and manipulate a table within the IDK
- To provide a capability to re-sequence or reload a table based on a Combobox selection of logical sort sequences
- To show the use of previously used fields with new short-list representations and the difference in use within a table control
- To expose the developer to the use of palette filters to improve usability of the IDK
- To go through the process of creating a new field and assigning an appropriate representation to that field when it is added to the display
- To add several lines of RPG code to implement one of the Jacada Studio table level APIs
- To differentiate Jacada Studio for iSeries table processing from iSeries green-screen Subfile processing
- To explain the different runtime behaviors of record selection between the Java and XHTML clients

2. THE JACADA STUDIO WORKFLOW OVERVIEW

The Jacada Studio for iSeries general workflow is as follows:

In the IDK:

1. Create an Application	Create an application to contain your GUI windows.
2. Build the KnowledgeBase	Build the elements that will be reused throughout the application.
3. Create Subapplication Windows	Create GUI windows and assign Knowledgebase templates.
4. Design Subapplication Windows	Modify the look and feel of your windows for enhanced usability and design.
5. Generate Runtime	Compile GUI application. Create and transfer RPG **shell programs** and **copybooks** to host.

On the iSeries:

6. Start Jacada Monitor	Monitor listens for connection requests from the Jacada Server, initiates new jobs, and delegates sessions to those jobs.
7. Write Host Code	Add Program Logic to Auto-Generated shell program.
8. Compile Program and DDS PF	Compile RPG host program and the DDS physical file copybook.

In the IDK:

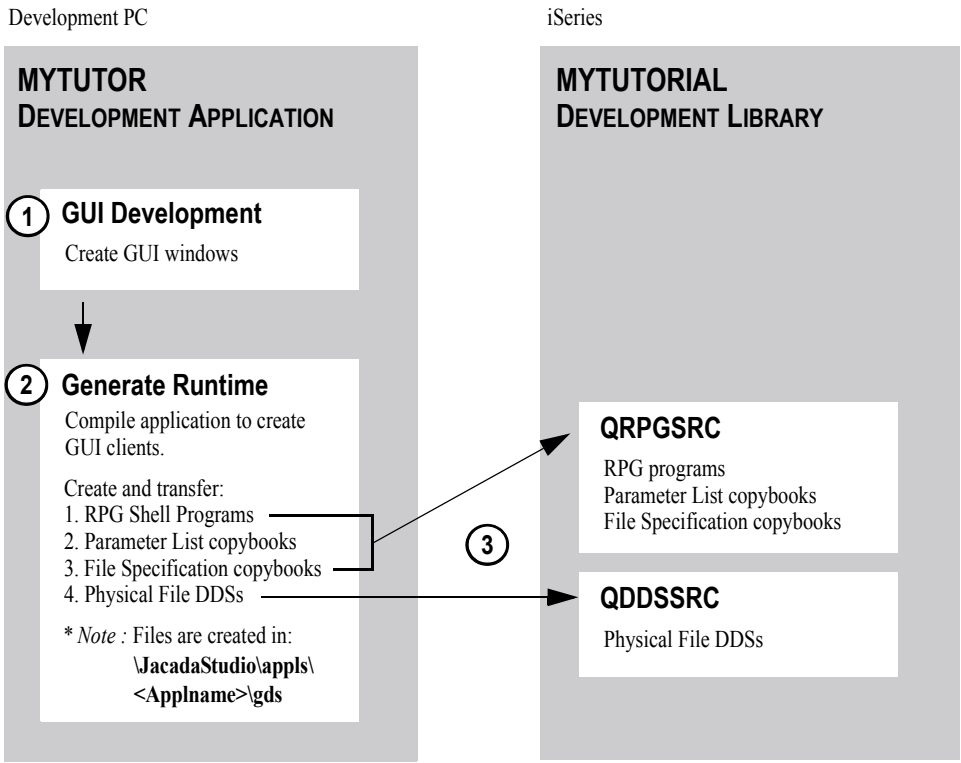
9. Run Application	Run your application with either a Java or HTML client.

3. THE JACADA STUDIO DEVELOPMENT ARCHITECTURE

The name of the application in which you will be doing your development on the PC is **MYTUTOR** The name of the Library in which you will be doing your development on the host is **MYTUTORIAL**. The Jacada Studio Development architecture concept is as follows:

In the IDK:

1. GUI Development
   Create GUI windows
2. Generate a Runtime
   The Runtime Generation process creates:
   RPG shell programs, Parameter List (PLIST) copybooks, File Specification copybooks and DDS physical files.
3. The Runtime Generation process also transfers files to the iSeries and places them in the appropriate libraries / objects.



4. THE SHELL PROGRAM

The Shell Program is a template for the host program that drives a Jacada Studio window. The Shell Program provides a jump-start for the host programmer. It is an automatically generated program that performs the mandatory API actions involved in displaying a Jacada Studio window. This program should be used by the programmer to implement the business logic.

* *Note :* The generated shell program contains all you need to test run an Application. The Read/Write statements in the automatically generated shell programs are very similar to the way you Read/Write to a display file.

Each time a Jacada Studio developer generates a Runtime for an application, a Shell Program for each window within the application is also generated. The Shell Program, when completed, performs the same functions as a standard iSeries interactive application program. The difference is that instead of using a display file to interact with the user's terminal, a special file is used to interact with a graphical client that was created with the Jacada Studio IDK. In addition, the program runs in batch mode.

Location of the Shell Program

After Runtime Generation, the Shell Program is found in the directory **<Jacada Studio Root Path>\appls\<applname>\gds.**

The file name is ShellProgram.RPG_OPM.<window name> or ShellProgram.RPG_ILE.<window name>, depending on the version of RPG specified in the wizard when the application was created.

5. OTHER GENERATED FILES RELEVANT TO THE SHELL PROGRAM

There are three other files created in the runtime generation process. These files are used by the automatically generated shell program for the following purposes:

Files related to the window

The following three files are created for every window, in addition to the Shell Program:

File Name	File Type and Usage	Description
RPG_xxx.<saname>$D	DDS code for window	Buffer definition for the window. Jacada Studio generates one such file for each window. This file defines one record, named <WindowName>B, which holds a list of the fields defined in the window. Gets copied to the QDDSSRC file, and must be compiled.
RPG_xxx.<saname>$F	File specification copybook for window	Contains RPG F-specs. Defines an external, 'special' file. Links the window with a special file definition and with a Parameter List (PLIST) definition. Gets copied to the QRPGSRC file.
RPG_xxx.<saname>$P	Parameter List (PLIST) copybook for window	Parameter List (PLIST) definitions. These are RPG C-specs. Gets copied to the QRPGSRC file.

Files related to tables

If the window includes a table, three additional files are generated:

File Name	File Type and Usage	Description
RPG_xxx.<saname>#D	DDS code for table	Buffer definition for one table row. Jacada Studio generates one such file for each table in the window. This file defines one record, named <TableName>B, which holds a list of the fields defined in the row. Gets copied to the QDDSSRC file, and must be compiled.
RPG_xxx.<saname>#F	File specification copybook for table	Contains RPG F-specs. Defines an external, 'special' file. Links the table with a special file definition and with a Parameter List (PLIST) definition. Gets copied to the QRPGSRC file.Same as the preceding file, but is created only if the window contains a table.
RPG_xxx.<saname>#P	Parameter List (PLIST) copybook for tableParameter list definition.	Consists of RPG C-specs. Gets copied to the QRPGSRC file.

The files are generated in the directory: **<Jacada Studio Root Path>\appls\<applname>\gds.**

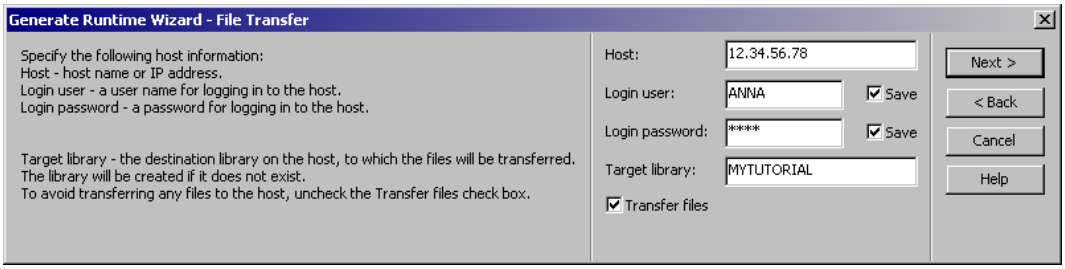
For Example:

You create an application called APPL01 to run in conjunction with RPG OPM programs. You name one of the windows SCR01. This window includes a table, which you call TAB01. When you generate a runtime for this application, the following files are created for window SCR01:

File name	Description
RPG_OPM.SCR01#D	DDS statements for table TAB01
RPG_OPM.SCR01#F	F-spec copybook for table TAB01
RPG_OPM.SCR01#P	Parameter List copybook for table TAB01
RPG_OPM.SCR01$D	DDS statements for window SCR01
RPG_OPM.SCR01$F	F-spec copybook for window SCR01
RPG_OPM.SCR01$P	Parameter List copybook for window SCR01
ShellProgram.RPG_OPM.SCR01	Shell Program for the application.

Installing the Shell Program

Each Shell Program and its associated copybooks must be moved to the iSeries. This is accomplished automatically by the FTP process that occurs as part of Runtime Generation. At Runtime Generation, in the "Specify Host Information" dialog box, the wizard will ask you for the information it needs to move the files. You will be walked-through this wizard later.



While we recommend that you let Jacada Studio's Runtime Generation wizard copy the Shell Program and its associated files to the iSeries, you do have the option of copying the files yourself.

To install the Shell Program:

1. Add code in the Shell Program to process the incoming data and to return the appropriate response to the terminal operator.
2. In file `QDDSSRC`, compile the `<WindowName>$D` member, and the `<WindowName>#D` member if the window has a table.
3. Compile the Shell Program.

Keeping Buffer Definitions in Sync

Be aware that modifying a window or table layout in Jacada Studio can change the total size of the associated window data buffer or table data buffer. An example of this would be if you added a new field to a screen or table, or deleted or changed the length (size) of an existing field, as shown in the Window Fields palette.

After such a change, it is important to remember to **recopy** the affected DDS file and Parameter List copybook to the iSeries machine, and to **recompile** the DDS file and RPG program associated with the window. Otherwise there will be a discrepancy between the size and/or layout of the buffer used by Jacada Studio compared to the size and/or layout of the buffer expected by your supporting RPG programs on the iSeries, which may lead to unpredictable results.

The Generate Runtime wizard will take care of recopying the DDS files and Parameter List copybooks to the iSeries, but you must remember to recompile the DDS file and the Shell Program.

Contents of the Shell Program

This sample program is taken from the ITUTORIAL Demo application that is included in the package you downloaded from the Internet. The code in the shaded portion of the program was added to the basic shell program by the programmer. Comments in Ariel Narrow font have been added for your benefit. The Alpha references in the sample program are explained on page 10.

```
=================================================================
   * ------------------------------------------------------
   * This code was generated automatically by Jacada.
   *    Sub-Application: PMENU
   *    Time generated:  Tue May 28 15:08:01 2002
   * ------------------------------------------------------
   *

   * Import the file specifications for the current window.
A  F/COPY PMENU$F              << This copybook was created at Runtime Generation
   * Place other file specifications imports here.
   * No other file specs needed for this program
   *
   * Import GDS E specifications
B  E/COPY JRPGSRC,GDSESPECS    << This copybook is installed with Jacada Studio.
   * Place your E specifications here.
   * No additional E-specs needed for this program
   *
   * Import GDS I specifications
C  I/COPY JRPGSRC,GDSISPECS    << This copybook is installed with Jacada Studio.
   * Place your I specifications here.
   * No other I-specs needed in this particular program
   *
   * Import the parameters list defined for the window.
D  C/COPY PMENU$P              << This copybook was created at Runtime Generation
   *
   *
   * ------------------------------------------------------
   * Start of main loop.
   * ------------------------------------------------------
   *
   * Place your code for refreshing the window's data here.
   * There's no "refreshing" to be done for this particular screen
   *
   *


   * This following line is added to Perform Loop until JSTACT = 'EXIT'. This will occur when the user presses
   * the "EXIT" button.
    C            JSTACT    DOUEQ'EXIT'
   *
   * Display the menu screen
   *
E  C                       WRITEPMENUB
   * Read the menu screen
   C                       READ PMENUB                   99
    *
   * Check JSTACT to see what button the user pressed. JSTACT was filled in by the method called
   * 'ActionPerformed' which is associated with each of the buttons on the main menu.
   *
   C* If 'Exit' was not clicked...
   C            JSTACT    IFNE 'EXIT'
   C*
   C* If 'Work with Projects' was clicked...
   C            JSTACT    IFEQ 'WWP'        JSTACT was set to WWP by the method
   C                      CALL 'PPROJ'       so call prog PPROJ
   C                      MOVEL*BLANKS   JSTACT
   C                      ENDIF
    C*
   C* 'Work with Resources' clicked
   C            JSTACT    IFEQ 'WWR'        JSTACT was set to WWR by the method
   C                      CALL 'PRESO'       so call prog PRESO
   C                      ENDIF
    C*
```

```
  C* 'Add Project' clicked
  C            JSTACT    IFEQ 'AP'           JSTACT was set to AP by the method
  C                      MOVEL'ADD'     MODE    5         set mode to ADD
  C                      CALL 'PADDPR'       call pgm PADDPR, pass 3 parms
  C                      PARM           MODE
  C                      PARM           PRO#    50
  C                      PARM           MENU    5
  C                      ENDIF
   C*
  C* 'Add Resource' clicked
  C            JSTACT    IFEQ 'AR'
  C                      MOVEL'ADD'     MODE1   5
  C                      Z-ADD0         RES#    50
  C                      CALL 'PADDRE'
  C                      PARM           MODE1
  C                      PARM           RES#
  C                      PARM           MENU
  C                      ENDIF
  C*
  C                      ENDIF                            IFNE EXIT
  C*
  C                      ENDDO                            DOUE EXIT
   C*
  C                      SETON                      LR
    *
    * ------------------------------------------------------
    * End of main loop.
    * ------------------------------------------------------
    *
  C            *INZSR    BEGSR
    *
    * Place your initialization code here.
    * There's no special initialization code in this menu program.
  C                      ENDSR
================================================================================
```

Copybook PMENU$F is used by program PMENU

```
================================================================================
     F* Generated at: Sun Jun 30 12:38:59 2002
     FPMENU$D CF  E                    SPECIAL       GDSRAP
     F                                              KPLIST PMENU
================================================================================
```

Copybook PMENU$P is used by program PMENU.

```
================================================================================
     C* Generated at: Sun Jun 30 12:38:59 2002
     C           PMENU     PLIST
     C                     PARM 1         GDSPLV         COPYBOOK VERSION
     C                     PARM 'W'       GDSTYP         OBJECT TYPE
     C                     PARM 'PMENU '  GDSNAM         WINDOW NAME
     C                     PARM 152       GDSLEN           BUFFER LENGTH
     C           *IN       PARM *IN       GDSIND           INDICATORS
     C                     PARM           GDSRC            RETURN CODE
     C                     PARM 0         GDSLC            LISTS COUNT
     C                     PARM           LN               LIST NAMES ARY
     C                     PARM           GDSEXT           GDS EXT. INFO.
================================================================================
```

File PMENU$D defines the menu screen buffer for program PMENU.

```
=====================================================================
     A* HEADER FOR RECORD 'MAIN' OF SUB APPLICATION 'PMENU'
     A* THIS MEMBER WAS AUTOMATICALLY GENERATED BY JACADA
     A* AT Thu Jun 20 14:22:44 2002
     A* DO NOT MODIFY THIS MEMBER.
     A* (C)COPYRIGHT JACADA
     A*
     A          R PMENUB
     A            JSTACT        10A
     A            JSTMSG        80A

=====================================================================
```

The Following Comments Describe the Shell Program Example Above (PMENU):

1. The elements that will normally appear in every Shell program are:
   - A - Copy statement(s) for the File Spec copybook(s) for the window's screen(s) (item "A" in the listing)
   - B - Copy statement(s) for one or two special Jacada Studio copybooks, depending on RPG language version.
   - C - For RPG OPM these are `GDSESPECS` and `GDSISPECS` (items "B" and "C" in the listing); for RPG ILE there's just `GDSCOMMON`.
   - D - Copy statement(s) for the Parameter Lists associated with the window (item "D").
   - E - A write and read (via the API) of the screen (item "E" in the listing).
2. If the screen included a table, and additional copy statement for the table's file specification would have been inserted after item "A". Additional code would also have been generated in the form of basic subroutines to clear, initialize, and read user input from, the table.
3. Item "B" is a copy statement for `GDSESPECS` copybook. This copybook is provided with Studio and is required in all programs using the API for RPG OPM.
4. Item "C" in the sample program is a copy statement for the `GDSISPECS` copybook. This copybook is provided with Studio and is required in all programs using the API for RPG OPM.
5. The `WRITE` and `READ` statements at "E" invoke the RPG API. Note that they were automatically inserted into the shell with the name of the screen file as defined in the "DDS" statements in file `'RPG_xxx.<saname>$D'`.

Adding your Code to the Shell Program

Remember that the Shell Program as generated by Jacada Studio is not enough to accomplish meaningful work for you. In order to do that, you must add the business logic code to it. Of course, the code that you add will depend entirely on the specific requirements of your window.

To help you understand the sorts of additions you need to make to your Shell Programs, we've included two sample Shell Programs that were modified to work with specific windows in Appendix A of the API document. Like the program listed above, the windows in Chapter 3 of the API document are also part of the Jacada Studio evaluation. You can also find the source code for these programs and the rest of the RPG programs associated with this Jacada Studio evaluation, in the RPG source libraries for the iSeries that were installed.

Controlling the Initial Contents of the Shell Program

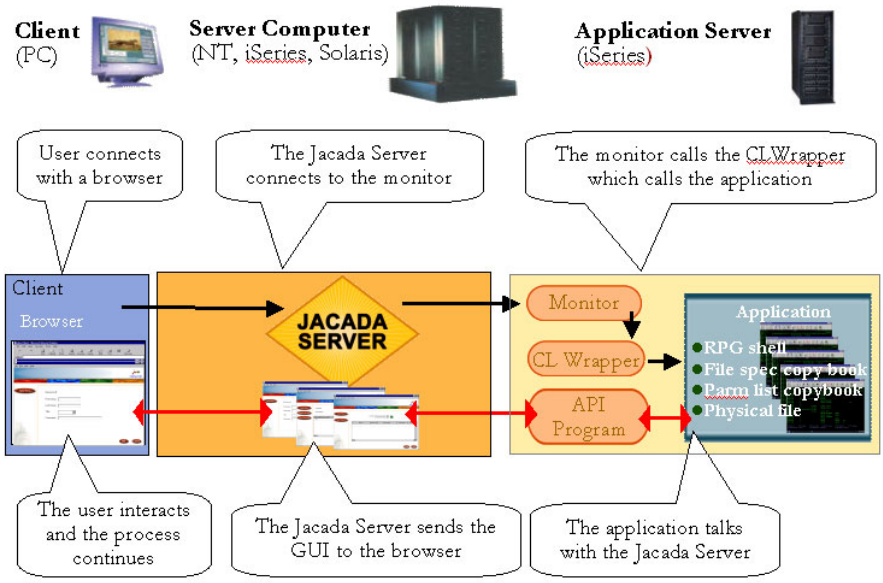
The Shell Skeleton controls the statements that will be generated in the Shell Program of every window. Besides the required copybooks, there may be common subroutines or comment blocks required in every program at your particular shop. Rather than enter them manually in each application's Shell Program, you can enter them once in the Shell Skeleton and they will be automatically included in the generated Shell Programs.

The Shell Skeleton is called ShellSkeleton.RPG_ILE or ShellSkeleton.RPG_OPM, and is located in the **<Jacada Studio Root Path>\appls\<applname>** directory.

* *Note :* **Do not change the order of the copybooks** as they appear in the original version of the Shell Skeleton.

6. THE JACADA STUDIO RUNTIME ARCHITECTURE

The following diagram shows the Jacada Studio Runtime architecture:



7. WHAT YOU CAN EXPECT

The following table shows an overall picture of the files that you can expect to encounter throughout this tutorial, and their location in the development architecture. Some of the files have been prepackaged and installed by the installation process and some you will create for yourself in the semi-built MYTUTOR application. The **GUI Information** columns show files that were either prepackaged or you will create in the IDK on your development machine. The **iSeries Files** columns show the files that were either prepackaged or you will be created during the runtime generation process and transferred to the iSeries.

	GUI Information			iSeries Files					
Window Title	Application Name	Window Type	Window Name	RPG Library	RPG Program / Shell	Records Window / Table	Copybook File Specs / Table Specs	Copybook Parm Lists	Exercises
You will create in FIRSTAPP				**You will create in FIRSTAPP**					
[None]	FirstApp	Output	HelloW	FIRSTAPP	HELLOW	HELLOW$D	HELLOW$F	HELLOW$P	1
Prepackaged with ITUTOR; you will create in MYTUTOR				**Prepackaged with ITUTORIAL; you will create in MYTUTORIAL**					
Main Menu	iTutor	Menu	PMENU	ITUTORIAL	PMENU	PMENU$D	PMENU$F	PMENU$P	2
Add/Edit Resource	iTutor	I/O	PADDRE	ITUTORIAL	PADDRE	PADDRE$D	PADDRE$F	PADDRE$P	3
Add/Edit Project	iTutor	I/O	PADDPR	ITUTORIAL	PADDPR	PADDPR$D	PADDPR$F	PADDPR$P	4
Work with Projects	iTutor	Table	PPROJ	ITUTORIAL	PPROJ	PPROJ$D	PPROJ$F	PPROJ$P	5
						PPROJ#D	PPROJ#F	PPROJ#P	
Prepackaged with ITUTOR & MYTUTOR				**Prepackaged with ITUTORIAL & MYTUTORIAL**					
Project Assignments	iTutor	Table	PASSI	ITUTORIAL	PASSI	PASSI$D	PASSI$F	PASSI$P	
	MyTutor			MYTUTORIAL		PASSI#D	PASSI#F	PASSI#P	
Display Resource Assignments	iTutor	Table	PDRESA	ITUTORIAL	PDRESA	PDRESA$D	PDRESA$F	PDRESA$P	
	MyTutor			MYTUTORIAL		PDRESA#D	PDRESA#F	PDRESA#P	
Work with Resources	iTutor	Table	PRESO	ITUTORIAL	PRESO	PRESO$D	PRESO$F	PRESO$P	
	MyTutor			MYTUTORIAL		PRESO#D	PRESO#F	PRESO#P	
Assign New Resource to Project	iTutor	Table	PASRSC	ITUTORIAL	PASRSC	PASRSC$D	PASRSC$F	PASRSC$P	
	MyTutor			MYTUTORIAL		PASRSC#D	PASRSC#F	PASRSC#P	

Columns in the GUI Information Section

Application Name	Name of the IDK application in which this window resides or will be created.
Window Type	Type of window (e.g. I/O, Table, Output).
Window Name	Name of the window in the IDK.

Columns in the iSeries Files Section

RPG Library	Name of the library in which the files associated with the window reside or will be created.
RPG Program / Shell	Name of the RPG Program associated with the window (preexisting or will be created).
Records Window / Table	Name of the DDS Physical File associated with the window (preexisting or will be created).
Copybook - File Specs / Table Specs	Name of the File /Table Specification copybook associated with the window (preexisting or will be created).
Copybook - Parm Lists	Name of the Parameter List copybook associated with the window (preexisting or will be created).
Exercises	Number of the tutorial exercise in which the window will be built.

Jacada Studio for iSeries

Walk-Through of the IDK Interface

Objectives:

- To introduce the reader to the Jacada Studio for iSeries Interface Development Kit (IDK)
- To familiarize the reader with the basic structure and navigation of the IDK
- To expose the reader to the fundamental components of the IDK including toolbars, palettes, and basic terminology

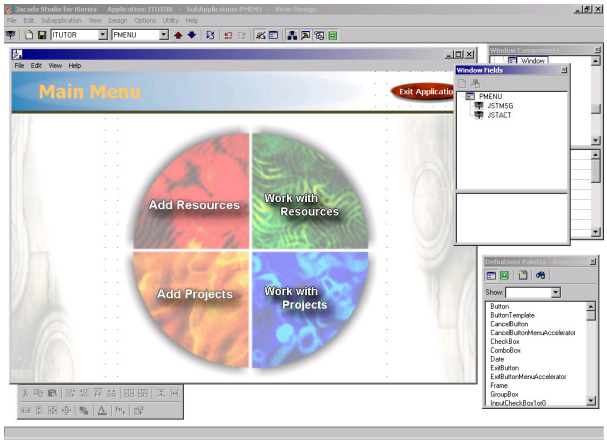
In this section, you open the Jacada Studio for iSeries Interface Development Kit (IDK) for the first time and get acquainted with the IDK interface. You open the ITUTOR demo Application and use it to identify the different features of the IDK interface. During this introduction to the IDK interface, it is important that you not make any changes to the Application, so that it will run it properly later in the Application Walk-through section. So, if at any point in the IDK overview you are prompted by a message box to save your Application, reply to the prompt by clicking the **No** button. This section also acts as a stand-alone reference and glossary of IDK interface features and terminology. It is not necessary that you learn each of the interface elements in this section by heart before going on to do the exercises. Just remember that you can always use these pages as a reference, when you need to.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

In this section you learn about:

1. Opening the Jacada Studio for iSeries IDK
2. Opening the ITUTOR Application
3. The Application Combobox
4. The SubApplication Combobox
5. IDK Views
6. IDK Menus
7. The Standard Toolbar
8. The KnowledgeBase
9. Apply Design Changes
10. The Design View Palettes
11. Setting Up Your Workspace
12. Control Editing and Manipulation Options

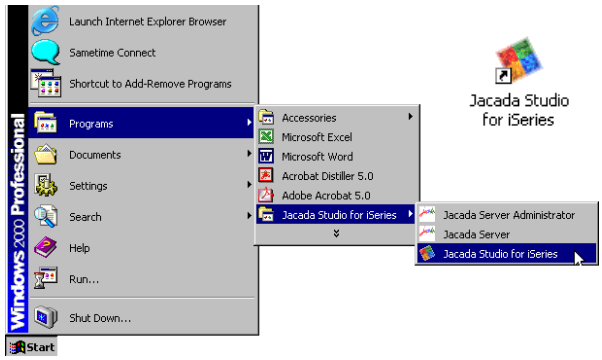


The PMENU Subapplication of the ITUTOR demo application open in the IDK Interface.

1. OPENING THE JACADA STUDIO FOR ISERIES IDK

To open the IDK interface after installing the product:

From your Windows **Start** menu > choose **Programs** > **Jacada Studio for iSeries** > **Jacada Studio for iSeries**.
The Jacada Studio Interface Development Kit (IDK) is invoked.

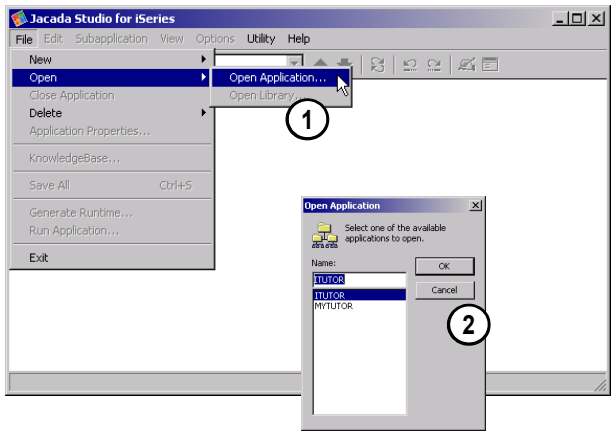


From your Windows Start menu, navigate to the Jacada Studio for iSeries icon to bring up the IDK interface.

2. OPENING THE ITUTOR APPLICATION

To open the ITUTOR Application from within the IDK interface:

1. From the IDK **File** menu > choose **Open** > **Open Application**.
   The Open Application Dialog is invoked.
2. In the **Open Application Dialog** >
   Select ITUTOR from the Application Name List >
   Click OK.



Choose File > Open > Open Application. Select ITUTOR from the Application name List and click OK to open the ITUTOR application.

3. THE APPLICATION COMBOBOX

The IDK Standard toolbar now shows that the ITUTOR is the active Application in the both the **Titlebar** and the **Application Combobox**.

Both the IDK Titlebar and the Application Combobox show that ITUTOR is the active application

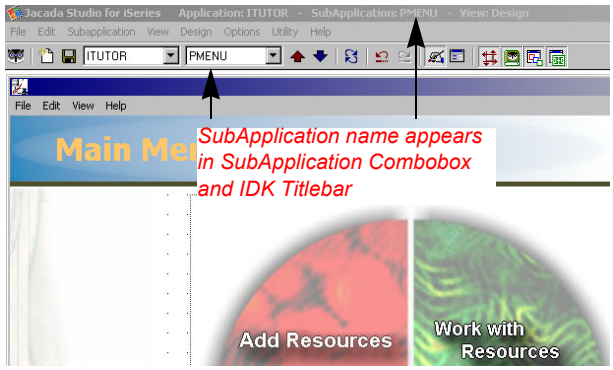
4. THE SUBAPPLICATION COMBOBOX

Now we will open the first of the pre-built Subappplications in the ITUTOR demo Application.

The term "Subapplication" is used to describe the contents of a GUI Window in the Studio IDK. Each window is referred to as a Subapplication. Subapplications consist of GUI Elements, Host Fields and the links between them.

To open the PMENU Subapplication in the ITUTOR Application:

From the **Standard Toolbar** > open the **Subapplication Combobox** > select the **PMENU** Subapplication.
The PMENU window is opened within the IDK interface. The Subapplication combobox and IDK Titlebar show that PMENU is the active Subapplication.



* *Note :* The PMENU Subapplication is the menu window, the first window that you will see in the demo Application runtime.

The Subapplication List

The Subapplication Combobox shows a list of all of the Subapplications that exist in the current Application.

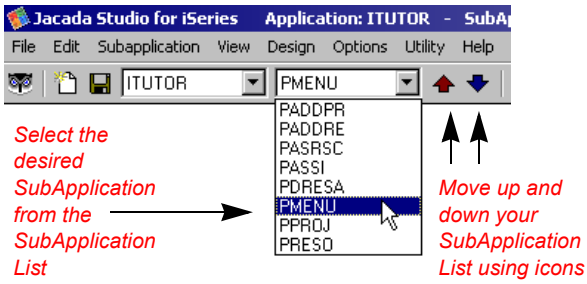
To open an existing Subapplication:

From the **Standard Toolbar** >
Select the Subapplication from the Subapplication List in the **Subapplication Combobox**.

or

From the **Standard Toolbar** >

Use the  and  icons to move up and down in your Subapplication List.



To open an existing SubApplication, In the standard toolbar > select the desired SubApplication from the SubApplication Combobox or use the icons to move up and down in your SubApplication List

* *Note :* When moving between subapplications, you receive a message box prompting you to save your application. If you switch focus between Jacada Studio and another open application when a message box dialog is open in the Jacada Studio IDK, this message box disappears behind the open subapplication window.
To retrieve it, **while holding down the Alt key:**

1. Press the Tab key to bring up a windows dialog box showing all active tasks.
2. Press Tab again as many times as necessary to select the windows icon.
3. Release the Alt key and you will see the message box.

5. IDK VIEWS

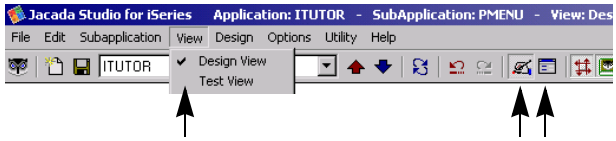
The PMENU Subapplication is opened to *Design View* by default. IDK has two working views, *Design View* and *Test View*. You can switch between these two views in the **Standard Toolbar** via:

The Design View Icon 

or

The Test View Icon 

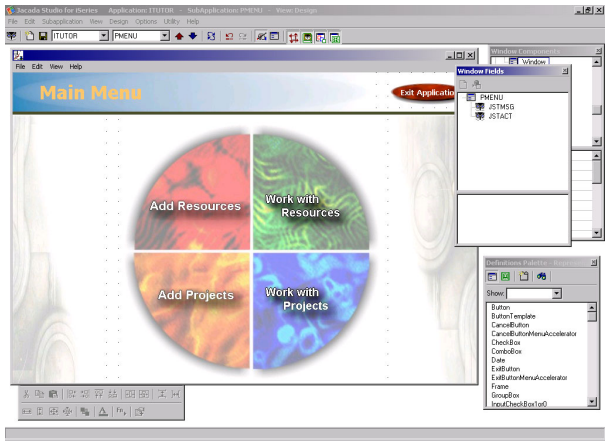
You can also switch between the two views via the *View Menu*.



Switch between Design View and Test View via the View Menu or the View Icons on the Standard Toolbar.

Design View

Design View is the IDK view in which you build and modify your Application. Here you create and modify your GUI window and its contents.

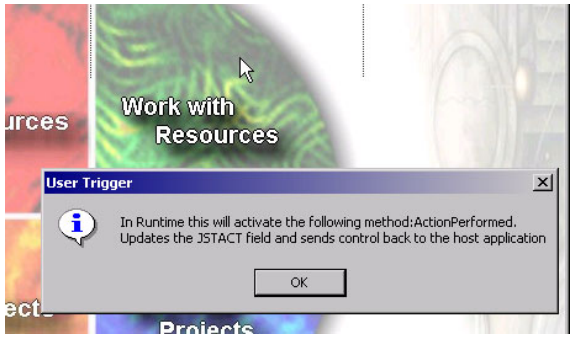


The PMENU Subapplication open in Design View of the IDK Interface.

Test View

Test View is the IDK view in which you test the functionality of your Subapplications virtually, without running an actual runtime. In Test View, you can see the functionality attached to GUI elements in your window.

For example: Clicking on one of the menu buttons in your Subapplication will bring up a dialog that explains the functionality associated with the menu button.

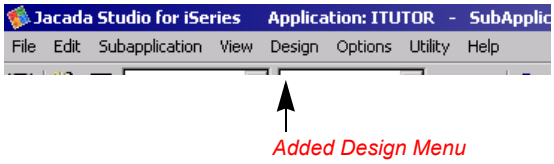


Clicking on one of the menu buttons in your SubApplication will bring up a dialog that explains the functionality associated with the menu button.

6. IDK MENUS

The Menus in the Jacada Studio for iSeries IDK are context sensitive; they change according to the view that you are in. A *Design menu* is added when you are in Design View. This menu does not appear when you are in Test View.

Available Menu Options in Test View.

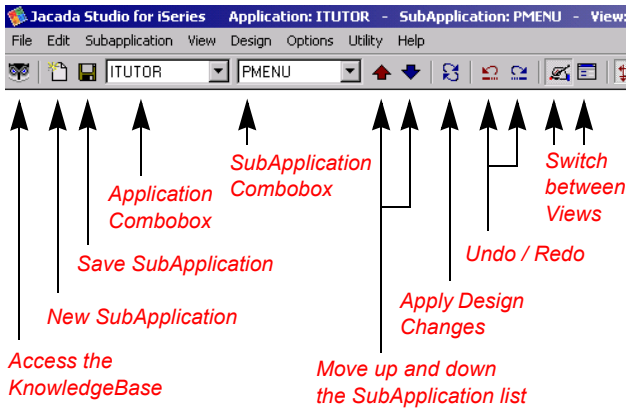


Available Menu Options in Design View.

7. THE STANDARD TOOLBAR

The IDK Standard Toolbar is also context sensitive, and changes according to the view that you are in. From the Standard Toolbar you can:

Click the	icon to	Access the KnowledgeBase
Click the	icon to	Create a New Subapplication
Click the	icon to	Save your Subapplication
Click the	icon to	Move up in your Subapplication List
Click the	icon to	Move down in your Subapplication List
Click the	icon to	Apply Design Changes
Click the	icon to	Undo your last operation
Click the	icon to	Redo your last operation
Click the	icon to	Switch to Design View
Click the	icon to	Switch to Test View



Various Components of the IDK Standard Toolbar

8. THE KNOWLEDGEBASE

The KnowledgeBase is the place where rules regarding the properties of IDK elements are stored. If you create a definition once in the KnowledgeBase, instances of the knowledgebase definition can be reused throughout the Application. We refer to changes made in the KnowledgeBase as **global modifications**, since they effect all instances of the modified object across the Application. The KnowledgeBase instances can be modified locally in *Design View*. Modifications made in Design View override the properties of KnowledgeBase definitions. Modifications made in Design View are considered **local modifications**, since they only effect the one specific instance that was modified.

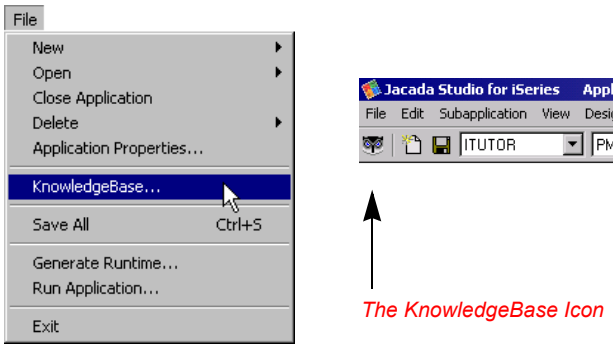
Accessing the KnowledgeBase

You can access the KnowledgeBase Interface:

From the **File** menu > **KnowledgeBase...**

or

By clicking the **KnowledgeBase Icon**  in the **Standard Toolbar**



Access the KnowledgeBase Interface from the File menu and the Standard Toolbar

The KnowledgeBase Interface

The KnowledgeBase interface consists of four panes:

Upper Left Pane	Field Definitions Pane
Upper Right Pane	Field Definition Properties Pane
Lower Left Pane	Representation Definitions Pane
Lower Right Pane	Representation Definition Properties Pane

Field Definitions Pane

Shows the Field Definitions that reside in the KnowledgeBase.

Field Definition Properties Pane

Where Field Definition properties are defined and modified.

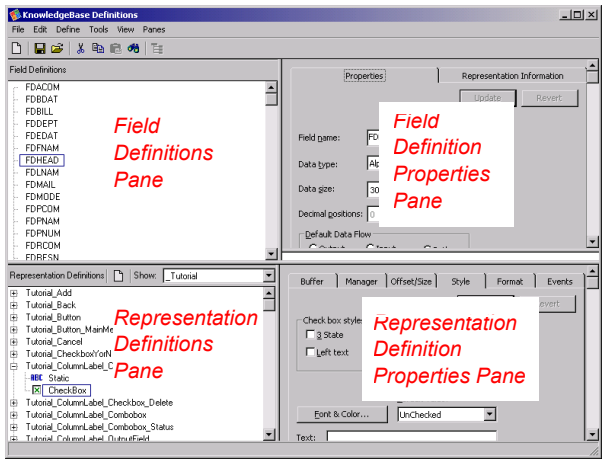
Representation Definitions Pane

Shows the Representation Definitions that reside in the KnowledgeBase. You can also see the components that make up the Representation Definition in this pane.

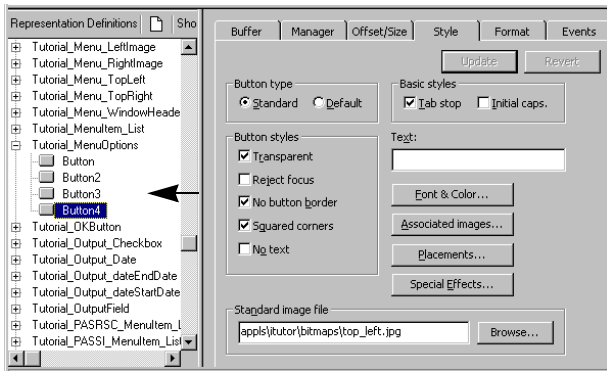
Representation Definition Properties Pane

Where Representation Definition properties are defined and modified.

* *Note :* The *Field Definition* and *Representation Definition* explanations will follow.



The KnowledgeBase Interface



An Example of Tutorial_MenuOptions - a Representation Definition defined in the KnowledgeBase. In this example you see the four buttons attached to this Representation Definition and the properties associated with one of the buttons.

KnowledgeBase Definitions

Jacada Studio for iSeries comes with a default KnowledgeBase that consists of various types of definitions. The same definitions exist in the KnowledgeBase that ships with the demo version of the product, plus other definitions that were constructed for the purpose of this tutorial. KnowledgeBase definitions created solely for the purpose of this tutorial have the prefix "Tutorial_".

For example, Tutorial_Label_Combobox is one of the definitions created just for this tutorial.

* *Note :* The properties of default knowledgebase definitions can be modified to fit your company's specific standards

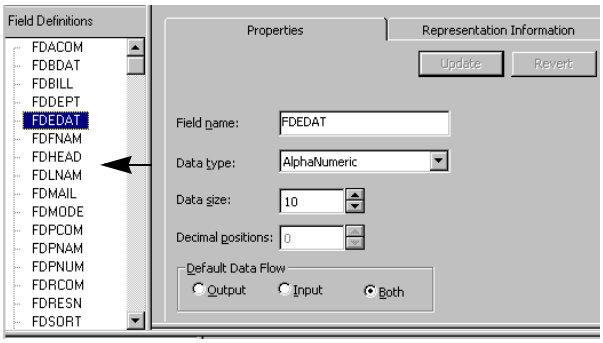
Representation Definitions

A definition in the KnowledgeBase can be constructed of one or multiple GUI controls. Together, these definitions make up the "Representation Definition" - or the "rule" stored in the KnowledgeBase, which describes controls and the properties assigned to them.

For example: The four buttons in the menu screen were created in Design View and saved to the KnowledgeBase as a group. This group, and the properties assigned to it (attached images, functionality and style) were all saved to the KnowledgeBase as a 'Representation" named **Tutorial_MenuOptions**. This method of definition allows you to create complex GUI controls, made up of multiple definitions, and save them for reuse across several Subapplications.

Field Definitions

The buffer fields that carry data to and from the host Application, and the properties assigned to them, make up the "Field Definition" - or the "rule" stored in the KnowledgeBase, which describes the fields and their properties. A Field Definition can be attached to one or more Representation Definitions in the KnowledgeBase for use within the Application. You can import your field standards from your databases into the KnowledgeBase. This feature is not covered in this tutorial, yet it is an important feature which substantially expedites KnowledgeBase creation.



An Example of FDEDAT - a Field Definition defined in the KnowledgeBase. In this example you see the properties associated with this field in the KnowledgeBase.

Methods

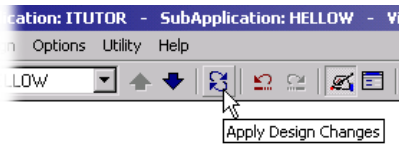
Methods are short scripts that allow you to enhance an application's functionality. Methods can be used to control and manage behavior at both the client and the host, and often act as conduit for communicating certain client activities to the host or vice versa. Methods can also be used to implement basic presentation logic such as client-side validations or the conditional display of controls or text. Jacada Studio for iSeries comes with many predefined methods that take care of the more general aspects of data flow and interaction between the host application and its graphical clients. Jacada Studio also provides you with the tools necessary to write new methods and modify existing methods. Methods can be highly specific, or quite generalized allowing a high degree of reuse. The degree of interaction that you can implement with the proper use of methods allows you to achieve a level of application sophistication that far surpasses that of applications limited by green-screen presentation.

Window Layouts

Window Layouts can be thought of as display templates that enable developers to design and implement consistent looks and feels within an application. An application will likely have several Window Layouts that define different window styles. Single record windows may have their own look and feel, while windows with tables have another. Although a single application may implement different Layouts for different window styles, Layouts can share common properties such as background graphics, company logos, and an Exit button. Because Window Layouts are defined and stored in the Knowledgebase as global components, sweeping change to the look and feel of an entire application can be made by altering a Layout and applying that change to all subapplications built using that Layout.

9. APPLY DESIGN CHANGES

If you make a change in Design View, it is recommended to apply your changes by pressing the Apply Design Changes button in the Standard Toolbar.



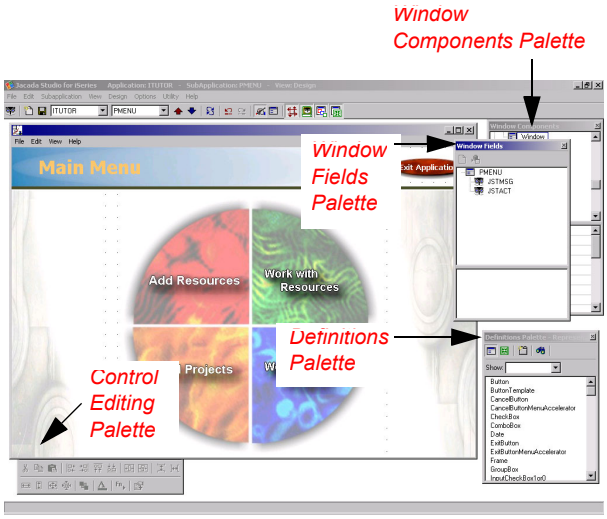
Use the Apply Design Changes button to Apply your changes

10. THE DESIGN VIEW PALETTES

When working in Design View, there are four palettes at your disposal. The image to the right illustrates the default placement of the four Design View palettes, when a Subapplication is opened. The palettes are floating and can be dragged by their title bars to any location in the screen.

Usage of Design View Palettes

Control Editing Palette	Manipulate / edit controls
Definitions Palette	Shows KnowledgeBase fields and representations
Window Fields Palette	Lists fields in the open subapplication
Window Components Palette	Lists representations in the open subapplication

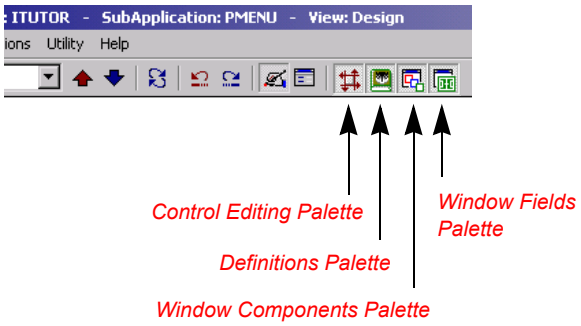


Default placement of the Design View palettes

Viewing the Design View Palettes

You can toggle the Design View palettes on and off via the Standard Toolbar by:

Toggle the Control Editing Palette	by clicking the	icon
Toggle the Definitions Palette	by clicking the	icon
Toggle the Window Components Palette	by clicking the	icon
Toggle the Window Fields Palette	by clicking the	icon

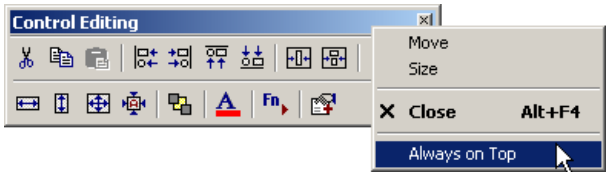


Icons Added to the IDK Standard Toolbar in Design View. These icons can be used to toggle the Design View Palettes palettes on and off.

Bring Palettes to Front

If any of the palettes become obscured by a window, click the **F12** key on your keyboard to bring them to the front.

You can also right-click (with your mouse pointer over the title bar, click the right button on your mouse) the title bar of the control editing palette, and choose **Always on Top** if you want the palette to always appear on top of the active window.



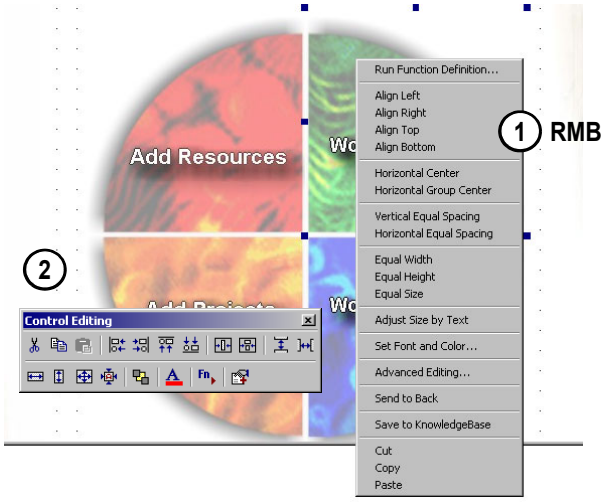
Right click the control editing Palette Title bar and choose > Always on Top if you want the palette to always appear on top of the active window.

The Control Editing Palette

When a control is selected in Design View:

1. Right click the control (with your mouse pointer over the control, click the right button on your mouse) to receive a Right Mouse Button menu with the object's editing and manipulation options.
2. Access the object's editing and manipulation options via the Control Editing Palette.

* *Note :* Control editing and manipulation options are discussed in the next step.



When an object is selected in Design View, access the control editing / manipulation options via the right mouse button menu or the Control Editing Palette.

The Definitions Palette

The Definitions Palette allows you to view the names of Fields and Representations that are defined in the KnowledgeBase. From this palette, you can add instances of the Fields and Representations that exist in the KnowledgeBase to your Subapplication, by dragging them onto your window. There are two views in the Definitions Palette: *Representation Definitions View* and *Field Definitions View.*

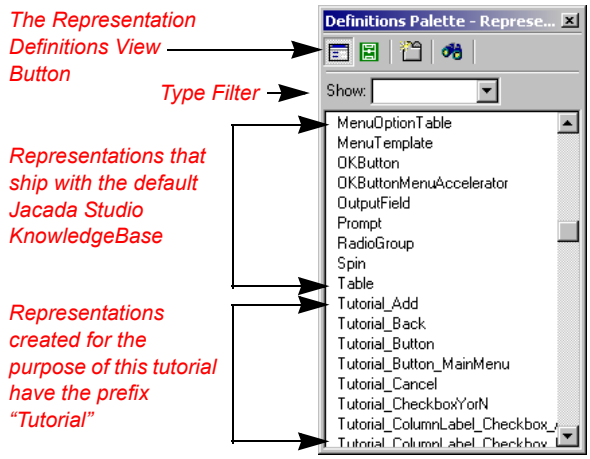
Representation Definitions View Button

Click the Representation Definition View button in the Definitions Palette to see the list of Representation Definitions defined in the KnowledgeBase. Representation Definitions created solely for the purpose of this tutorial have the naming convention Tutorial_xxx, with the prefix "Tutorial".

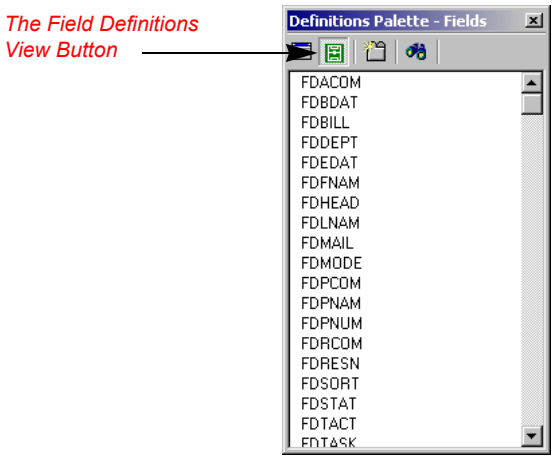
* *Note :* Use the filter at the top of the Definitions Palette to sort the definitions in the palette by type. The filter options can be customized within the IDK.

Field Definitions View Button

Click the Field Definition View button in the Definitions Palette to see the list of Field Definitions defined in the KnowledgeBase.



See the list of Representation Definitions defined in the KnowledgeBase by clicking the Representation Definitions View button in the Definitions Palette

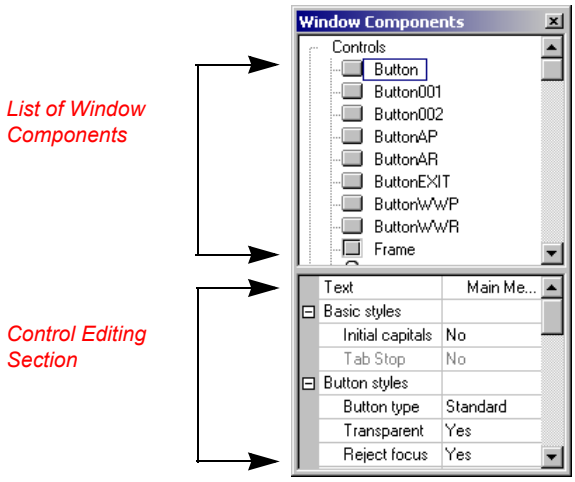


See the list of Field Definitions defined in the KnowledgeBase by clicking the Field Definitions View button in the Definitions Palette

The Window Components Palette

A list of the components that reside in your Subapplication. When components are added to the window via the Definitions Palette, they automatically receive a name that appears in the list of components in the Window Components Palette. Modify Control properties via the *control editing section* of the Window Components Palette.

* *Note :* Remember: Please do not modify the ITUTOR application.



See a list of the components that reside in your Subapplication in the Window Components Palette. Modify Control properties via the control editing section.

The Window Fields Palette

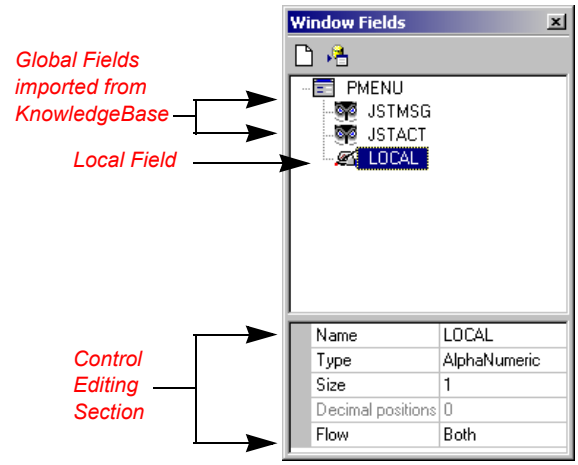
A list of the fields that reside in your Subapplication. Notice the difference in *icons* for local fields in the list and global fields in the list, defined in the KnowledgeBase. At the top of this dialog there are two buttons. You can create local fields and import global fields via these buttons.

Create New Local Field

Use this button to create a new local field in the Subapplication. You can define a field's properties via the control editing section of the Window Fields Palette. You can only use a local field in the Subapplication in which it was created.

Add Field From KnowledgeBase

Add an instance of a field defined in the KnowledgeBase. Pressing this button brings up the *Import Fields Dialog* from which you choose the fields that you wish to use. You cannot modify the properties of fields imported from the KnowledgeBase.



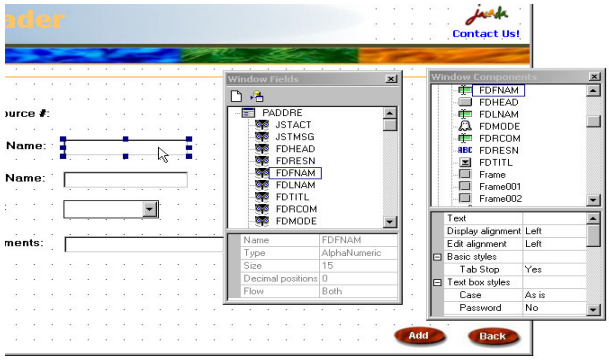
See a list of the fields that reside in your Subapplication in the Window Fields Palette. Modify the properties of locally defined fields via the control editing section.

Using Palettes to View The Relationship Between Window Components and Window Fields

When you select a component in the window, the component's name is automatically selected in the Window Components Palette. If there is a field attached to the component, the field name is also selected in the Window Fields Palette.

The converse is also true: if you select a component in the Window Components Palette, it is automatically selected in the window. The attached field (if any), is selected in the Window Fields Palette.

This selection system also works when fields are selected in the Window Fields Palette.



When you select a component in the window, the component's name is selected in the Window Components Palette. If there is a field attached to the component, the field name is also selected in the Window Fields Palette.

11. SETTING UP YOUR WORKSPACE

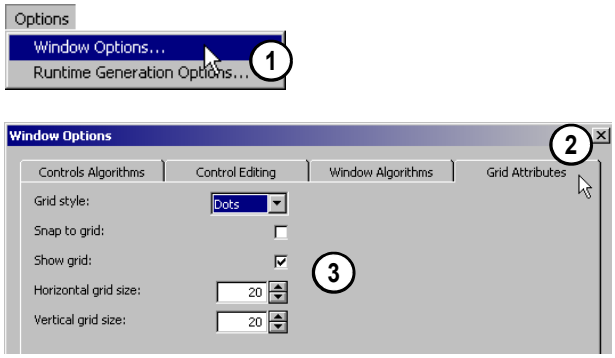
The grid tool provides a visual guide for aligning controls on the GUI in Design View. The *Snap to Grid* option is useful in regularizing the placement of controls on the GUI.

Configuring the Grid

To configure the grid:

1. From the **Options** menu select **Window Options**. *The Window Options dialog box opens.*
2. Select the **Grid Attributes** tab:
3. In this dialog you can set the following options:

Grid Style	The grid markings can be either lines or dots.The default is dots.
Snap to Grid	When enabled the upper left corner of the control snaps to the nearest crossing of horizontal vertical grid lines. When a group of controls is selected, the upper left corner of the imaginary rectangle that encompasses the group snaps to the grid line crossing.
Show Grid	When enabled the window is displayed with a grid. When disabled the grid is not displayed
Horizontal Grid Size	Set the horizontal distance, in dialog units, between the grid lines.
Vertical Grid Size	Set the vertical distance, in dialog units, between the grid lines.

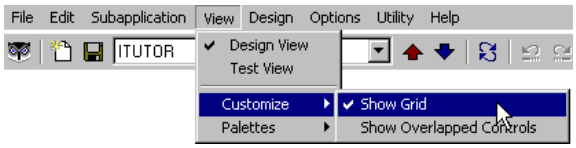


Configure the Grid by selecting the Options menu > Window Options and configuring the parameters in the Grid Attributes tab.

Toggling the Grid on and Off

You can toggle the grid on and off by:

Going to the **View Menu** > select **Customize** > **Show Grid**



Toggle the grid on and off via the View menu > Customize > Show Grid option.

12. CONTROL EDITING AND MANIPULATION OPTIONS

This step will introduce you to the following subjects:

- Modifying Component Properties
- Selecting Controls in the GUI
- Control Editing and Manipulation Options

Modifying Component Properties

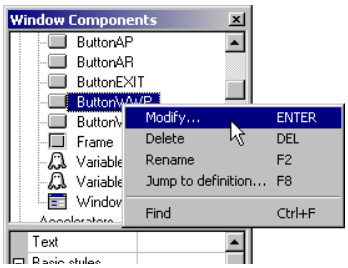
Modifying component properties is done through the **Component Properties Dialog** in Design View. The Component Properties dialog automatically displays the properties of the component that it was accessed from. For example, when accessed by clicking on an edit component, the edit component's properties will appear, when accessed by clicking on a button component, the button component's properties appear.

To access the **Component Properties Dialog**:

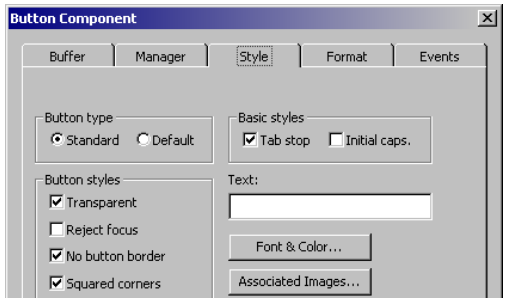
In the **Window Components Palette**, double click the component's name or right click the component's name and choose **Modify** from the floating menu.

or

Double click the control on the window (for menu items, open the menu and double click the item).



Access a component's properties dialog by double clicking the component's name in the Window Components Palette, and choosing Modify from the floating menu.



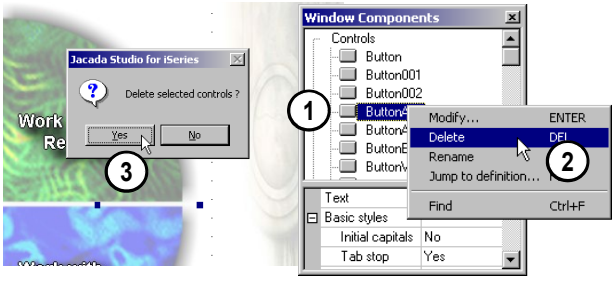
The style properties of a button component in the button component's properties dialog.

Deleting Components

To delete a component:

1. In the **Window Components Palette**, select the component you wish to delete.
2. Press the **Delete** key
   or
   Click with the right-mouse button. In the floating menu, choose **Delete**.
3. In the messagebox that opens, click **Yes**.

* *Note :* A control can be also deleted by selecting it in the window and pressing the **Delete** key.

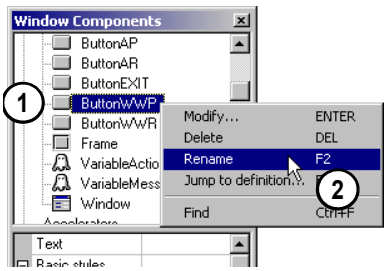


To delete a control, select it either in the window, or in the Window Components Palette and press the delete key on your keyboard. In the messagebox that opens, click Yes.

Renaming Components

To rename a component:

1. In the **Window Components** box, select the component you wish to rename.
2. Press the **F2** key
   or
   Click with the right-mouse button. In the floating menu that opens, choose **Rename**.
3. The component name becomes editable. Type in the new name.



To rename a component, select it in the Window Components Palette and choosing Rename from the floating menu.

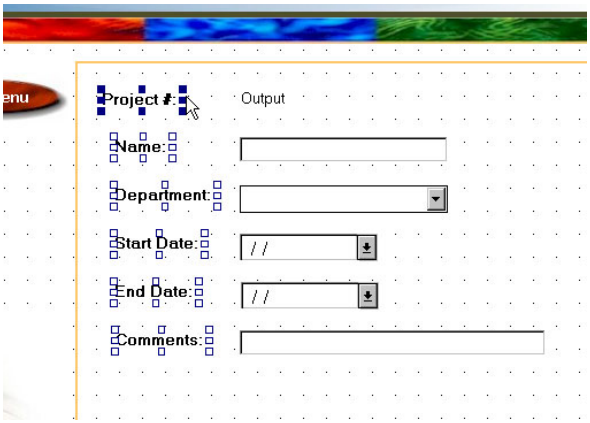
Selecting Controls in the GUI

In Design View, controls can be selected in several ways:

Individually	A single control can be selected. Additional controls can be added individually to the selection.
In Groups	All the controls can be selected or A group of controls of a certain type can be selected.

Leading Control

All selected controls are surrounded by sizing handles. The Leading control is the control whose sizing handles are emphasized when a group of controls are selected. The control editing options use a leading control to establish the standard by which the other controls are manipulated. For example, when a group of controls are resized they assume the size of the leading control. In the illustration to the right, the "Project #" label is the leading control.



Click a selected control to designate it as the leading control.

Selecting Controls Individually

Select a control by clicking on it. When a control is selected it receives sizing handles.

- To select more than one control press **SHIFT + Click** each control to be selected.
- To add controls to a selection press **SHIFT + Click** on the additional controls.
- To remove one control from a selection, use **SHIFT + Click** on the control to remove.
- Clicking a selected control designates the clicked control as the leader control.
- Clicking the window's client area clears all selected controls.

Selecting Controls by Group

To select a group of controls on a Window:

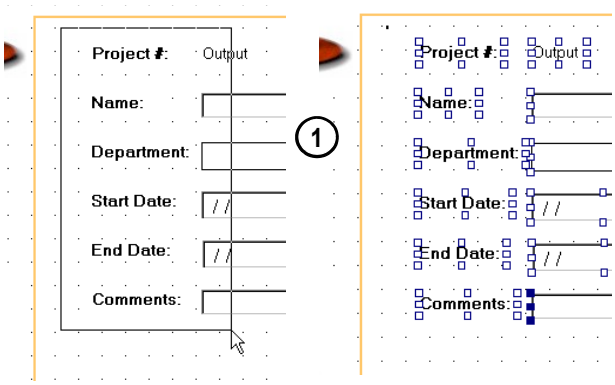
1. Click on the window client area while holding down the left mouse button and drag the cursor across the window to mark a rectangular around the controls to be selected. *The controls that are inside of the rectangle or touching the rectangle are selected.*
2. Use the **Select** options in the **Design** pull-down menu to select all controls of a specific type.

The Select Options in the Design Menu

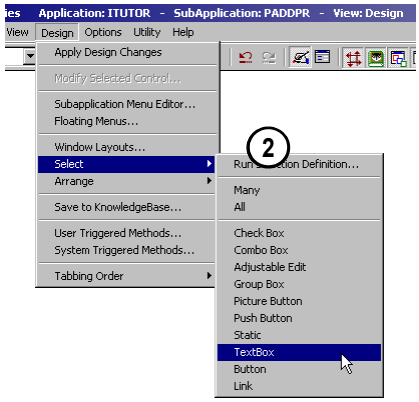
Selecting one of the **Select** options in the *Design Menu* changes the cursor to a cross-hair. Drag this cursor across the Window to mark a rectangular area on the screen. The controls that are inside of the rectangle or touching the rectangle are selected.

The following options are available for selecting controls:

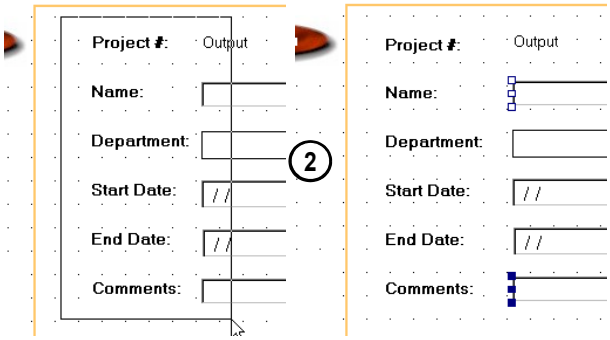
Run Selection Definition	An advanced topic that will not be introduced in this tutorial.
Many	Selects all the controls within the rectangle. When selected, the mouse cursor becomes a crosshair. While holding down the left mouse button and dragging the mouse across the window, the "Many" option selects all of the controls in the rectangle.
All	Selects all the controls.
Check Box	Selects all the Check Boxes within the rectangle.
Combo Box	Selects all the Combo Boxes within the rectangle.
Group Box	Selects all the Group Boxes within the rectangle.
Static	Selects all the Static controls within the rectangle.
Textbox	Selects all the Adjustable Edits within the rectangle.
Button	Selects all the Buttons within the rectangle.
Link	Selects all the Links within the rectangle.



Click on the Window client area and drag the cursor around the controls to be selected.



Use the Textbox Select options in the Design pull-down menu to select all controls of a textbox type.



Only textbox controls are selected.

Clearing Selected Controls

To clear selected controls in the window client area:

- Click the window's client area to clear all selected controls.
- Press the **Shift** key while dragging the cursor to add controls to those already selected in the Window.
- Press the **Shift** key and click an already selected control.

Control Editing and Manipulation Options

The Control Editing and Manipulation options are the heavy-duty functions that you can use to design the placement of the controls on your window. To access these options select **Arrange** from the **Design** menu.

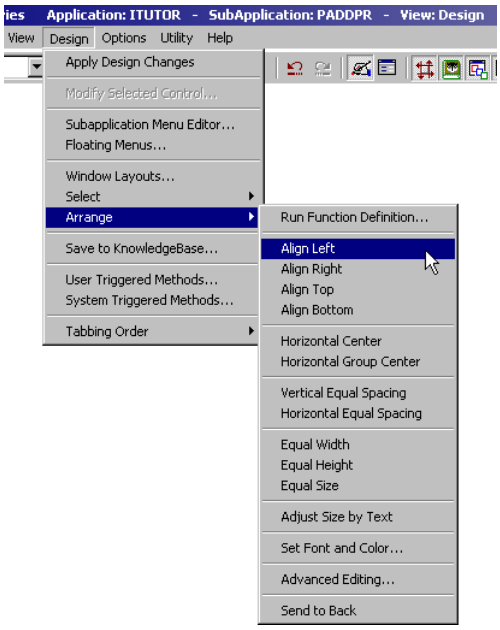
Calling up the Arrange Menu with the Right Mouse Button

The Arrange options can be called up by pressing your right mouse button. This is a useful shortcut for calling up the Arrange cascading menu when a control is selected. To call up the **Arrange** menu:

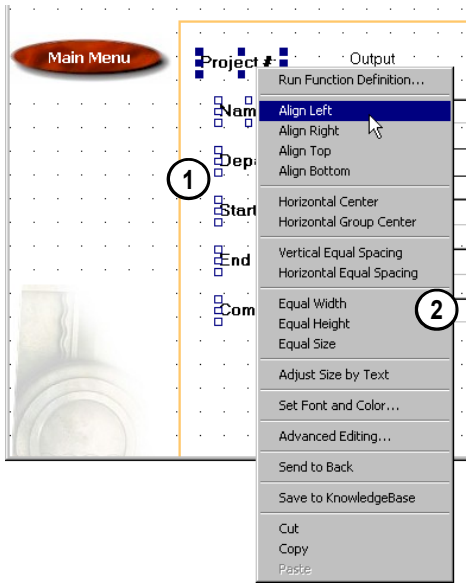
1. Select a control or group of controls.
2. Click the right mouse button. The RMB menu will be displayed where you have clicked on the window.

The following are the Control Editing and Manipulation options:

Run Function Definition	An advanced topic that will not be introduced in this tutorial.
Align Left	Aligns the selected controls to the left according to the left edge of the leading control.
Align Right	Aligns the selected controls to the right according to the right edge of the leading control.
Align Top	Aligns the selected controls according to the top edge of the leading control.
Align Bottom	Aligns the selected controls to the bottom edge of the leading control.
Horizontal Center	Places each of the selected controls in the horizontal center of the suggested Window's client area. To center several controls that are located on the same horizontal line, use the Horizontal Group Center option.



Access the control editing and manipulation options via the Arrange option in the Design menu.



Access the control editing and manipulation options right mouse button menu.

Horizontal Group Center	Places the selected controls (as a group) in the horizontal center of the suggested Window's client area.
Vertical Equal Spacing	Spaces the selected controls at equal vertical distances.
Horizontal Equal Spacing	Spaces the selected controls at equal horizontal distances.
Equal Width	Resizes the selected controls to an equal width, according to the width of the leading control.
Equal Height	Resizes the selected controls to an equal height, according to the height of the leading control.
Equal Size	Resizes the selected controls to an equal size, according to the size of the leading control.
Adjust Size by Text	Resizes the selected controls according to the length of the text.
Set Font and Color	Change the font and/or color of the selected control(s)
Advanced Editing	Additional editing features not covered in this tutorial.
Send to Back	When the selected control is covering other controls in the IDK, makes the next control visible. Does not affect the runtime.
Cut	Removes the selected control from the GUI and places it on the internal clipboard.
Copy	Copies the selected control and places it on the internal clipboard.
Paste	Adds the internal clipboard's contents to the GUI.

Control Manipulation via Keyboard Arrows

Use the keyboard arrow keys to move selected controls by single units, or according to the grid lines when the grid is displayed.

Exercise 1 - Jacada Studio for iSeries

Your First Application

Objectives:

- To introduce the reader to the development process workflow
- To familiarize the reader with the mechanics of constructing and executing of a very simple Studio for iSeries graphical program
- To expose the reader to the wizard-driven design and deployment features and automated code generation capabilities of Jacada Studio

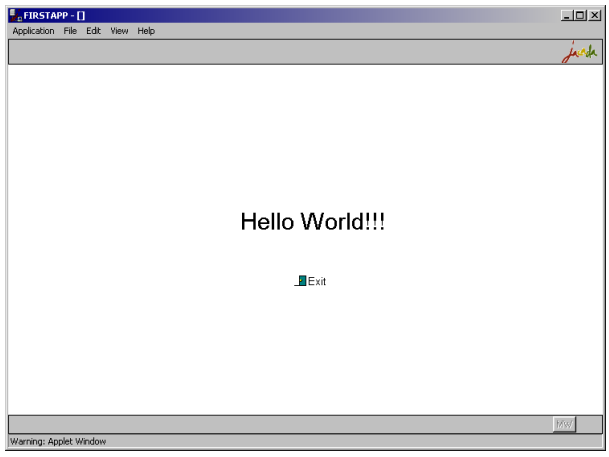
In this exercise you learn about the underlying structure that is common to all Jacada Studio for iSeries Applications. The topics in this chapter lay the groundwork of understanding which you need before you can modify the look and feel of your new GUI Application. In this chapter, you learn how to build a brand new Application, using the default Jacada Studio Knowledgebase which ships with the full version of the product. The purpose of this exercise is to let you to experience the entire Jacada Studio workflow with the full default Knowledgebase. In following chapters, you will use the predefined KnowledgeBase created especially for the ITUTOR Application provided for the evaluation version of the product. For now, let's see exactly what you can do with a bit of imagination, and the default Jacada Studio for iSeries environment.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

The major steps to this exercise are:

1. **Window Design Specifications**
2. **Open the Jacada Studio for iSeries IDK**
3. **Create an Application**
4. **Create a Subapplication**
5. **Add GUI Components to the Window**
6. **Generate A Runtime**
7. **Compile Transferred Files**
8. **Ensure the Jacada Monitor is Active**
9. **Run Application with a Java Client**
10. **Run Application with an XHTML Client**



The final product of your efforts in this exercise.

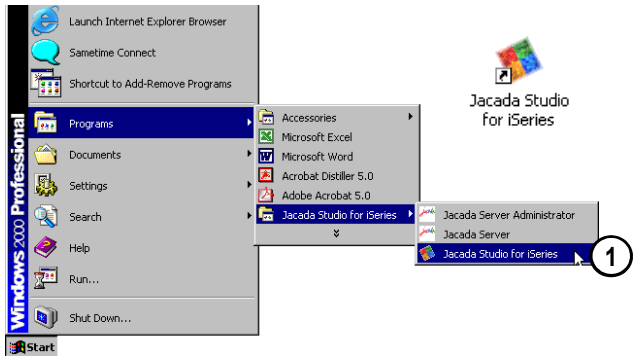
1. WINDOW DESIGN SPECIFICATIONS

The design specification for this window is to create a GUI window with the text "**Hello World**", add a button to **Exit** the application and run a built Application in both the Java and XHTML runtime clients.

2. OPEN THE JACADA STUDIO FOR ISERIES IDK

If you are not already in the Jacada Studio development environment, open the IDK interface. To open the Jacada Studio for iSeries IDK:

1. From your Windows **Start** menu > choose **Programs** > **Jacada Studio for iSeries** > **Jacada Studio for iSeries**.
   The Jacada Studio Interface Development Kit (IDK) is invoked.



From your Windows Start menu, navigate to the Jacada Studio for iSeries icon to bring up the IDK interface.

3. CREATE AN APPLICATION

Applications in the IDK are created via the New Application Wizard. An IDK Application consists of all Client and Server elements, as well as information which is used to write the Host code.

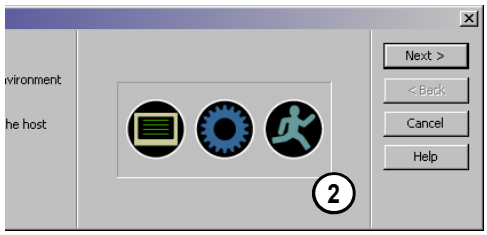
1. From the **File** menu > choose **New** > **New Application...**

Create a New Application.

2. In the **New Application Wizard**, specify the following Application properties:

* *Note :* Notice that each of the following parameters are on different steps of the wizard.

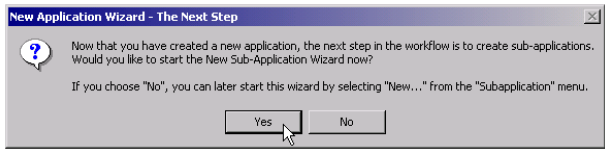
Application Name:	**FIRSTAPP**
Language:	English USA
Host Programming Environment:	RPG_OPM
Resolution:	SVGA (800x600)



Set Application properties via the New Application Wizard.

3. Click **Finish** to come out of the New Application Wizard.

* *Note :* Upon exiting the New Application Wizard, you are prompted to start the New Subapplication Wizard, the next logical step in the workflow. If you choose 'Yes', skip step number 1 in the next section.



Upon exiting the New Application Wizard, you are prompted to start the New Subapplication Wizard, if you choose 'Yes', skip step number 1 in the next section

4. CREATE A SUBAPPLICATION

The term "Subapplication" is used to describe the contents of an IDK GUI Window. Each window is referred to as a Subapplication. Subapplications consist of GUI Elements, Host Fields and the links between them. In this step, you will create your first Jacada Studio Subapplication with the *BasicLayout* Window Layout.

To create a new SubApplication:

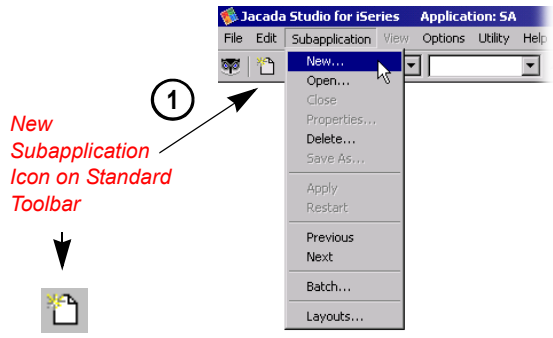
1. From the **Subapplication Menu** > **New...**
   or
   Click the **Subapplication Icon** on the *Standard Toolbar*
2. In the **New Subapplication Wizard**, Specify the following Subapplication properties:

* *Note :* Notice that each of the following parameters are on different steps of the wizard.

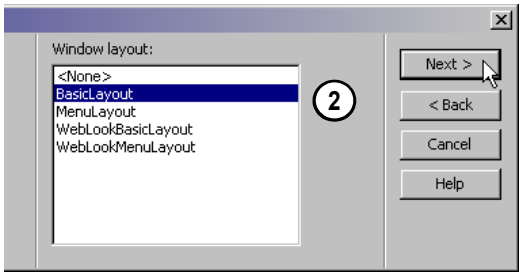
Subapplication Name:	**HELLOW**
Popup Window:	Unchecked
Window Layout:	**BasicLayout**
Subapplication Description:	None

* *Note :* The Window Layout feature will be explained in detail in the next exercise.

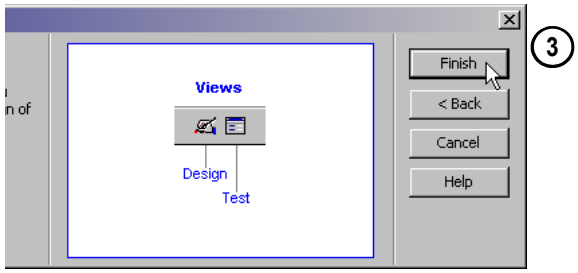
3. Click **Finish** to come out of the New Subapplication Wizard.



Three ways to access the New Subapplication Wizard.



Specify subapplication properties in the New Subapplication Wizard. Choose <none> from the Window layout list in the Window Layout step of the New Subapplication Wizard.



Click Finish to come out of the New Subapplication Wizard.

5. ADD GUI COMPONENTS TO THE WINDOW

In the IDK, GUI Components are referred to as 'Representations', because they are the visual presentation of information that would have been found in the host screen display. In this segment, we use the **Definitions Palette** to add Representations and Fields stored in the KnowledgeBase to our Subapplication. We will then use the **Window Fields Palette** to View and Edit the Field properties and the **Window Components Palettes** to View and Edit the Representation properties.

Add a GUI Component and Edit Control Properties

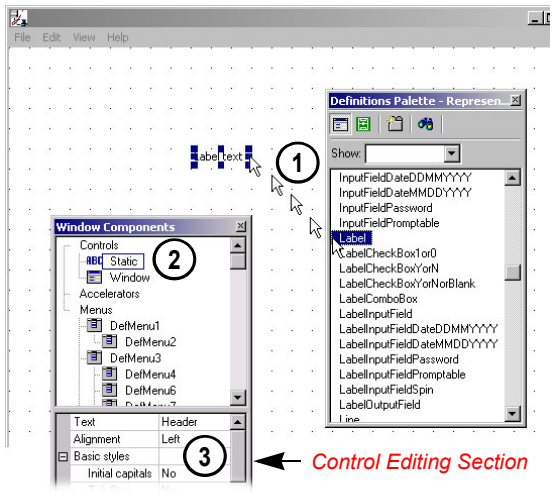
1. From the **Definitions Palette**, select the **Label** representation, and drag it onto your Subapplication.
2. Notice that when a Representation in the window is selected, it is subsequently selected in the *Window Components Palette*.
3. In the **Control Editing Section** of the *Window Components Palette*, change the following control properties, in the following order:

* *Note :* After you change any control properties, you must press **ENTER** for the change to take effect.

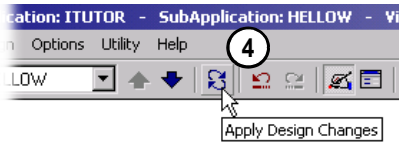
Text:	Hello World !!!
Static Styles > **Use Separator:**	No
Font and Color > **Font >** **Size:**	 22
Location > **X:** **Y:**	 280 180
Size > **Width:** **Height:**	 200 30

4. From the Standard Toolbar, click the **Apply Design Changes** button.

* *Note :* Use the Apply Design Changes button to apply your changes to the window.



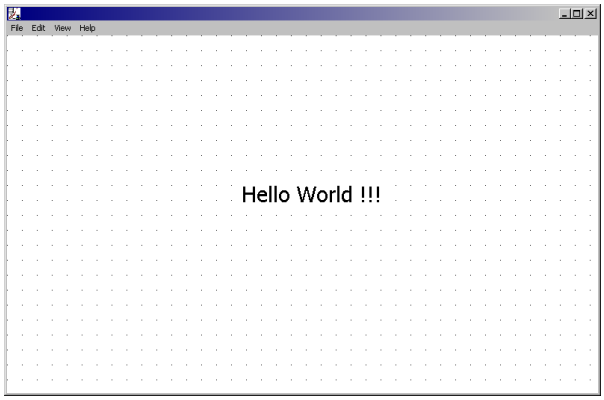
Click and Drag Representations onto your window. Representations selected in the Window, are also selected in the Window Components Palette.



Use the Apply Design Changes button to Apply certain changes to your design that do not update automatically.

What your Window Should Look Like

This is what your window should look like at the end of the last step.

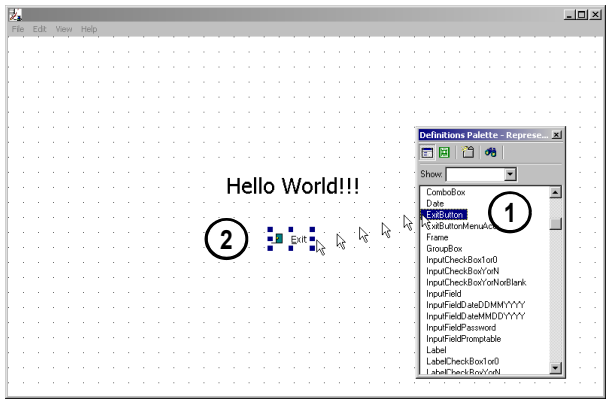


What your window should look like at the end of the last step

Add an 'Exit' Button

In order to gracefully terminate your runtime session, add an 'Exit' button to your Subapplication. To add the 'Exit' button:

1. From the **Definitions Palette**, select the **ExitButton** representation, and drag it onto your Subapplication.
2. Manually place the 'Exit' button under the 'Hello World' representation.



Select the ExitButton representation, and drag it onto your Subapplication.Manually place the 'Exit' button under the 'Hello World' representation.

Save Subapplication

Before Generating a Runtime, you should save your Subapplication. To save your Subapplication:

From the Standard toolbar choose the  icon.

Use the Save button on your standard toolbar to Save your Subapplication.

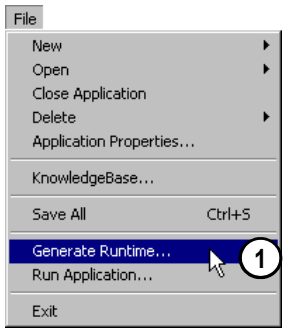
6. GENERATE A RUNTIME

The individual Subapplications are analogous to source code. You compile the Subapplications into an executable in a process called Generating the Runtime. In the IDK, Generate Runtime is a wizard driven process. To generate a Runtime:

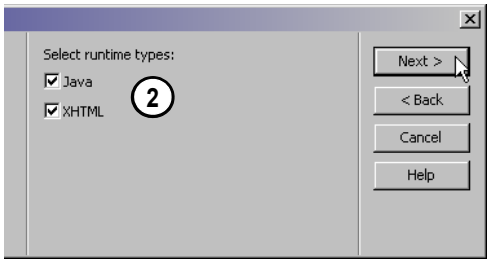
1. From the **File** menu > choose **Generate Runtime...**
   The Generate Runtime Wizard is invoked.
2. In the **Generate Runtime Wizard**, click **Next** to accept the following default settings:

* *Note :* Notice that each of the following parameters are on different steps of the wizard.

Runtime Type:	**Java** and **XHTML**
Jacada Server Platform:	Windows NT(2000) x86



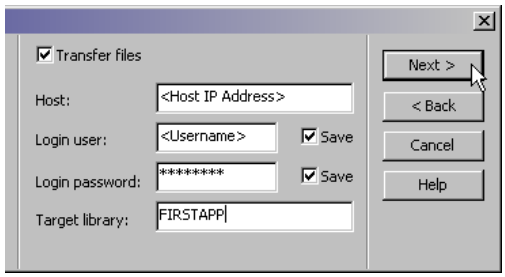
Generate a Runtime to compile your Subapplication.



Choose both Java and XHTML runtime types.

3. In the **File Transfer** screen, specify the following information, then click **Next**:

Transfer files:	**<Checked>**



Fill out the File Transfer screen of the Generate Runtime Wizard

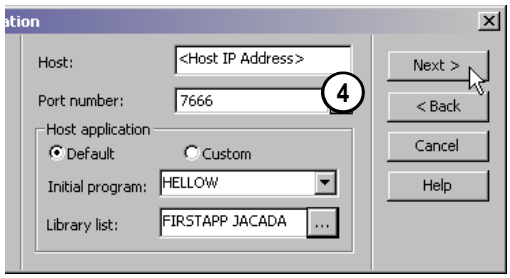
* *Note :* When checked, the "*Transfer files*" feature automatically transfers the files created by the Generate Runtime process to the appropriate Libraries / Objects on the host. If you don't check this checkbox, no connection to the host will be made and no files will be transferred.

Host:	<YourHostIPAddress>
Login User:	<YourLoginUserNameOnHost>
Login Password:	<YourLoginPasswordOnHost>
Target Library:	**FIRSTAPP**

* *Note :* Specify a target library for your Application files. If a library by the name you specify does not exist, one will be created for you.

4. In the **Specify Host Connection and Application Information** screen, specify the following information, then Click **Next**:

Host:	<YourHostIPAddress>
Port Number:	**7666**



Fill out the Host Connection and Application Information screen of the Generate Runtime Wizard

* *Note :* By default, the Jacada Monitor is set to listen on port 7666. If this port is in use by another Application, is recommended that you (a) bring down the service that is currently occupying port 7666 for the duration of this tutorial, or (b) type **`CFGJACMON`** into your iSeries command line and press Enter. This invokes the Configure Jacada Monitor utility. Use the Change function to change the port that the Jacada Monitor listens on.

Default and Custom Host Application Buttons

The settings in the Host Application section change according to whether you choose the **Default** or **Custom** radio button.

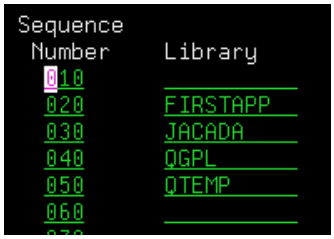
In this example, the **Default** settings are used. The **Default** settings allow you to use the pre-configured CLWrapper that was created for the purpose of this tutorial. The CLWrapper can be customized to provide additional advanced capabilities (i.e. program calls) and the **Custom** settings can then be used to invoke your customized settings. For the duration of this tutorial, you will use the basic CLWrapper by selecting the **Default** option.

Initial Program:	HELLOW

* *Note :* This is the first program that is called during runtime.

Library List:	FIRSTAPP JACADA

* *Note :* Specify the library list that will be set for the host session. Include your Application Library and the JACADA library.



In the Library List setting, you specify the library list that will be set for the host session. Always include your Application library and the JACADA library.

5. Click **Finish** to come out of the Generate Runtime Wizard, and commence with the compilation process.
   The Generating the Runtime dialog appears.

* *Note :* In this dialog, you can view the output regarding the status of the Runtime Generation process. There is crucial information output to this dialog, go over the output to see which files were transferred to the host. Look for the message "Runtime was successfully generated" at the end of the compilation process, this is an indication that all is well and you can safely go on to the next step.

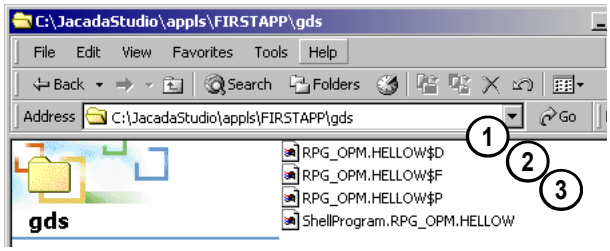
6. Click **Close** to come out of the Generating the Runtime dialog.

The Generating the Runtime Dialog shows the status of a Runtime Generation

Files Created By the Generate Runtime Process on the Development PC

Look in the **JacadaStudio\appls\<ApplName>\gds** directory - 4 files were created on the development PC:

1. **RPG_OPM.HELLOW$D** - *The DDS Physical File* - Describes the data structure of the records in the Subapplication. This file is used as a source for Jacada's Service Program. The structure of the Jacada DDS File is based on the structure of Physical File DDSs (PF) and does not contain the information found in Display File DDSs (DSPFs).
2. **RPG_OPM.HELLOW$P** - *The Parameter List Copybook* - Contains a parameter list containing control fields that are passed by the Jacada Studio API from the end program to Jacada's Service Program.



The files created on the development PC as a result of the Runtime Generation Process

3. **RPG_OPM.HELLOW$F** - *The File Specification Copybook* - This file declares each window / table as a special file. It links the window / table DDS and the parameter list from the Parameter List Copybook to the Jacada Service Program.

* *Note :* The dollar sign ($) is used in the name of the files generated for window definitions. It will be replaced by a pound sign (#) for files generated for table definitions.

4. **ShellProgram.RPG_OPM.HELLOW** - *The Shell Program*. An automatically generated program that performs the mandatory actions involved in displaying a Jacada Studio Subapplication. This program should be added to by the programmer to implement the business logic. For testing purposes, the window can be run without any change whatsoever to the automatically generated shell program.

* *Note :* Automatically generated shell program contains all you need to test run an Application. The Read/Write statement in the automatically generated shell program is very similar to the way you Read/Write to a display file.

These are text files that can be opened with any text editor. Feel free to open these files and have a look at their structure. IMPORTANT! - do not change the content of these files.

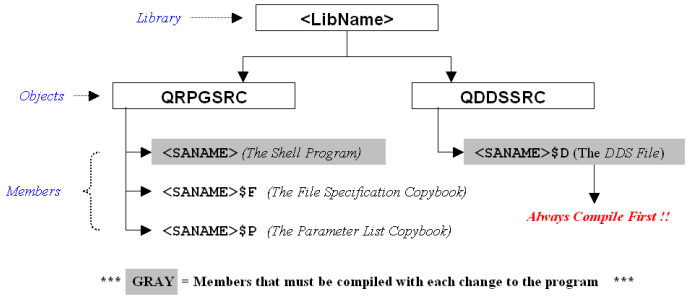
```
  * This code was generated automatically by Jacada.
  *    Sub-Application: HELLOW
  *    Time generated:  Tue Jun 25 17:13:55 2002
  * ------------------------------------------------------------
  *
  * Import the file specifications for the current window.
F/COPY HELLOW$F
  * Place other file specifications imports here.
  *
  * Import GDS E specifications
E/COPY JRPGSRC,GDSESPECS
  * Place your E specifications here.
  *
  * Import GDS I specifications
I/COPY JRPGSRC,GDSISPECS
  * Place your I specifications here.
  *
  * Import the parameters list defined for the window.
C/COPY HELLOW$P
  *
  * ------------------------------------------------------------
  * Start of main loop.
  * ------------------------------------------------------------
  *
  * Place your code for refreshing the window's data here.
  *
  * Display the window
C                     WRITEHELLOWB
C                     READ HELLOWB                  99
  *
  * Place your program logic here.
  *
C                     SETON                       LR
  *
  * ------------------------------------------------------------
  * End of main loop.
  * ------------------------------------------------------------
  *
C           *INZSR    BEGSR
  *
  * Place your initialization code here.
  *
C                     ENDSR
```

4

Automatically generated shell program contains all you need to test run an Application.

Libraries Objects and Members Created by the Generate Runtime Process

The first time that you choose to transfer the files created by the Runtime Generation process to the host by checking the *Transfer files* checkbox in the *Transfer Files* screen of the Generate Runtime Wizard, the library structure in the diagram to the right is created on the host in the Target Library that you specified. After the first time that you choose to transfer files via the Generate Runtime Wizard, each time that you Generate a Runtime, only the DDS physical file is transferred.



The libraries, objects and members created the first time that you transfer files to the host.

7. COMPILE TRANSFERRED FILES

Of the four files transferred to the host, only the DDS physical file and the Program file ever need to be compiled. The File Specification copybook and the Parameter List copybook are not compiled. Furthermore, unless there is a change to the fields in the Subapplication, there is no need to recompile the DDS physical file and the program file on the host.

Compile the DDS Physical File, **HELLOW$D** file and the RPG Shell Program., **HELLOW**.

* *Note :* Make sure the target library is in your library list.

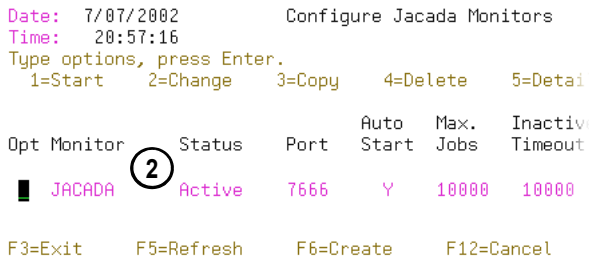
8. ENSURE THE JACADA MONITOR IS ACTIVE

To ensure that the Jacada Monitor is active:

1. Type **CFGJACMON** in the iSeries command line.
2. In the Configure Jacada Monitors screen, make sure that the word '**Active**' Appears in the *Status* column to the right of the Jacada Monitor.

If the Jacada monitor is not active:

Type **1** in the *Opt* column to start the Jacada Monitor.



In the Configure Jacada Monitors screen, make sure that the word 'Active' Appears in the Status column to the right of the Jacada Monitor

9. RUN APPLICATION WITH A JAVA CLIENT

In order to test the runtime of your new GUI Application, you can run the executable created during the Runtime Generation process from within the development environment. You actually created two clients during the Runtime Generation process: The Java client and the XHTML client. We will run the Java client now through the IDK wizard driven feature.

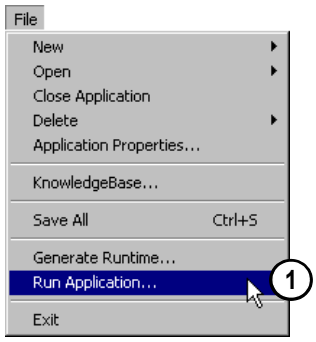
Running the Application is a wizard driven process and can be achieved by:

1. From the **File** menu > choose **Run Application...**
2. In the **Run Application Wizard**, click **Next** to accept the following default settings:

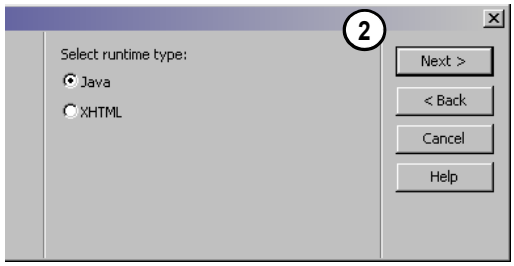
Runtime Type:	**Java**
Web Server:	Integrated HTTP Service
Application URL:	http://localhost:8080/FIRSTAPP.html

3. Click **Finish** to come out of the Run Application Wizard. *Wait a couple of seconds. The Jacada Server is activated in a DOS command window and Your Default Browser window is opened to the Jacada <ApplName>.html page.*

* *Note :* If the server is still loading, you will receive a message box. Answer "Yes" to wait for the server to load.

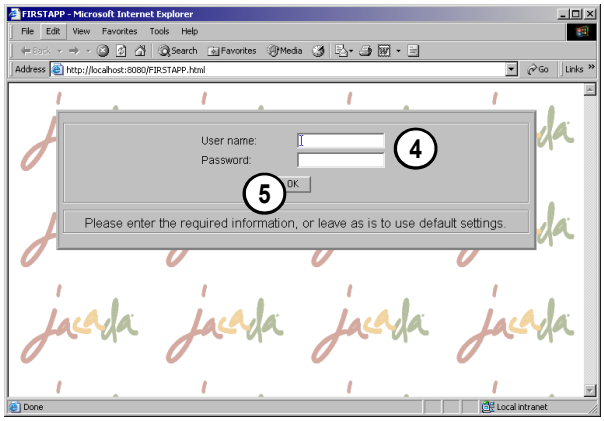


Run your Application through the development environment by choosing File > Run Application



Choose Java as your runtime type in the Run Application Wizard

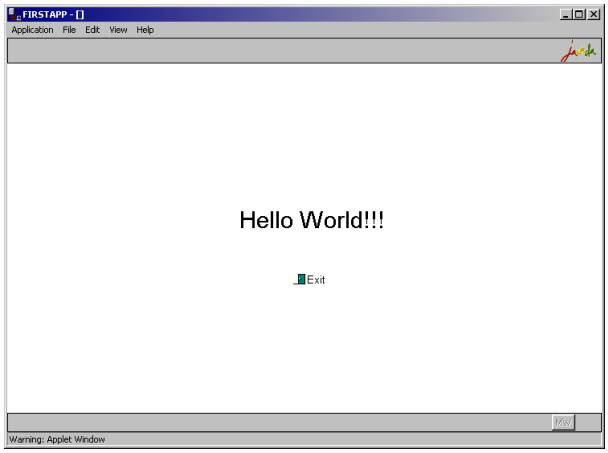
4. Type your iSeries **Username** and **Password** into the appropriate fields.
5. Click the **OK** button to run your Java client Application.



Supply your iSeries username and password to invoke the Application via the Jacada <ApplName>.html page

The Finished Product

This is what your window should look like in a Java runtime.

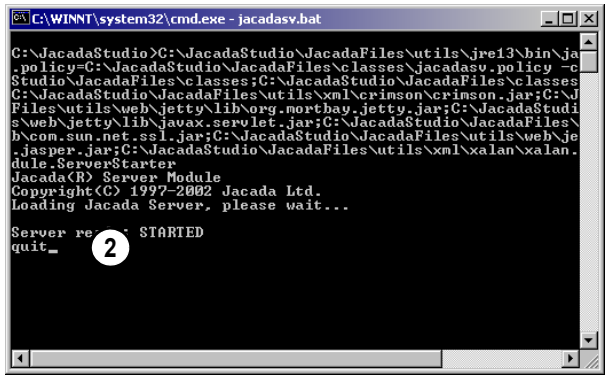


What your window should look like in a Java runtime

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by clicking the 'Exit' button in your runtime window.
2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes if a message appears.
3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.
4. Close your browser window.



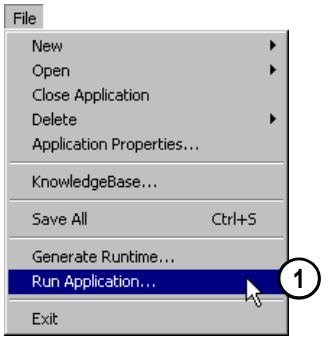
Type 'quit' in the Jacada Server command window or press Ctrl+C to close the Jacada Server

10. RUN APPLICATION WITH AN XHTML CLIENT

Now, let's run the XHTML client. XHTML is a reformulation of HTML in XML. XHTML stands for Extensible Hypertext Markup Language. For all practical purposes, XHTML is just like HTML. When we run an XHTML runtime, we receive an HTML client.

To run an HTML client:

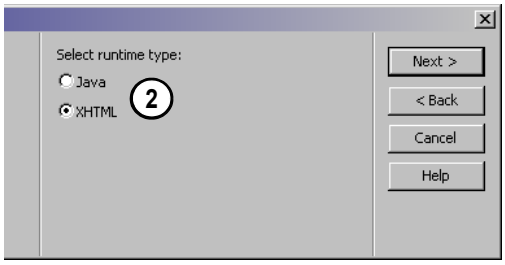
1. From the **File** menu > choose **Run Application...**



Run your Application through the development environment by choosing File > Run Application

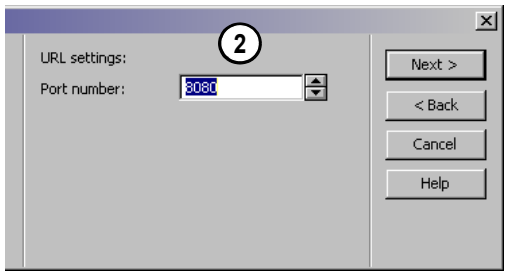
2. In the **Run Application Wizard**, specify the following settings:

Runtime Type:	**XHTML**
Port Number:	8080
Application URL:	http://localhost:8080/FIRSTAPP-xhtml.html



Choose XHTML as your runtime type in the Run Application Wizard

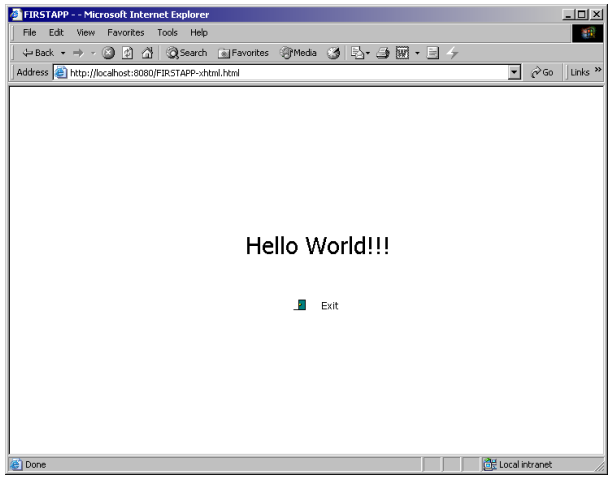
3. Click **Finish** to come out of the Run Application Wizard. *Wait a couple of seconds. The Jacada Server is activated in a DOS command window and your Default Browser window is opened to the Jacada <ApplName>.html sign-on window.*
4. Type your iSeries **Username** and **Password** into the appropriate fields in the Jacada <Applname>.html sign-on window.
5. In the sign-on window, click the **OK** button to run your HTML client Application.



Click Next to accept 8080 as the default port number for your XHTML Runtime

The Finished Product

This is what your window should look like at the end of the last step.

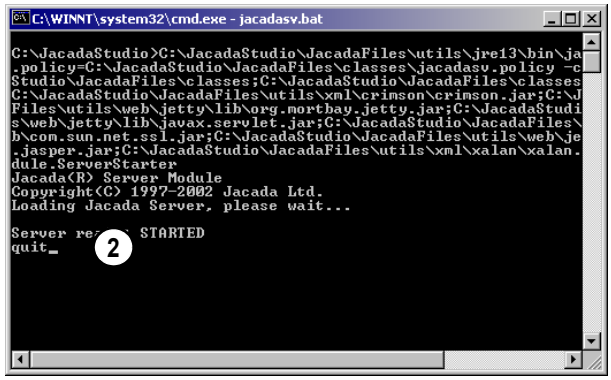


What your window should look like in an XHTML runtime

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by clicking the 'Exit' button in your runtime window.
2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes if a message appears.
3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.
4. Close your browser window.



Type 'quit' in the Jacada Server command window or press Ctrl+C to close the Jacada Server

Exercise 2 - Jacada Studio for iSeries

Create the Main Menu Window

Objectives:

- To provide an exercise that replicates the menu application window in the prepackaged iTutor application
- To show one possible way to build a menu window and to explain how it works
- To introduce the reader to the power of pre-built, reusable graphical components
- To explain the relationships between client-side graphical components and host-side navigation and process logic
- Briefly examine host code to understand what was generated and what process logic must be added to complete the program

In this exercise, you create the first Subapplication in the ITUTOR demo Application. You create this Subapplication in the IDK MYTUTOR Application, prepackaged for you. This Application already contains four of the eight Subapplications in the ITUTOR Application. The other four Applications you will build yourself, with a little help from this tutorial. This Application has a predefined KnowledgeBase, created especially for this tutorial. The purpose of this exercise is to allow you to experience how easily and speedily a window can be constructed and deployed with a predefined KnowledgeBase. After the window is built and deployed, feel free to go on to the next exercise. If you're up for an extra challenge, go on to the optional exercises at the end of the exercise, and build the KnowledgeBase components used in this window for yourself.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

The major steps to this exercise are:

1. **Window Design Specifications**
2. **Open the Jacada Studio for iSeries IDK**
3. **Open the MYTUTOR Application**
4. **Create the PMENU Subapplication**
5. **Add Representation to the Window**
6. **Generate Runtime and Transfer Files**
7. **About Host Code**
8. **Compile DDS and Program File on Host**
9. **Ensure the Jacada Monitor is Active**
10. **Run Application with a JAVA Client**
11. **Run Application with an XHTML Client**

The final product of your efforts in this exercise.

1. WINDOW DESIGN SPECIFICATIONS

The design specification for this window is that it include four buttons that provide access to four other windows in the Subapplication. A predesigned Window Layout will be associated with the window. This Window Layout will provide the standard look and feel of the window as well as an "Exit Application" button, that will end the host session when pressed.

* *Note :* The Window Layout will bring the GUI elements that are used in multiple Subapplications into the window. It will impose a GUI standard by provide a standardized look and feel and it will allow for the reuse of GUI elements.

2. OPEN THE JACADA STUDIO FOR ISERIES IDK

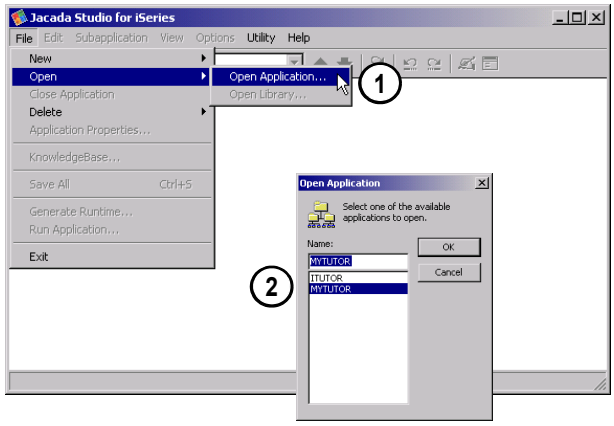
If you are not presently in the IDK, open the IDK by:

1. From your Windows **Start** menu > choose **Programs** > **Jacada Studio for iSeries** > **Jacada Studio for iSeries**.
   The Jacada Studio Interface Development Kit (IDK) is invoked.

3. OPEN THE MYTUTOR APPLICATION

In order to open the MYTUTOR Application from within the IDK interface:

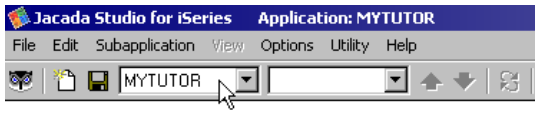
1. From the IDK **File** menu > choose **Open** > **Open Application**.
   The Open Application Dialog is invoked.
2. In the **Open Application Dialog** >
   Select MYTUTOR from the Application Name List > **Click OK**.



Choose File > Open > Open Application. Select MYTUTOR from the Application name List and click OK to open the MYTUTOR Application.

4. CREATE THE PMENU SUBAPPLICATION

In this next step, you create the PMENU Subapplication in the MYTUTOR semi-built Application. Make sure that the Application combobox shows that you are in the MYTUTOR Application before proceeding. In this exercise, you apply a prebuilt Window Layout called *Tutorial_MenuLayout* to your Subapplication. We will discuss Window Layouts later in this step.



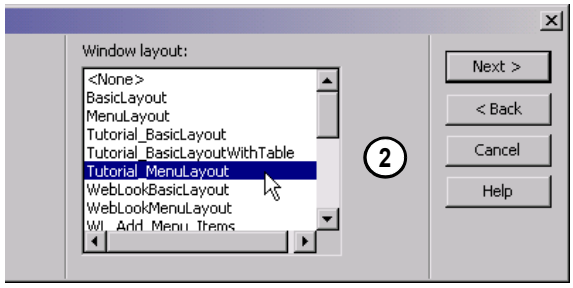
Make sure that the Application combobox shows that you are in the MYTUTOR application.

To create a new Subapplication:

1. From the **Subapplication Menu** > **New...**
   The New Subapplication Wizard is invoked.
2. In the **New Subapplication Wizard**, specify the following Subapplication properties:

Subapplication Name:	PMENU
Popup Window:	Unchecked
Window Layout:	**Tutorial_MenuLayout**
Subapplication Description:	<none>

3. Click **Finish** to exit the New Subapplication Wizard.

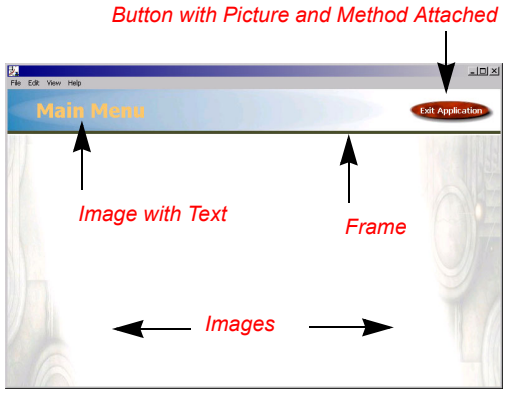


In the Select Window Layout step of the New Subapplication Wizard, choose the MenuLayout Window Layout.

Elements Added to the Window by the Tutorial_MenuLayout Window Layout

Did you notice the various elements that were brought into your Subapplication when you selected the *Tutorial_MenuLayout* Window Layout in the New Subapplication Wizard?

* *Note :* Keep in mind, the Jacada Studio IDK provides you with the ability to pre-define the Window Layout template for your own GUI look and feel.

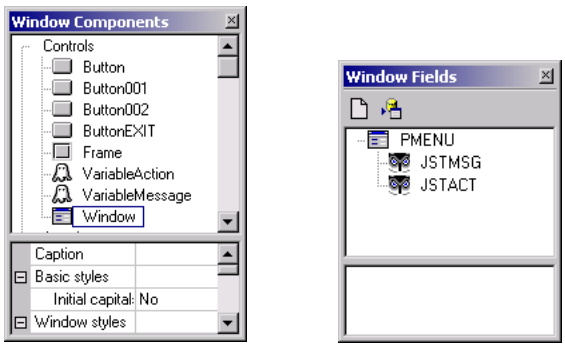


Elements added to the window by the Tutorial_MenuLayout Window Layout

Checking the Contents of the Window

Check the contents of the Window Components Palette to see the GUI components brought in by the Window Layout.

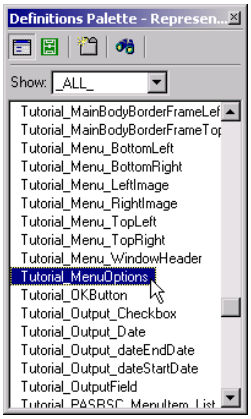
Check the contents of the Window Fields Palette to see the Window Fields brought in by the Window Layout.



Check which Fields and GUI components were added to the window by the Tutorial_MenuLayout Window Layout

5. ADD REPRESENTATION TO THE WINDOW

Representations can consist of one window component, or complex, consisting of multiple window components grouped into one. In this exercise, we drag the representation **Tutorial_MenuOptions** onto our window from the Definitions Palette. This representation is an example of a complex representation. It consists of four buttons whose properties were predefined for this demo. When you are done dragging the **Tutorial_MenuOptions** representation onto your window, and positioning the buttons, you are ready to Generate a Runtime, for your window is complete.



Drag the representation Tutorial_MenuOptions onto your window from the Definitions Palette.

** Note :* KnowledgeBase components created for this tutorial all have the prefix "**Tutorial_**". This is to allow you to easily distinguish them from the default KnowledgeBase components that ship with the product yet are not used in the tutorial.

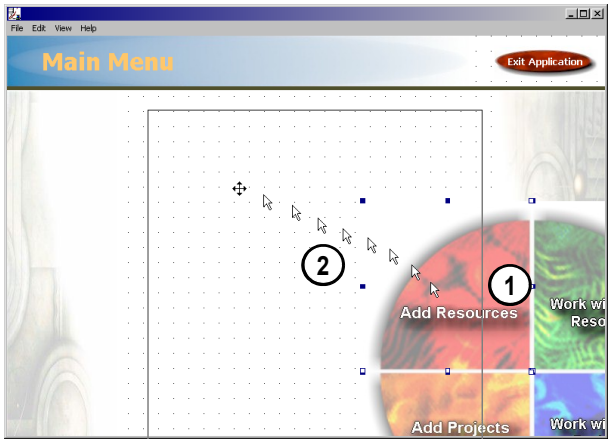
Positioning the Menu Buttons

When dragged onto the window, the menu buttons are not perfectly centered. To position the button group if all of the buttons are still selected:

1. **Click** on the group of buttons.
2. **Drag** the buttons to the center of the window.

If the buttons are not selected, to select the buttons and position them:

1. **Click** on the first button to select it.
2. **Shift+Click** on each of the other buttons to add them to the selection group
3. Position the buttons by **Clicking and Dragging** them to the center of the window.



Click and drag the selected group of buttons to position them in the center of the window.

** Note :* You can use any of the selection / manipulation options discussed in the *IDK Walk-through* section of the tutorial to select / manipulate the buttons as you like.

Save Subapplication

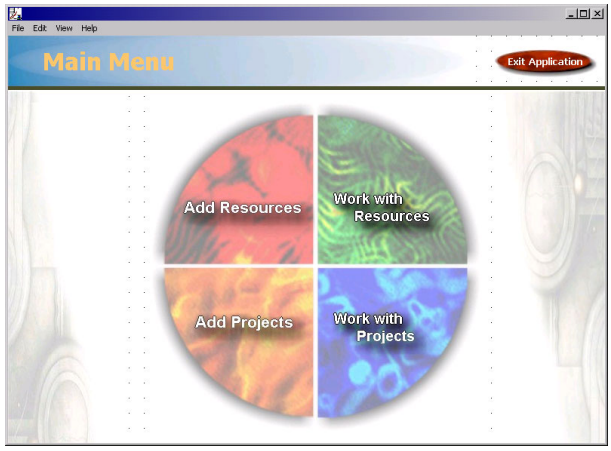
Before Generating a Runtime, you should save your Subapplication. To save your Subapplication:

From the Standard toolbar choose the  icon.

Save your changes by clicking the save button in the Standard toolbar.

The Finished Product

This is what your window should look like at the end of the last step.



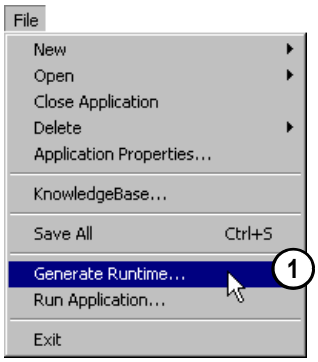
What your window should look like after positioning the menu buttons in the center of the window.

6. GENERATE RUNTIME AND TRANSFER FILES

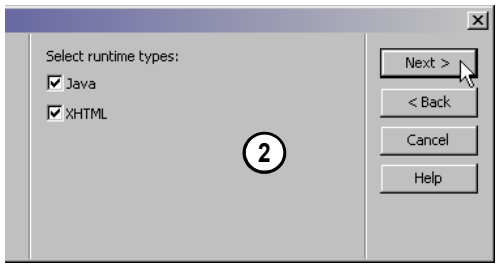
Compile the Subapplications into an executable, and transfer the DDS physical files to the host via the Generate Runtime Wizard. The Shell programs will not be transferred, since the host code has been prepackaged for you and already exists in the MYTUTORIAL host library. To generate a Runtime:

1. From the **File** menu > choose **Generate Runtime...** *The Generate Runtime Wizard is invoked.*
2. In the **Generate Runtime Wizard**, specify the following information, then Click **Next**:

Runtime Type:	Java and XHTML
Jacada Server Platforms:	Windows NT(2000) x86
Subapplications to Include:	All
Subapplications to Process:	All



Generate a Runtime in order to compile your Subapplication into an executable

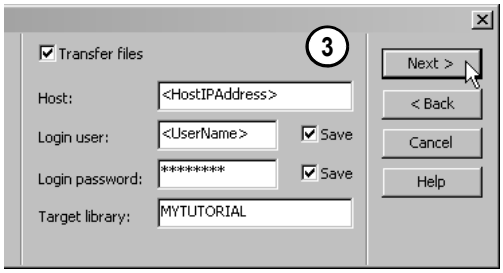


Select both Java and XHTML as your runtime types in the Generate Runtime Wizard.

3. In the **File Transfer** screen, specify the following information, then click **Next**:

Transfer files:	<Checked>
Host:	<YourHostIPAddress>
Login User:	<YourUserNameOnHost>
Login Password:	<YourPasswordOnHost>
Target Library:	**MYTUTORIAL**

* *Note :* If you are working in a multi-evaluator environment, specify your respective Library (i.e. TUTORIAL01) as the Target Library.

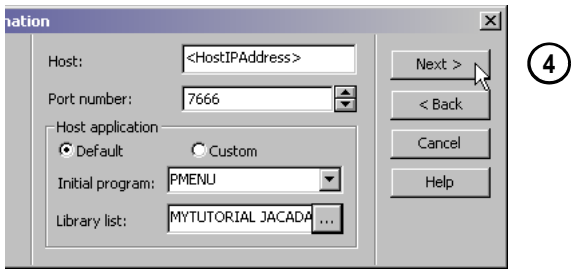


Fill out the File Transfer screen of the Generate Runtime Wizard

4. In the **Specify Host Connection and Application Information** screen, specify the following information, then Click **Next**:

Host:	<YourHostIPAddress>
Port Number:	7666
Initial Program:	PMENU
Library List:	**MYTUTORIAL JACADA**

* *Note :* If you are working in a multi-evaluator environment, remember to have your respective Library (i.e. TUTORIAL01) be the first library in the Library List entry and include the JACADA library after your library.

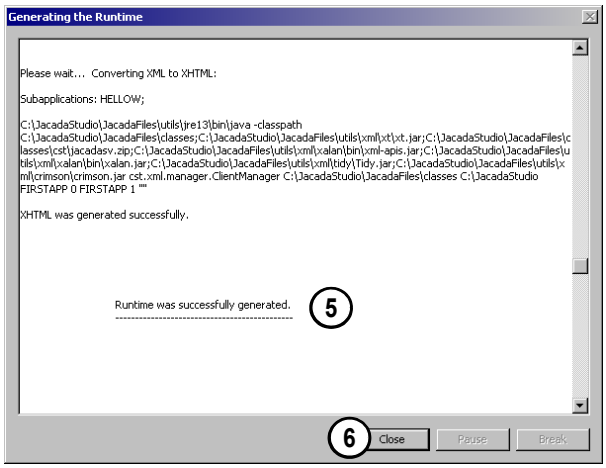


Fill out the Host Connection and Application Information screen of the Generate Runtime Wizard

5. Click **Finish** to come out of the Generate Runtime Wizard, and commence with the compilation process. *The Generating the Runtime dialog appears.*

   Look for the message "Runtime was successfully generated" at the end of the compilation process; this confirms that you can continue with the next step.

6. Click **Close** to come out of the Generating the Runtime dialog.



The Generating the Runtime Dialog shows the status of a Runtime Generation

7. ABOUT HOST CODE

To allow you to run through this tutorial at a reasonable rate of speed, the host code for the MYTUTOR Application has been prepackaged for you, and was installed in the MYTUTORIAL library created on the host machine when you installed this product. In a real-world scenario, you would write your host logic in this step of the workflow. Since the host code has been prepackages for you, there is no need and you can go on to the next section.

8. COMPILE DDS AND PROGRAM FILE ON HOST

Verify that the JACADA library and the MYTUTORIAL library are in your library list.

Compile the DDS physical file, **PMENU$D** and the RPG program file **PMENU** in your **MYTUTORIAL** library.

* *Note :* If the FIRSTAPP application is still in your library list, make sure to replace it with MYTUTORIAL.

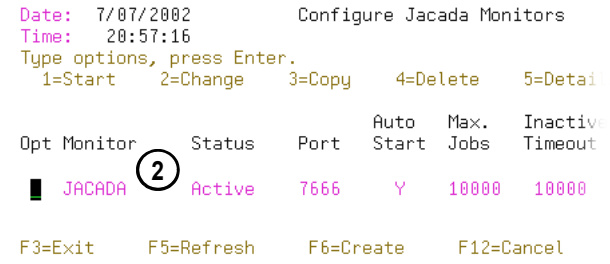
9. ENSURE THE JACADA MONITOR IS ACTIVE

To ensure that the Jacada Monitor is active:

1. Type **CFGJACMON** in the iSeries command line.
2. In the Configure Jacada Monitors screen, make sure that the word '**Active**' Appears in the *Status* column to the right of the Jacada Monitor.

If the Jacada monitor is not active:

Type **1** in the *Opt* column to start the Jacada Monitor.



Make sure that the word 'Active' Appears in the Status column to the right of the Jacada Monitor

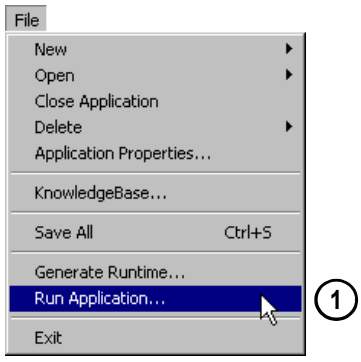
10. RUN APPLICATION WITH A JAVA CLIENT

Run the executable created during the Runtime Generation process from within the development environment:

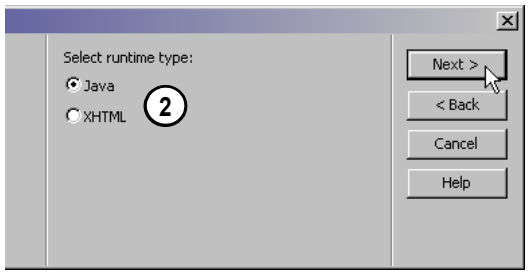
1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*
2. In the **Run Application Wizard**, agree to the default Runtime properties, by clicking the **Next** button, when prompted for:

Runtime Type:	Java
Web Server:	Integrated HTTP Service
Application URL:	http://localhost:8080/ MYTUTOR.html

3. Click **Finish** to come out of the Run Application Wizard. *The Jacada Server is activated and your Default Browser window is opened to the Jacada <ApplName>.html page.*



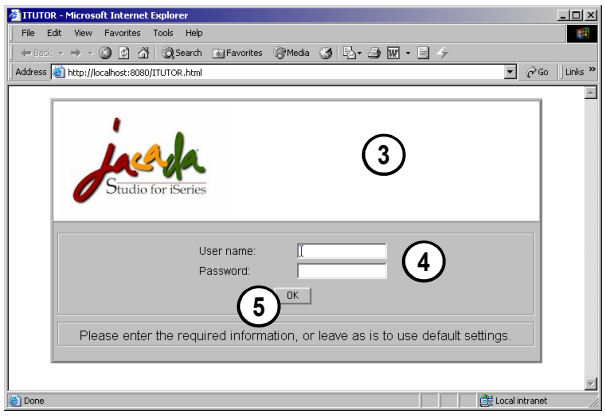
Run your Application through the development environment by choosing File > Run Application



Choose Java as your Runtime Type in the Run Application Wizard

4. Type your iSeries **Username** and **Password** into the appropriate fields.

5. Click the **OK** button to run your Java client Application.



Type your iSeries username and password into the appropriate fields in the <Applename>.html page.

* *Note :* Jacada Studio allows you to customize the look of the sign-on *<ApplName>.html* window. The example to the right shows a customized window.

The Finished Product

This is how your window should look after running your Application with a Java client.

IMPORTANT! - The only working button in this window is the 'Work with Resources' button. In the following exercises, you will build the 'Add Resources', 'Work with Projects' and 'Add Projects' Subapplications. Only click on the 'Exit Application' or 'Work with Resources'. Clicking on any other button will result in an error message and a termination of your host session (much as it would if you were working with DDSs).

* *Note :* If you clicked a non-working button go on to the 'Close the Jacada Server'section below and skip step number one.

The final product of your efforts in this exercise.

Close the Jacada Server

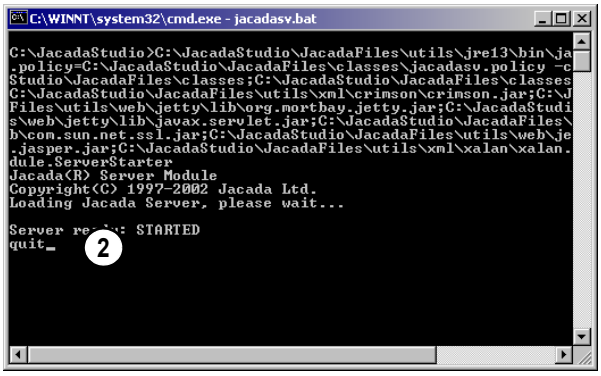
When you are done running your Application

1. Exit the Application and end your host session by clicking the 'Exit Application' button in your runtime window.

2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes if a message appears.

3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.

4. Close your browser window.



Type 'quit' in the Jacada Server command window or press Ctrl+C to close the Jacada Server

11. RUN APPLICATION WITH AN XHTML CLIENT

To run your Application with an XHTML client:

1. From the **File** menu > choose **Run Application...**

2. In the **Run Application Wizard**, specify the following settings:

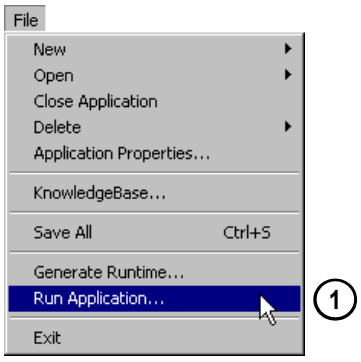
Runtime Type:	XHTML
Port Number:	8080
Application URL:	http://localhost:8080/MYTUTOR-xhtml.html

3. Click **Finish** to come out of the Run Application Wizard. *Wait a couple of seconds. The Jacada Server is activated in a DOS command window and your Default Browser window is opened to the Jacada <ApplName>.html page.*

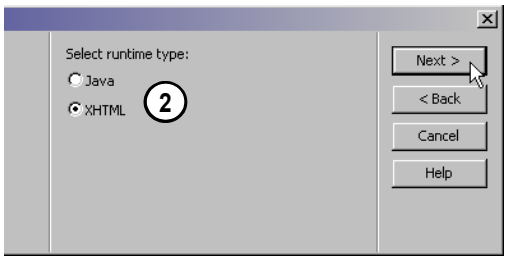
4. Type your iSeries **Username** and **Password** into the appropriate fields.

5. Click the **OK** button to run your HTML client Application.

* *Note :* Don't forget to close the Jacada Server when you're done.



Run your Application through the development environment by choosing File > Run Application



Choose XHTML as your Runtime Type in the Run Application Wizard

Exercise 2 - Jacada Studio For iSeries

How It Works: The Main Menu Window

If you feel comfortable with the level of detail provided in this exercise, feel free to skip this section and go on to the next one. If you'd like to find out a bit more about how it all works, keep reading. In this section you learn about how everything comes together during runtime. Are you ready? Well, what are you waiting for?

Look for the ActionPerformed Method Attached to the OnClick Event of the Menu Buttons

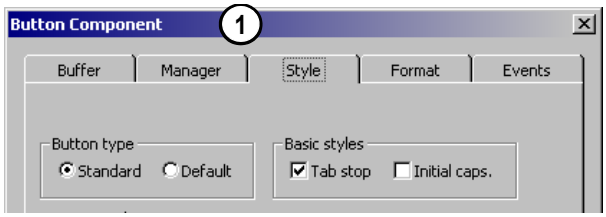
To view the association between the menu buttons and their associated Methods:

1. **Double click** on any one of the menu buttons. *The Component Properties Dialog appears.*

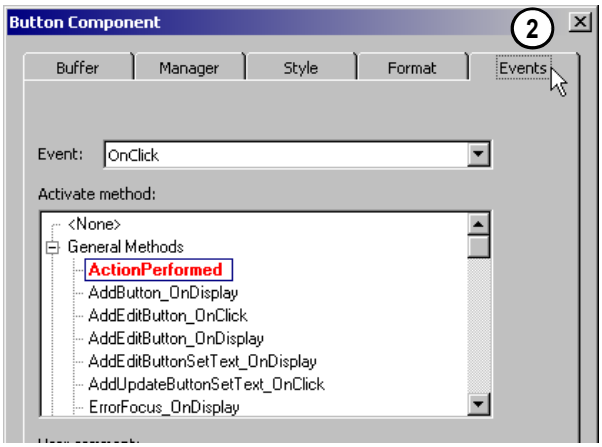
** Note :* The Component Properties dialog automatically displays the properties of the component that was double clicked.

2. Go to the **Events Tab** of the Component Properties dialog. *You can see that the ActionPerformed Method is attached to the OnClick event of this component.*

3. Click **Cancel** to come out of this dialog.



Double click on one of the menu buttons to access the Component Properties Dialog



You can see that the ActionPerformed Method is attached to the OnClick event of the menu button component.

Events

Many graphical controls have Events associated with them. Events can be thought of as logical associations to physical actions such as the clicking of a mouse on a button. When a Pushbutton is physically clicked at runtime, the graphical client can respond to a logical event called "OnClick" and do something with the knowledge of that action. Even the displaying of a control by the client runtime code itself is a physical action that has an "OnDisplay" event associated with it. Methods are used to define the actions that are performed when an Event is sensed. Consequently, a Method can be triggered by the OnClick event that occurs when a user clicks on a control such as a button.

Window Component Names

In the iSeries green screen world, keyboard actions are relatively limited – 24 Function Keys, Enter, Help Home, Page Up, and Page Down. These "events" are automatically interpreted and managed by the operating system, and interactive programs are typically notified of keyboard activity via Indicators. When a Function key is pressed, the operating system is notified of a keyboard Fkey 'action', strips off the FKey number, and sets an associated Indicator that your program can respond to. Although Jacada Studio for iSeries fully supports Indicator activity, we have also provided an alternative that allows your programs to easily recognize graphical events in a fashion very similar to Indicators. We provide a Method that is associated with the OnClick event for graphical push-button.

By using a Prefix/Suffix syntax when naming pushbutton controls a developer can put this Method to work to send an action notifier back to his RPG or COBOL program. The naming syntax for a pushbutton control is Prefix "Button" and Suffix "Action". The Suffix or "Action" portion may be fairly generic such as "Update", "Add", or "Exit". The button name would then be ButtonAdd, or ButtonExit. A more specific implementation is used in the PMENU pushbuttons where each button action identifies a function request.

The button name ButtonWWR is automatically decomposed by the **ActionPerformed** Method to send a value of "WWR" back to the host menu program (PMENU) in a hidden field called **JSTACT**. The menu program can then initiate a call to a program based on logic that evaluates the content of field JSTACT. In the case of the PMENU program the "Work With Resources" program is called when the value of JSTACT is equal to "WWR". (The "action" field JSTACT is predefined in the Jacada Studio Knowledgebase and is automatically included on every Jacada Studio for iSeries shipped window layout as a hidden field.)

```
C* Exit Not Clicked
C* "Change original shell which had IFEQ"
C           JSTACT    IFNE 'EXIT'
C*
C* Work with Projects Clicked
C           JSTACT    IFEQ 'WWP'
C                     CALL 'PPROJ'
C                     MOVEL*BLANKS   JSTACT
C                     ENDIF
C*
C* Work with Resources Clicked
C           JSTACT    IFEQ 'WWR'
C                     CALL 'PRESO'
C                     ENDIF
```

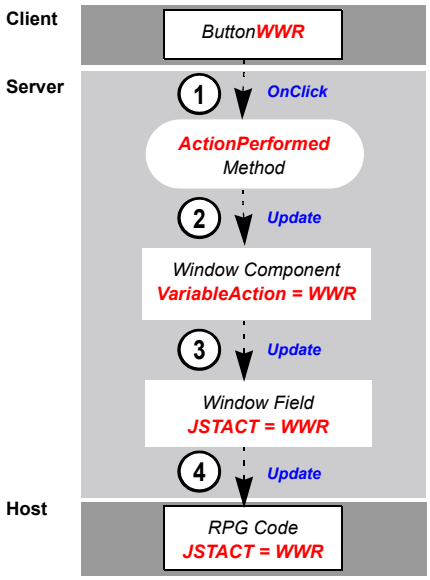
Sample Code taken from the program logic written for the PMENU Subapplication of the ITUTOR Application.

Values get passed to the host program during runtime in the following way:

1. The ***ACTION*** string (WWR) of the button's component name is extracted by the ***ActionPerformed*** Method when an ***OnClick*** event occurs.

The ActionPerformed Method:

2. Updates the ***VariableAction*** variable with the value of the ACTION string (WWR).
3. Passes the value of the ACTION string (WWR) from the VariableAction variable to the ***JSTACT*** field in the DDS Physical File (the buffer).
4. Sends value **WWR** in JSTACT to the RPG code.



Exercise 2 - Jacada Studio For iSeries

Optional Exercises: The Main Menu Window

If you feel comfortable with the level of detail provided in the previous exercise, feel free to skip this exercise and go on to the next one. If you'd like to "break it down" some more - to work with the "nuts and bolts" a little bit - how about building the menu items yourself? In this section you learn all of the steps necessary to turn the simple representations in the default KnowledgeBase that ships with the product, into complex representations like the Tutorial_MenuOption representation used in the previous exercise. Are you ready? Well, what are you waiting for?

The major steps to this exercise are:

1. **Delete the Buttons that Were Inserted by the Tutorial_MenuOption Representation**
2. **Drag a Simple Button Representation onto Window**
3. **Set Button Component Name**
4. **Set Button Style Properties**
5. **Copy Paste Button**
6. **Position Buttons in Center of Window**
7. **Generate Runtime and Run Application**

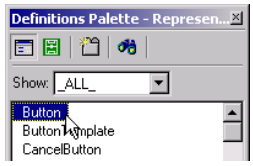
The final look of the Representation that you will build in this exercise.

1. Delete the Buttons that Were Inserted by the Tutorial_MenuOption Representation

Simply select each of the buttons on your window and press the **Delete key** on your keyboard to delete the items. When the "*Delete Selected Controls*?" message appears, click **Yes**.

2. Drag a Simple Button Representation onto Window

We will first create the "*Add Resources*" button, the top left button in the button arrangement. From the Definitions Palette, drag a simple **Button** representation onto the window.

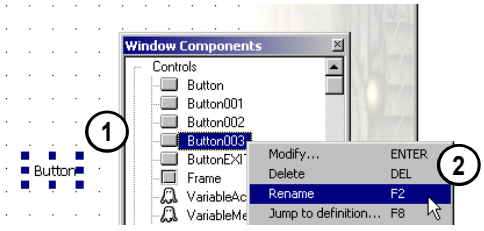


Drag a simple Button representation onto your window from the Definitions Palette.

3. Set Button Component Name

In the Window Components Palette, set the button component name property, to give the button an explicit name that will be used to call the button from the host code.

1. **Select** the added Button in the window. *The added button's automatically given component name will subsequently be selected in the Window Components Palette.*
2. **Right Click** on the button name > **Rename**

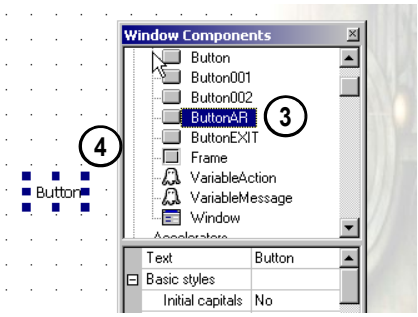


Rename the component in order to call the button explicitly from the host code.

* *Note :* If the Window Components Palette is behind the window and being obscured by it, use the F12 function key on your keyboard to bring the palette to the top of the window. The F12 function key brings all palettes to the top layer of visible items.

3. Type '**AR'** in **ALL CAPS** after the button component name to create the component name **ButtonAR**

4. Verify that the correct button has been renamed by selecting the button component name that you created **ButtonAR** in the *Window Components Palette*. This action will consequently select the button representation in the window.

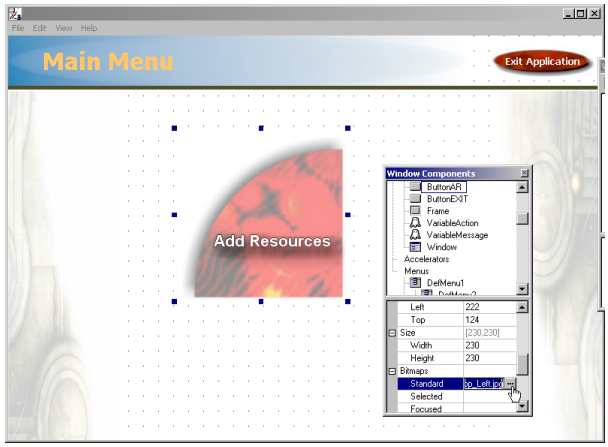


Change the button's component name to ButtonAR. Select the renamed component and verify that the correct button component is selected in the window

4. SET BUTTON STYLE PROPERTIES

In the Component Editing section of the Window Components Palette, set the button style properties

Text:	<none>
Button Styles: **Squared Corners**	Yes
Size: **Width:** **Height:**	 230 230
Images: **Standard**	 \JacadaStudio\appls\mytutor\Bitmaps\top_left.jpg
Placement: **Bitmap Relative:**	 On Top



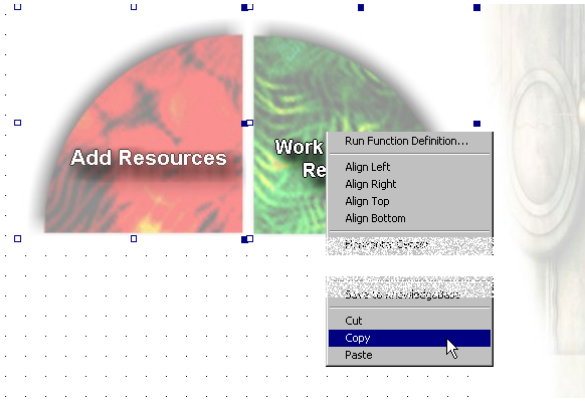
This is what you should see after having set the Button Style properties.

5. COPY PASTE BUTTON

The easiest way to create the other three menu buttons without again setting all the component properties that you set in the previous step is to copy paste the button that you created. You can use your default windows shortcuts of:

Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+Z	Undo

Or you can **Right Click** on a button and choose *Copy* or *Paste* from the Design View Right Mouse Button Menu.

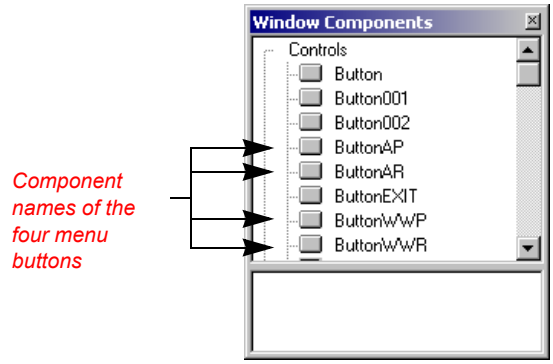


Use the Window shortcuts to copy paste components to and from your screen or use the Right Mouse Button Menu

Change Component Names and Image Associations

Make sure that you change the *images associated* with the new buttons and their *component names* as follows:

Button Component Name	Attached Bitmap
ButtonWWR	top_right.jpg
ButtonWWP	bottom_right.jpg
ButtonAP	bottom_left.jpg



Change the component names of the new buttons

6. POSITION BUTTONS IN CENTER OF WINDOW

Position the buttons in the center of the window using the *Control Editing and Manipulation Options discussed in step 12 of the IDK Walk-through* section of this tutorial.

7. GENERATE RUNTIME AND RUN APPLICATION

You are done! Do you want to run your Application? It's easy:

1. **Generate a Runtime**, transferring files to the host.

2. Make sure the Jacada Monitor is Active.

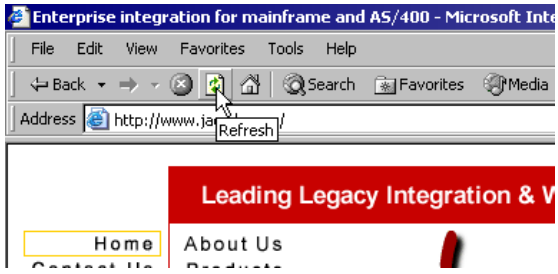
3. Run the Application.

How did you do?

* *Note :* Did you remember to close your application and the Jacada Server?

It is not necessary to recompile the DDS physical file and RPG program on the iSeries unless a change was made to either (a) the number of fields in the IDK subapplication or (b) the RPG program. Although it is not necessary to recompile in these situations, we recommend you recompile upon each change to the subapplication for the duration of this tutorial, until you are thoroughly acquainted with the workflow. Recompiling your DDS physical file and RPG program on the iSeries with every change to the subapplication will spare you unnecessary troubleshooting later.

* *Note :* If you did not exit your browser since the last time you ran a runtime, you may receive a "Version Mismatch" message. In this case, hold down the Control key on your keyboard while clicking the 'Refresh' button in your browser window to clear your browser's cache.



Hold down the Control key on your keyboard while you click the 'Refresh' button in your browser window to clear your browser's cache if you receive a "Version Mismatch" error while trying to run a runtime.

Exercise 3 - Jacada Studio for iSeries

Create the Add / Edit Resource Window

Objectives:

- To provide an exercise that replicates the Add/Edit Resource window in the pre-packaged iTutor application
- To build a window in which new data can be Added or existing data can be retrieved and Updated
- To examine dynamic GUI display alternatives to Indicator driven Display File behaviors
  - Conditional runtime display driven by client events (based on program mode of Add or Update)
  - Resource # show/hide depending on mode
  - Update or Add button displayed depending on mode
- Passing variable text display values from a host program
- To gain an understanding of the relationship between fields and their graphical representations
- To gain a better understanding of client-controlled application activity as an alternative to host-controlled application activity
  - Data validation at the client or server instead of the host
- Using a graphical link control to launch an external URL in a separate browser frame
- Briefly examine the host code to understand what was generated and what process logic must be added to complete the program

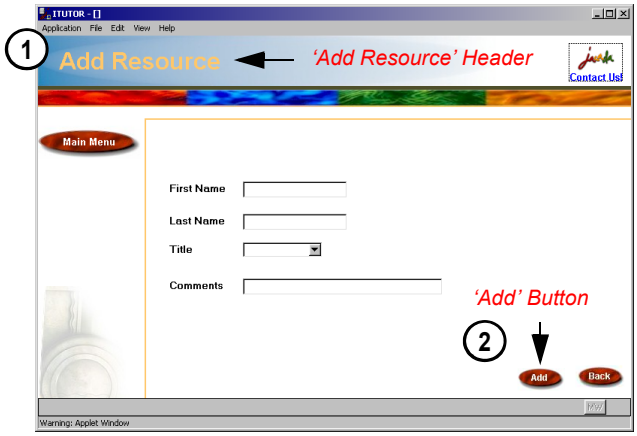
In this exercise, you create the Add / Edit Resources window of the ITUTOR Application. Your MYTUTOR Application should already contain five of the eight Subapplications in the ITUTOR demo, four that were prepackaged for you and the Main Menu Window that you created in the previous exercise. This window is used both as an Add Resources window and as an Edit Resources window, depending on the entry point chosen by the user. The purpose of this exercise is to expose you to the use of server logic as a means of controlling conditional variations in the interface, and as an alternative to driving those variations from the host using Indicators and DDSs. The Method implementations in this exercise are good representations of this capability. The value idea is to simplify host business logic by reducing presentation-oriented coding that could be managed in the presentation layer. In this exercise, you will experience how easily and speedily a window can be constructed and deployed, with a predefined KnowledgeBase.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

The major steps to this exercise are:

1. **Window Design Specifications**
2. **Create the PADDRE Subapplication**
3. **The Window Header**
4. **Add Fields to the Window**
5. **The "Resource #" Representation**
6. **The Add and Update Buttons**
7. **Error Handling**
8. **Generate Runtime and Transfer Files**
9. **About Host Code**
10. **Compile DDS and Program File on Host**
11. **Run Application**



The final product of your efforts in this exercise. The look of the PADDRE Subapplication, when accessed through 'Add' mode.

1. WINDOW DESIGN SPECIFICATIONS

First, you must define the differences in the Subapplication interface, between when it is accessed through 'Add' mode and when it is accessed through 'Edit' mode. The differences are:

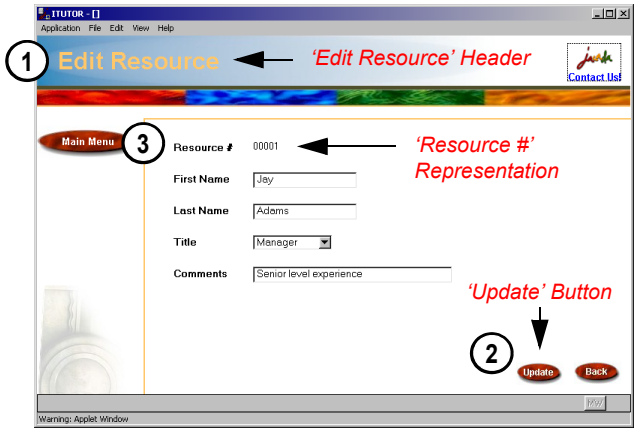
In 'Add' mode:

1. The window header says "Add Resource".
2. There is an 'Add' Button, to the left of the 'Back' button in the bottom right corner of the window.

In 'Edit' mode:

1. The window header says "Edit Resource"
2. There is an 'Update' Button, to the left of the 'Back' button in the bottom right corner of the window.
3. There is a "Resource #" representation.

* *Note :* You will reuse GUI elements from the PMENU (Main Menu) window in this Subapplication. You will do so via the *Window Layout* feature.



The final product of your efforts in this exercise. The look of the PADDRE Subapplication, when accessed through 'Edit' mode.

2. CREATE THE PADDRE SUBAPPLICATION

In this next step, you create the PADDRE Subapplication in your MYTUTOR Application. In this exercise, you apply a prebuilt Window Layout called *Tutorial_BasicLayout* to your Subapplication.

Open the IDK and the MYTUTOR Application

To open the IDK interface and the MYTUTOR Application:

1. From your Windows **Start** menu > choose **Programs** > **Jacada Studio for iSeries** > **Jacada Studio for iSeries**
   The Jacada Studio Interface Development Kit (IDK) is invoked.
2. From the IDK **File** menu > choose **Open** > **Open Application**.
   The Open Application Dialog is invoked.
3. In the **Open Application Dialog** > **Select MYTUTOR** from the Application Name List > **Click OK**.

Create the PADDRE Subapplication

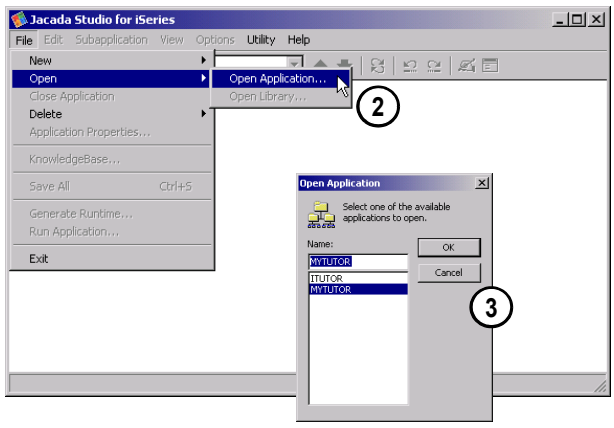
1. From the **Subapplication Menu** > **New ...** *The New Subapplication Wizard is invoked.*
2. In the **New Subapplication Wizard**, specify the following Subapplication properties:

Subapplication Name:	PADDRE
Popup Window:	Unchecked
Window Layout:	**Tutorial_BasicLayout**
Subapplication Description:	<none>

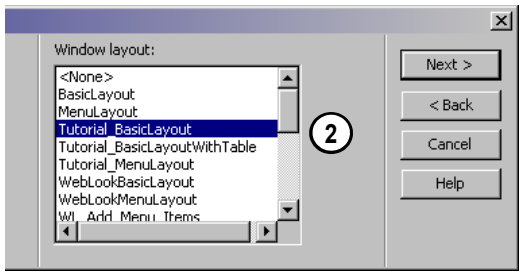
3. Click **Finish** to come out of the New Subapplication Wizard.

Leveraging Common GUI Look Between Layouts

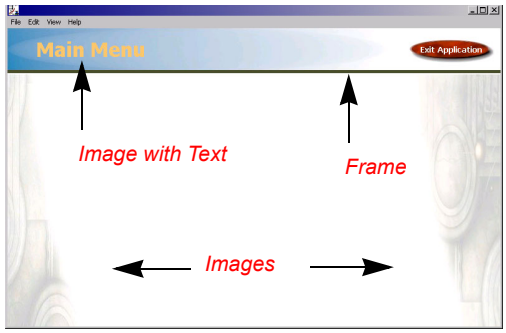
Do you remember the Window Layout that you applied to the Main Menu window in the last exercise? Did you notice the similarities between the Window Layout used in the last exercise and the Window Layout used in this one? The Window Layout feature allows us to leverage the common look between Window Layouts by re-using elements from one Window Layout in another. You can also modify existing Window Layout templates to enhance the functionality and GUI look between windows.



Choose File > Open > Open Application. Select MYTUTOR from the Application name List and click OK to open the MYTUTOR application.



In the Select Window Layout step of the New Subapplication Wizard, choose the Tutorial_BasicLayout Window Layout.

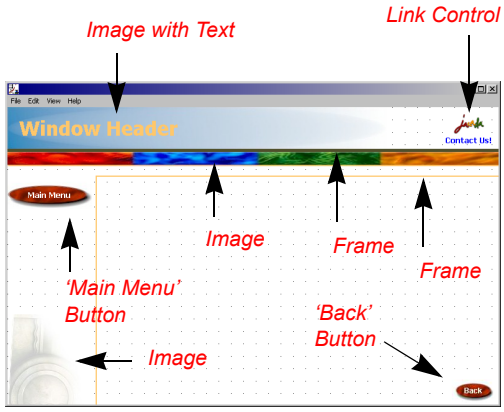


Elements that were used in the Tutorial_MenuLayout Window Layout used in the Main Menu window that were re-used in the Tutorial_BasicLayout Window Layout applied to this window.

In the **Tutorial_BasicLayout** Window Layout applied to this window, a 'Main Menu' button, 'Back' button and link to the Jacada Website were added to enhance the functionality and GUI look of the window.

Elements added to the Window by the Tutorial_MenuLayout Window Layout

Did you notice the various elements that were brought into your Application when you selected the *Tutorial_BasicLayout* Window Layout in the New Subapplication Wizard?

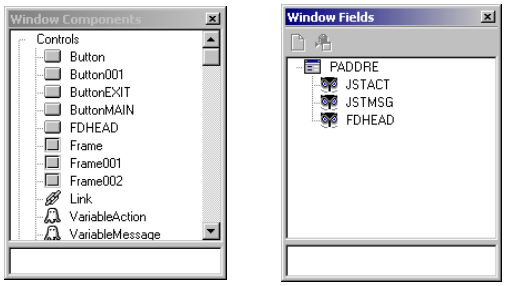


Elements added to the window by the Tutorial_BasicLayout Window Layout

Checking the Contents of the Window

Check the contents of the Window Components Palette to see the GUI components brought in by the Window Layout.

Check the contents of the Window Fields Palette to see the Window Fields brought in by the Window Layout.



Check which Fields and GUI components were added to the window by the Tutorial_BasicLayout Window Layout

FYI: The Link Control

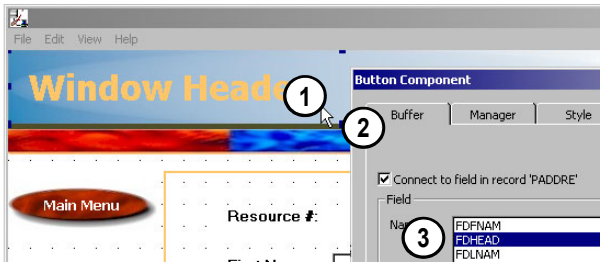
A Link control was placed in the upper right corner of the window when you applied the Tutorial_BasicLayout window layout in the last step. This link control has the Jacada website defined as a resource and will take the user to the Jacada website when he clicks on the link during runtime. The link representation can have an image attached to it, or it could just be plain text. This link control was defined globally in the KnowledgeBase, so that it could be used throughout the Application via the window layout.

In Jacada Studio, links may be used as URL Links or as Event Links. Event Links are controls that activate the IDK OnClick event. URL links are controls that points to an Http resource. When URL links are pressed they perform an action that is determined by the resource type. Some Examples: Open an HTML Page, Show an image in a separate window, Send e-mail, Download a File, Open a Word Document.

3. THE WINDOW HEADER

This example is to show you how a common behavior can be controlled. We defined the text that appears in the header in the host code, and passed it to a field attached to the window header representation. Let's have a look:

1. **Double click on the Window Header representation** that was brought into the window by the Tutorial_BasicLayout window layout. *The Component Properties Dialog appears.*



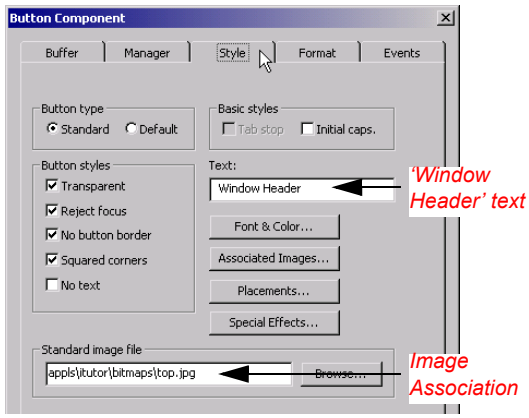
Double Click a control and go to the Buffer Tab to see and edit the connection between a window component and a field.

* *Note :* In the Style Tab of this dialog, you can see that this component is just a simple button with an image association and the text "Window Header". We will override this text setting via the RPG code in the host program. Check the Events Tab and you will see that there is no Method associated with this control.

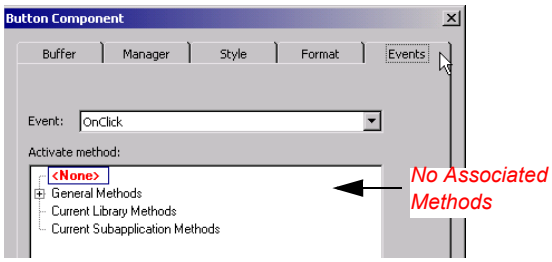
2. Go to the **Buffer Tab**.

3. **View the connection** between the window component and the FDHEAD field.

* *Note :* This connection will be used to transfer the appropriate text to this header, depending on the mode in which this window is accessed.



In the Style Tab of this dialog, you can see that this component is just a simple button with an image association and the text "Window Header"

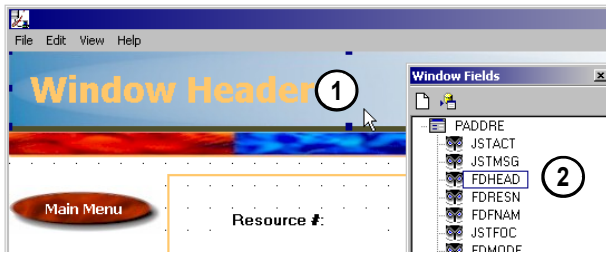


Check the Events Tab to see that there is no Method associated with this control.

You can also view the connection between the window component and the FDHEAD field:

1. **Select the Window Header** component in the window.
2. Look at the Window Fields Palette. The FDHEAD field is selected.

* *Note :* The association between a component and a field can only be modified through the Component Properties Dialog.



Select a control and look in the Window Fields Palette to see the connection between a window component and a field. You cannot modify the connection from the Window Fields Palette.

Now, let's see the code.

1. Open the PADDRE program file on the host.
2. Notice the 'Window Title' I specifications.
3. Notice the section that populates the window title.

```
 * Place your I specifications here
 * Window Title
I              '   Edit Resource'    C         ETITLE
I              '   Add Resource'     C         ATITLE
 ************************************************
 * Populate Window Title
 ************************************************
C           MODE      IFEQ 'EDIT'
C                     MOVELETITLE    FDHEAD
C                     ELSE
C                     MOVELATITLE    FDHEAD
C                     ENDIF
```

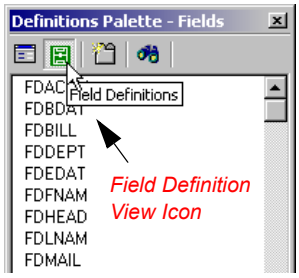
2
3

Sample Code taken from the program logic written for the PADDRE Subapplication of the ITUTOR application. Code populates FDHEAD field.

4. ADD FIELDS TO THE WINDOW

Until now, when you needed to add a Representation to the window, you dragged it from the Definitions Palette, which was set to *Representation Definition View*. In this step, you will add Representation Definitions to the window by dragging fields onto your window from ***Field Definitions View*** of the Definitions Palette. You will then choose the representations associated with the field from a "Short List".

Click the ***Field Definitions View Icon*** in the Definitions Palette to see the list of fields defined in the KnowledgeBase.

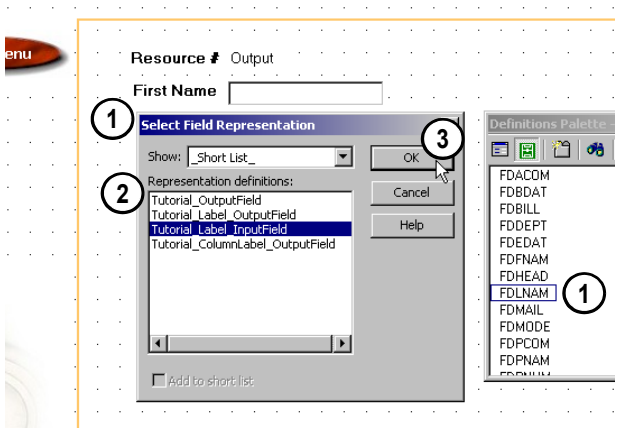


Click on the Field Definitions View Icon in the Definitions Palette to see the list of fields defined in the KnowledgeBase.

To add the remaining Representations to your Subapplication, refer to the table below:

Window Fields	Representations
FDRESN	Tutorial_Label_OutputField
FDFNAM	Tutorial_Label_InputField
FDLNAM	Tutorial_Label_InputField
FDTITL	Tutorial_Label_Combobox_Title
FDRCOM	Tutorial_Label_InputField

1. **Drag the fields** in the *Window Fields* column onto the window, from the Definitions Palette.
   The Select Field Representation Dialog appears.
2. **Select the representation** in the *Representations* column, from the _Short List_ in the Select Field Representation Dialog.
3. **Click OK** to come out of the dialog. *Your representation is added to the window. It is associated with the field that you dragged from the Definitions Palette.*



Drag fields onto the window, then choose the correct representation from the _Short List_ in the Select Field Representations Dialog

FYI: The Field Driven Design Concept

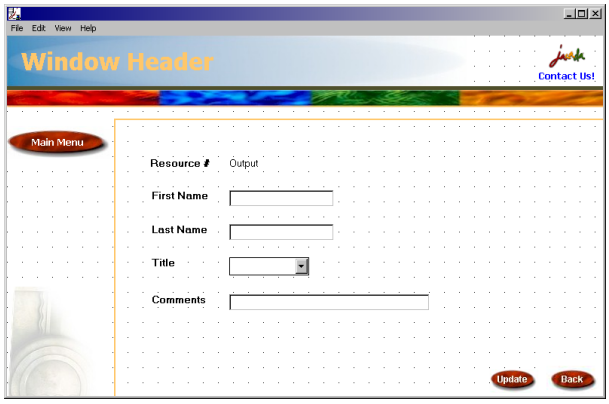
In the green-screen world, the appearance and behavior of a field on a Display File are controlled with Attributes and Indicators. Variations in appearance are fairly limited to text-oriented attributes like coloration, blinking, reverse image, etc. Additionally, a field may have variable behavior such as being an Output Only field under one condition and Input Only under another circumstance. Each time a new Display File is created, the display attributes for any particular field must be redefined. Some Field properties such as length, data type, and Field Text or Column Headings can be standardized by their definitions in Physical File DDS, but most display properties must be recorded in each Display File DDS, which is not only tedious but opens the application to inconsistencies among interfaces.

Jacada Studio for iSeries has a simpler, more consistent approach for allowing fields to be graphically "represented" in various fashions at different times in an application. This concept of representation not only allows full exploitation of the wide range of appearance and behavior permutations available in the graphical world, but also supports the enforcement of consistent graphical standards at the field level. For example, a single field may be used at different times in the application as Input Only, Output Only, or Both. Similarly, a field may be graphically represented at different times by a Check Box, a Combo Box, or a Radio Button on Input, and a Static Control with a large, red, bold font on Output. By associating a field with its most often used representations, and storing these associations in the Knowledgebase, a developer builds a collection of standard representations for each field that can be rapidly reused instead of re-defining them for each window. We call this collection of most often used representations for any given field its 'short list'.

** Similar to the Text or Column Heading properties in PF DDS, Jacada Studio can assign a standard Label at the field level and implement that Label consistently in a field's short list of representations.*

Arrange the Position of the Added Representations

Arrange the position of the added representations according to the image to your right. *For more information on control manipulation, please refer to the IDK Walk-through section of this tutorial.*



Arrange the position of the added representations like this.

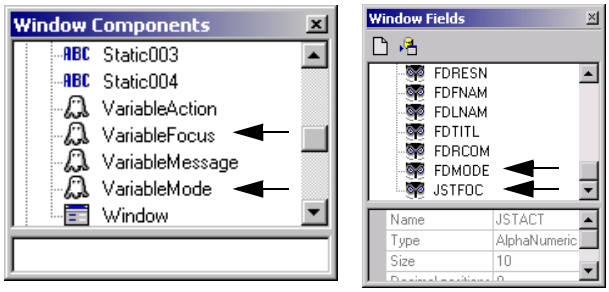
Add Variable Representations to the Window

Most representations have a visible control on the window. In this step, add the ***Variable*** Representation Definitions in the table below to the window by dragging ***Window Fields*** onto your window from ***Field Definitions View*** of the Definitions Palette. Then, choose the ***Representations*** associated with the field from a "Short List" of favorites. Drag the following fields onto your Subapplication window:

Window Fields	Representations
JSTFOC	Tutorial_Variable_Focus
FDMODE	Tutorial_Variable_Mode

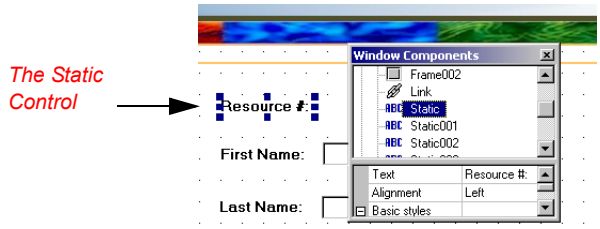
* *Note :* When you drag these fields onto the window, it will seem as if nothing has happened, and no representation will appear. This is because these fields are attached to ***hidden*** variable representations.

To make sure that these fields were added to the window: Check that the fields **JSTFOC** and **FDMODE** reside in the list of fields in the Window Fields Palette. Check that the ***VariableFocus*** and ***VariableMode*** variables exist in the list of window components in the Window Components Palette. *(they will have an icon with a picture of a ghost)*



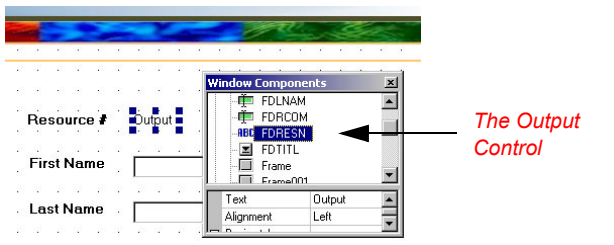
Check the contents of the Window Components and Window Fields palettes to ensure the variable representations have been added to your Subapplication.

* *Note :* See the 'How It Works' section at the end of the exercises for a detailed explanation of the runtime behavior between client, server and host.The VariableFocus representation connected to the JSTFOC field is explained in the next exercise.



5. THE "RESOURCE #" REPRESENTATION

According to our project requirements, the Resource Number controls must appear on the window if it is accessed in Edit mode, but not if it is accessed in Add mode. Let's use a show/hide Method and the FDMODE field again to control this interface variation.



The Static and Output controls that make up the 'Resource #' representation.

The "Resource #" representation that we added to our window consists of two controls: The *Static* control (with the text "Resource #") and the *Output* control (with the text "Output"). The condition of whether the "Resource #" controls are hidden or displayed is according to the mode you are in.

The ***Tutorial_HideShowControlBasedOnMode*** Method attached to the OnDisplay event of both components of this representation, checks the value of the *VariableMode* variable (updated with the value of FDMODE), and then hides or shows the control based on its value. The value of the FDMODE field is determined by host logic.

To connect the "Resource #" representation's ***Static*** component to this Method:

1. Double click the static component with the text "Resource #" in your window. *The Control Properties Dialog appears.*
2. Go to the **Events Tab**.
3. In the **Events combobox** > select the **OnDisplay event**.
4. Select the **Tutorial_HideShowControlBasedOnMode** Method from the General Methods tree.
5. Click on the **Link** button to link the Method to the OnDisplay event of this control.
6. Go to the **Manager Tab**.
7. Check the **Runtime data flow** checkbox.

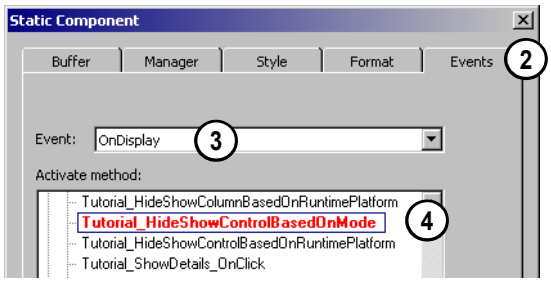
* *Note :* You must check the runtime data flow checkbox because there must be data flow from the host to the server in order for the method that performs the hide/show according to the value of the FDMODE field to update with the value of FDMODE from the host code during runtime.

8. Click **OK** to come out of the *Control Properties Dialog*.

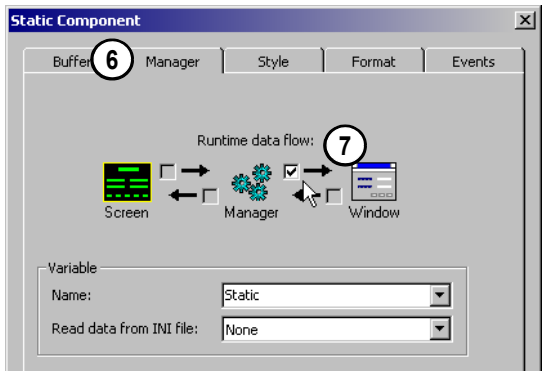
To connect the "Resource #" representation's ***Output*** component to this Method:

1. Double click the *Output* component in your window. *The Control Properties Dialog appears.*
2. Go to the **Events Tab**.
3. In the **Events combobox** > select the **OnDisplay event**.
4. Select the **Tutorial_HideShowControlBasedOnMode** Method from the General Methods tree.
5. Click on the **Link** button to link the Method to the OnDisplay event of this control.
6. Click **OK** to come out of the *Control Properties Dialog*.

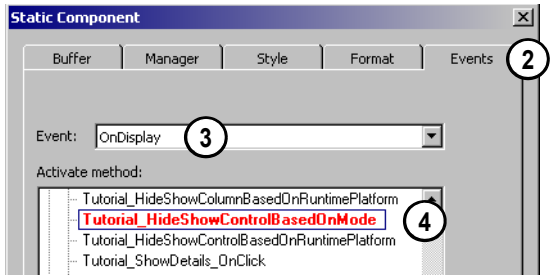
* *Note :* The runtime data flow checkbox of this component is checked by default.



Attach the Tutorial_HideShowControlBasedOnMode Method to the OnDisplay event of the Static component.



Check the Runtime data flow checkbox in the Manager Tab of the Static component.



Attach the Tutorial_HideShowControlBasedOnMode Method to the OnDisplay event of the Output component.

6. THE ADD AND UPDATE BUTTONS

In this step, you will drag the representations created for the Add and Update buttons onto the window. You will attach Methods to the OnDisplay event of both of these buttons, that will control which button is displayed based on the value of the VariableMode variable. OnDisplay Methods are executed on the server whenever the host program sends a window. In this way, you control which button is shown, and when. This is one alternative to driving interface variations from the host using Indicators and DDSs.

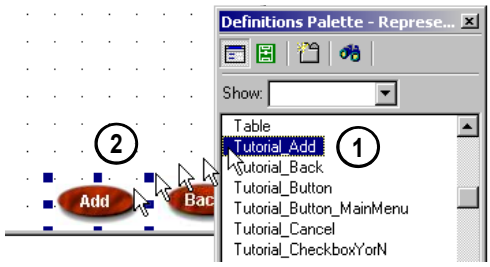
You will then attach a Method to the OnClick event of these buttons. This Method will perform server-side error handling when a button attached to it is clicked.

* *Note :* In the next exercise, you will see how error-handling can be performed host-side using the Jacada Studio IDK.

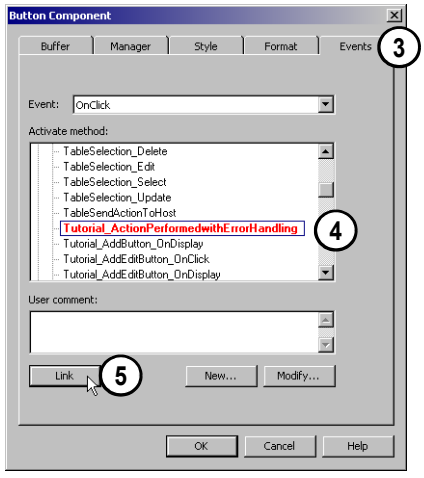
The Add Button

To add the 'Add' Button to your window, and control the functionality:

1. From ***Representations View*** on the **Definitions Palette** > Drag the **Tutorial_Add** representation onto your window > Place it to the left of the Back button.
2. Double-Click the Add Button.
   The Component Properties Dialog is invoked.
3. Go to the **Events** Tab of the Component Properties Dialog.
4. In the **Event** combobox > make sure the event is **OnClick**.
5. Scroll down the Activate Method list until you see the **Tutorial_ActionPerformedWithErrorHandling** Method.



Drag the Tutorial_Add representation onto your window and place it to the left of the Back button.

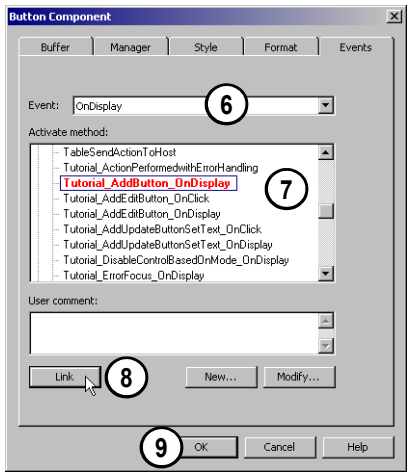


In the Events Tab of the Control properties Dialog, select the Tutorial_ActionPerformedWithErrorHandling Method and click the Link button.

6. **Select the Method** > Click the **Link** button.
7. In the **Event** combobox > choose the **OnDisplay** event.
8. Scroll down the Activate Method list until you see the **Tutorial_AddButton_OnDisplay** Method.

* *Note :* Double click this Method in the Events Tab of the Component Properties Dialog to view the code. This Method checks the value of the VariableMode variable (populated by FDMODE) to see whether the window was accessed in 'Add' or 'Edit' mode. If the window was accessed in 'Edit' mode, the button is hidden. If the window was accessed in 'Add' mode, the button is displayed.

9. **Select the Method** > Click the **Link** button.
10. Click **OK** to come out of the Component Properties Dialog.



In the Events combobox, choose the OnDisplay event and link the Tutorial_AddButton_OnDisplay Method to it.

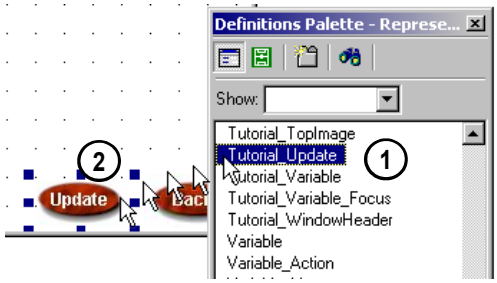
The Update Button

To add the 'Update' Button to your window, and control the functionality:

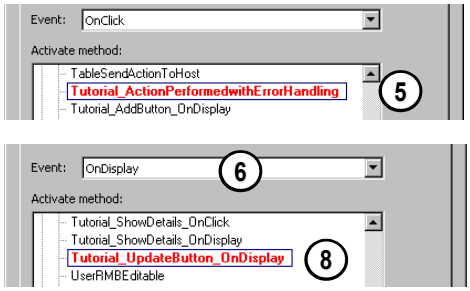
1. From ***Representations View*** on the **Definitions Palette** > Drag the **Tutorial_Update** representation onto your window > Place it on top of the 'Add' button.

* *Note :* Don't worry about aligning the 'Update' button precisely on top of the 'Add' button. We will align the buttons in a later step.

2. Double-Click the Update Button. *The Component Properties Dialog is invoked.*
3. Go to the **Events** Tab of the Component Properties Dialog.
4. In the **Event** combobox > make sure the event is **OnClick**.
5. Scroll down the Activate Method list until you see the **Tutorial_ActionPerformedWithErrorHandling** Method.
6. **Select the Method** > Click the **Link** button.
7. In the **Event** combobox > choose the **OnDisplay** event.
8. Scroll down the Activate Method list until you see the **Tutorial_UpdateButton_OnDisplay** Method.
9. **Select the Method** > Click the **Link** button.
10. Click **OK** to come out of the Component Properties Dialog.



Drag the Tutorial_Update representation onto your window and place on top of the 'Add' button.

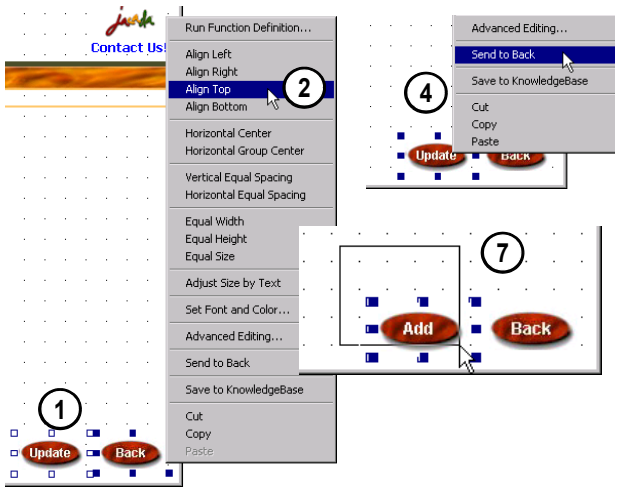


In the Events Tab, attach the Tutorial_ActionPerformedWithErrorHandling Method to the OnClick event and the Tutorial_UpdateButton_OnDisplay Method to the OnDisplay event.

Align the Buttons

Align the 'Update' and 'Add' buttons with the 'Back' button and with each other.

1. **Select** the 'Update' button > **Shift+Click** the 'Back' button to add it to the selection group. *Notice the 'Back' button has emphasized handles, making it the leading control.*
2. **Right-Click** anywhere on the window > Choose **Align Top** from the right mouse button menu.
3. **Click the window client** to deselect all controls.
4. **Select** the 'Update' button > **Right-Click** *to bring up the right mouse button menu* > Choose **Send to Back**. *The 'Add' button is now on top.*
5. **Select** the 'Add' button > **Shift+Click** the 'Back' button to add it to the selection group.
6. **Right-Click** anywhere on the window > Choose **Align Top** from the right mouse button menu.
7. **Click+Drag** a rectangle selection on the 'Add' and 'Update' Buttons. *Do not encompass the buttons fully or you may select the 'Back' button as well by mistake. It is enough that the selection window touches the two buttons for them to be selected. If the 'Update' button is not visible because it is completely covered by the 'Add' button...dont worry, it will be included in the selection.*
8. **Right-Click** anywhere on the window > Choose **Align Left** from the right mouse button menu. *The 'Add' and 'Update' buttons are now perfectly aligned with the back button and with each other.*

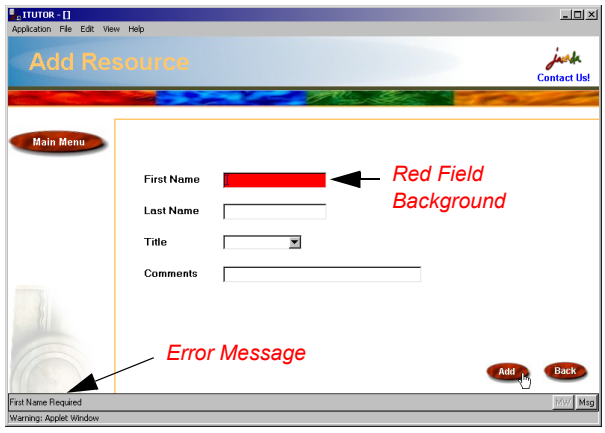


Align the 'Add' and 'Update' buttons with the 'Back' button and with each other

** Note : For more information on control manipulation, please refer to the IDK Walk-through section of this tutorial.*

7. ERROR HANDLING

This exercise exhibits error handling with the use on server-side code and not any host side code. The advantage of server-side code is to validate the correct data is being accepted prior to passing this data to the host data bases. This can help to improve performance by not sending data to the host and validating it there and then sending it back to the presentation for user correction. The example of a server-side validation, in this case, is done through the use of a method called Tutorial_ActionPerformedwithErrorHandling which is linked to both the "Add" and "Update" buttons. The "Add" is invoked when using the Add Resource window and the "Update" is used when in the Edit Resource Window. Each of these buttons is the only button on the respective window which sends information to either add or modify the data bases.



On error - the fields background is set to red and a message is displayed in the DIL.

The method is written to verify that both the FDFNAM (First Name field), FDLNAM (Last Name field), and FDTITL (Title) fields do not contain all blanks. Since these are required fields, each field is checked to make sure some text entry has been type before submitting this data. If one of these fields contains all blanks, an error message is presented after a refresh of the window is done. The refresh allows an OnDisplay method Tutorial_ErrorFocus_OnDisplay linked to each of these field to activate and change the associated representation background color to turn red and sets the cursor position to the field with all blanks.

```
If: Cond: `#1 == _TRUE`
  If: Cond: `#2 == VariableFocus`
    DoMethod: Receiver: `this` Method: SetFocus Parms: ( )
    DoMethod: Receiver: `this` Method: SetBackgroundColor Parms: ( `"Red"` )
  Else:
    DoMethod: Receiver: `this` Method: RestoreDefaultColors Parms: ( )
  EndIf:
EndIf:
```

IDK method lines that set background to red.

Example of the error messages sent:

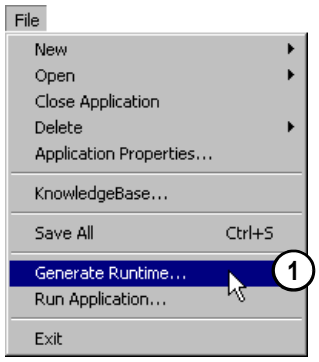
Field	Error Message
First Name	"First Name Required"
Last Name	"Last Name Required"
Title	"Title Required"

8. GENERATE RUNTIME AND TRANSFER FILES

Compile the Subapplications into an executable, and transfer the DDS physical files to the host via the Generate Runtime Wizard. The automatically generated Shell programs will not be transferred for this subapplication, since the host code has been prepackaged for you and already exists in the MYTUTORIAL host library.

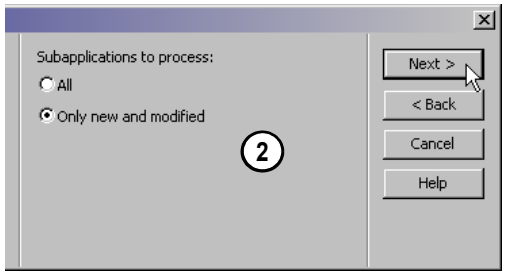
To generate a Runtime:

1. From the **File** menu > choose **Generate Runtime...** *The Generate Runtime Wizard is invoked.*
2. In the **Generate Runtime Wizard**, click **Next** to accept the following default settings:



Generate a Runtime in order to compile your SubApplication into an executable

Runtime Type:	**Java**
Jacada Server Platforms:	Windows NT(2000) x86
Subapplications to Include:	All
Subapplications to Process:	**Only new and modified**

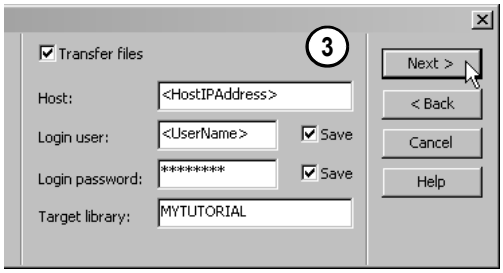


Select to process only new and modified subapplications.

* *Note :* The **Only new and modified** option allows you to only compile the Subapplications that were modified. Only the modified DDS physical file and RPG program of subapplication PADDRE are FTP'd back to the host, and only they need to be compiled.

3. In the **File Transfer** screen, specify the following information, then click **Next**:

Transfer Files:	<checked>
Host:	<YourHostIPAddress>
Login User:	<YourUserNameOnHost>
Login Password:	<YourPasswordOnHost>
Target Library:	**MYTUTORIAL**

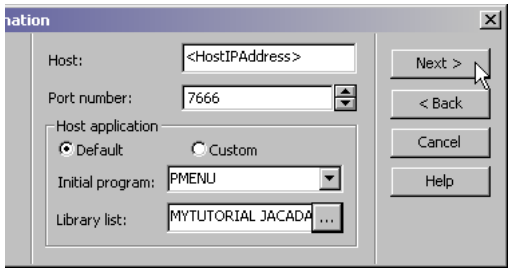


Fill out the File Transfer screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, specify your respective Library (i.e. TUTORIAL01) as the Target Library.

4. In the **Specify Host Connection and Application Information** screen, specify the following information, then Click **Next**:

Host:	<YourHostIPAddress>
Port Number:	7666
Initial Program:	PMENU
Library List:	**MYTUTORIAL JACADA**



Fill out the Host Connection and Application Information screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, remember to have your respective Library (i.e. TUTORIAL01) be the first library in the Library List entry and include the JACADA library after your library.

5. Click **Finish** to complete the Generate Runtime Wizard, and commence with the compilation process.
The Generating the Runtime dialog appears.

   Look for the message "Runtime was successfully generated" at the end of the compilation process, this confirms that the compile was successful and that you can continue to the next step.

6. Click **Close** to exit the *Generating the Runtime* dialog.

9. ABOUT HOST CODE

Let's look at the block of code to the right, taken from the program logic written for the PADDRE Subapplication of the ITUTOR demo.

This example shows how to read a particular record requested by the user, and how to display the information from the record on the window. In this example, database fields are moved to the window buffer fields. This process is similar to the one you use today, when working with display files.

```
C*********************************************
C* Load Data for Edit Mode
C*********************************************
C           EDTMOD    BEGSR
C           RES#      CHAINRESR                81
C           *IN81     IFEQ '0'
C                     MOVE RERESN    FDRESN
C                     MOVE RELNAM    FDLNAM
C                     MOVE REFNAM    FDFNAM
C                     MOVELRERCOM    FDRCOM
C                     MOVELRETITL    FDTITL
C                     ENDIF
C                     ENDSR
```

Sample Code taken from the program logic written for the PADDRE Subapplication of the ITUTORIAL application.

10. COMPILE DDS AND PROGRAM FILE ON HOST

Verify that the JACADA library is included in your library list.

Compile the DDS physical file, **PADDRE$D** and the RPG program file **PADDRE** in your **MYTUTORIAL** library.

11. RUN APPLICATION

In order to run your Application, you must first verify that the Jacada monitor is active. Only then can you run your Application.

Ensure the Jacada Monitor is Active

To ensure that the Jacada Monitor is active:

1. Type **CFGJACMON** in the iSeries command line.
2. In the Configure Jacada Monitors screen, make sure that the word '**Active**' Appears in the *Status* column to the right of the Jacada Monitor.

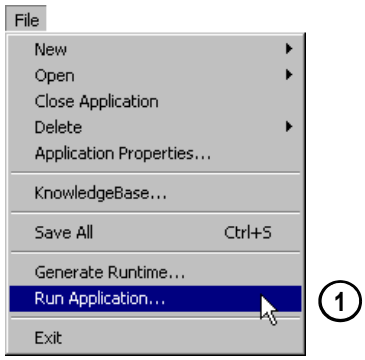
If the Jacada monitor is not active:

Type **1** in the *Opt* column to start the Jacada Monitor.

Run your Application with a JAVA Client

Run the executable created during the Runtime Generation process from within the development environment:

1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*



Run your application through the development environment by choosing File > Run Application

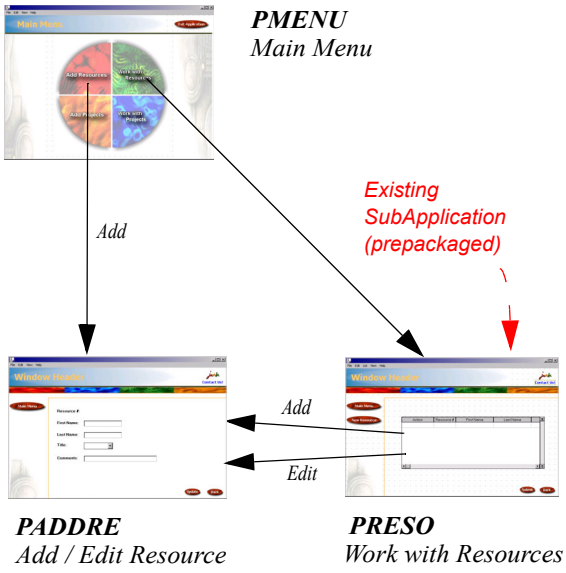
2. In the **Run Application Wizard**, agree to the default Runtime properties, by clicking the **Next** button, when prompted for:

Runtime Type:	**Java**
Web Server:	Integrated HTTP Service
Application URL:	http://localhost:8080/MYTUTOR.html

3. Click **Finish** to come out of the Run Application Wizard.
   The Jacada Server is activated and your Default Browser window is opened to the Jacada <ApplName>.html page.
4. Type your iSeries **Username** and **Password** into the appropriate fields.
5. Click the **OK** button to run your JAVA client Application.

Navigate to the Add / Edit Resource Window

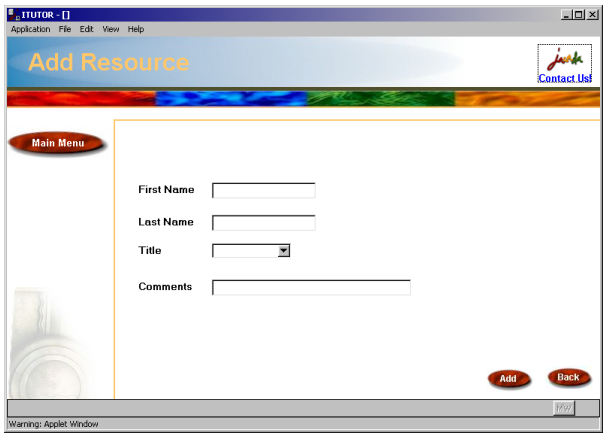
Navigate to the PADDRE Subapplication by using the diagram provided for you to the right. Navigate to the Subapplication in both Add and Edit mode to see the final product of your efforts in this exercise.



Navigate to the PADDRE SubApplication in both Add and Edit mode.

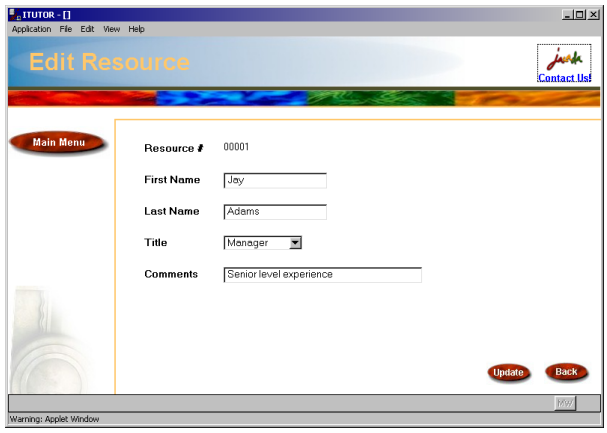
The Finished Product

This is what your window should look like when accessed in 'Add' mode in the Java client.



The final product of your efforts in this exercise. The look of the PADDRE SubApplication, when accessed through 'Add' mode.

This is what your window should look like when accessed in 'Edit' mode in the Java client.

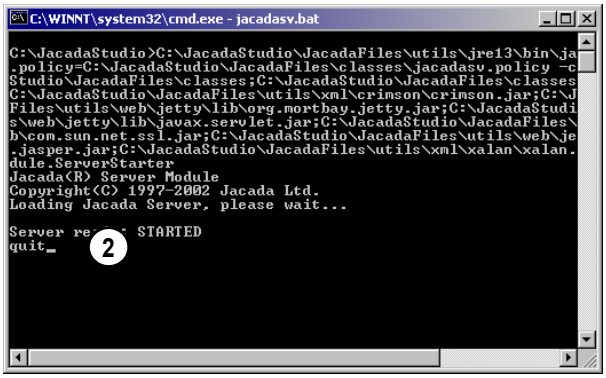


The final product of your efforts in this exercise. The look of the PADDRE SubApplication, when accessed through 'Edit' mode.

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by navigating back to the Main Menu window with the 'Back' button and clicking the 'Exit Application' button in the Main Menu window.
2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes if a message appears.
3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.
4. Close your browser window.



Type 'quit' in the Jacada Server command window or press Ctrl+C to close the Jacada Server

Exercise 3 - Jacada Studio For iSeries

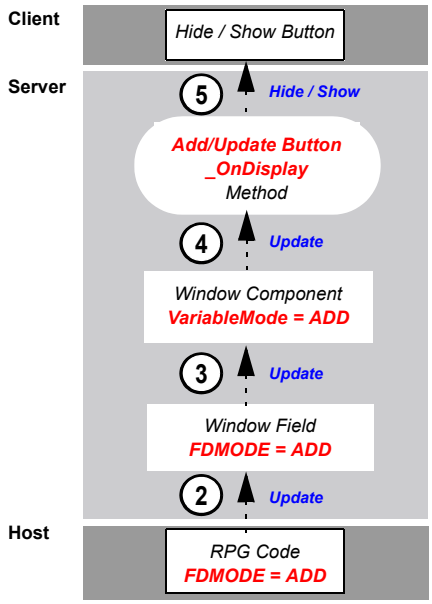
How It Works: The Add / Edit Resource Window

If you feel comfortable with the level of detail provided in this exercise, feel free to skip this section and go on to the next exercise. If you'd like to find out a bit more about how it all works, keep reading. In this section you learn about how everything comes together during runtime. Are you ready?

The VariableMode Hidden Variable

Remember the Methods that you attached to the OnDisplay events of the 'Add' and 'Update' buttons? These Methods are used to control whether the button will be displayed or not during runtime. To see how this works, open the PADDRE program on the host and take a look. Heres how it works:

1. The RPG program receives the **MODE** parameter from the program from which it was called. The value of the MODE parameter is either *ADD* or *EDIT* depending on the program from which this screen was called. The RPG program moves the value of the MODE field to the **FDMODE** field.
2. When the RPG program gets to a READ statement, the screen is sent to the Jacada Server with the value of the **FDMODE** field.
3. On the server, the value of the FDMODE field is passed to the **VariableMode** variable.
4. On the server, all OnDisplay Methods are executed when the new screen loads or is refreshed.
5. **Add/UpdateButton_OnDisplay** Method extracts value of VariableMode and decides whether to display button depending on that value.



Exercise 4 - Jacada Studio for iSeries

Create the Add / Edit Project Window

Objectives:

- To provide an exercise that replicates the Add/Edit Project window in the pre-packaged iTutor application
- To build a window in which new data can be Added or existing data can be retrieved and Updated
- To examine dynamic GUI display alternatives to Indicator driven Display File behaviors
  - Conditional runtime display driven by client events (based on program mode of Add or Update)
  - Project # show/hide depending on mode
  - Variable Text and behavior (Update or Add) of a single Action button displayed depending on mode
  - Disabling an input capable field at the client based on program mode
- Using methods to communicate the results of data validation performed at the host to the client
- Using code extensions to add advanced GUI controls in XHTML (date control)

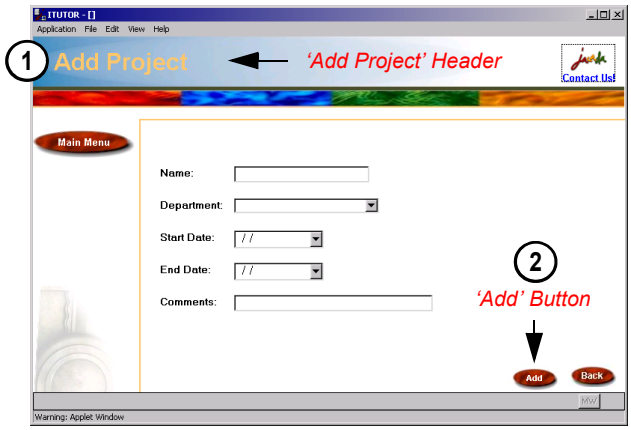
In this exercise, you create the Add / Edit Projects window of the ITUTOR Application. Your MYTUTOR Application should already contain six of the eight Subapplications in the ITUTOR demo, four that were prepackaged for you and the Main Menu and Add/Edit Resource windows that you created in the previous exercises. This window is used both as an Add Project window and as an Edit Project window, depending on the entry point chosen by the user. In the last exercise, you attached a Method that performed server-side error handling. In this exercise, we will show you one way of performing error handling through your RPG program's business logic. In the last exercise, you attached a Method that performed a hide / show on the 'Add' and 'Update' buttons, depending on the mode that you were in. In this exercise, we will show you another way of achieving the same results, server-side. Last but not least, you'll run an XHTML runtime, and extend your HTML code to achieve the desired results. In this exercise, you will experience how easily and speedily a window can be constructed and deployed, with a predefined KnowledgeBase.

You are here!

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

The major steps to this exercise are:

1. **Window Design Specifications**
2. **Create the PADDPR Subapplication**
3. **The Add and Update Button**
4. **Adding Fields to the Window**
5. **Error Handling On The Host**
6. **The "Project #" Representation**
7. **Generate Runtime and Transfer Files**
8. **Compile DDS and Program File on Host**
9. **Run Application with a Java Client**
10. **Extend the HTML Code to Include Date Controls**
11. **Run Application with an XHTML Client**



The final product of your efforts in this exercise. The look of the PADDPR Subapplication, when accessed through 'Add' mode.

1. WINDOW DESIGN SPECIFICATIONS

First, you must define the differences in the Subapplication interface, between when it is accessed through 'Add' mode and when it is accessed through 'Edit' mode. The differences are:

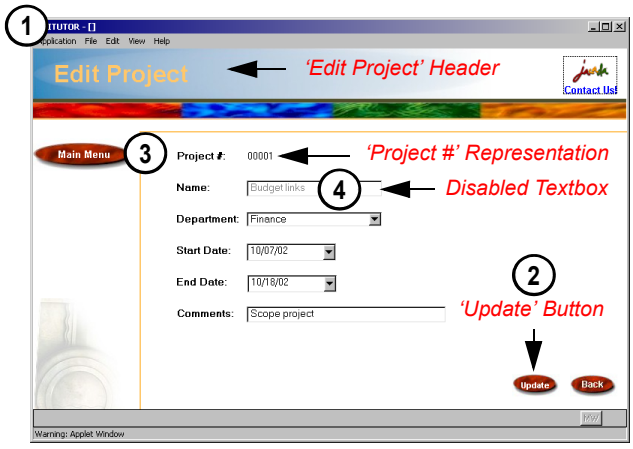
In "Add" mode:

1. The window header says "Add Project".
2. There is an "Add" Button, to the left of the "Back" button in the bottom right corner of the window.

In 'Edit' mode:

1. The window header says "Edit Project"
2. There is an 'Update' Button, to the left of the 'Back' button in the bottom right corner of the window.
3. There is a "Project #" representation.
4. The 'Name' textbox is disabled.

In this Subapplication, you must also take into account the requirement to support both the Java and XHTML runtimes with date controls. HTML support of date controls varies from Java today. Therefore, you will extend your HTML code with a prepackaged extension. This extension will allow you to implement the date control in the HTML runtime and provide you an example of the extension architecture.

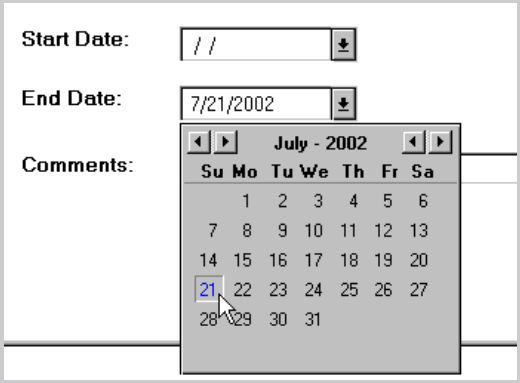


The final product of your efforts in this exercise. The look of the PADDPR SubApplication, when accessed through 'Edit' mode.

FYI: The IDK Date Control

The IDK date control is a specialized edit field that enables your users to easily enter data that is intended to be interpreted as date information. The date control contains the following features:

1. *Format masking that limits the allowed input characters.*
2. *Year data input as two digits is automatically translated to four digit data using the defined base year.*
3. *A calendar tool associated with the date control allows users to choose a date graphically.*



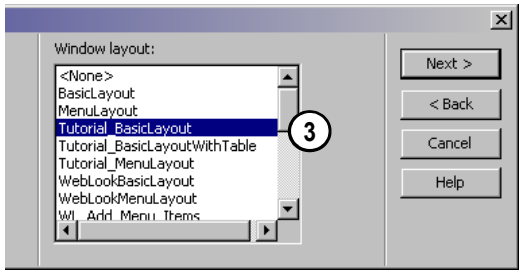
2. CREATE THE PADDPR SUBAPPLICATION

In this step, you create the PADDPR Subapplication in your MYTUTOR Application. Make sure that the Application combobox shows that you are in the MYTUTOR Application before proceeding. In this exercise, you apply a prebuilt Window Layout called *Tutorial_BasicLayout* to your Subapplication.

1. Open the Jacada Studio Interface Development Kit (IDK) and the MYTUTOR Application
2. From the **Subapplication Menu** > **New ...** *The New Subapplication Wizard is invoked.*
3. In the **New Subapplication Wizard**, specify the following Subapplication properties:

Subapplication Name:	**PADDPR**
Popup Window:	Unchecked
Window Layout:	**Tutorial_BasicLayout**
Subapplication Description:	<none>

4. Click **Finish** to come out of the New Subapplication Wizard.



In the Select Window Layout step of the New Subapplication Wizard, choose the Tutorial_BasicLayout Window Layout.

3. THE ADD AND UPDATE BUTTON

In the last exercise, you used two buttons and controlled the display with a method that controlled which button is displayed based on the value of the VariableMode variable.

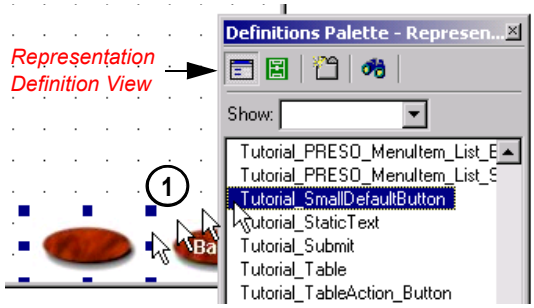
In this exercise, you use only one button. You will attach '"SetText" Methods to the OnDisplay and OnClick events of this button. The method will change the text that appears on the button, based on the value of the VariableMode variable. This is another example of controlling interface variations from the host, similar to using Indicators and DDSs.

To add a button to your window, and control its functionality:

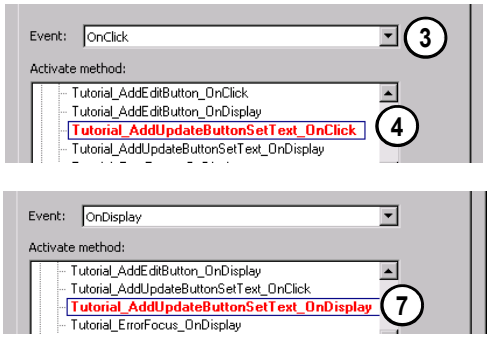
1. From *Representation Definition View* of the **Definitions Palette** > Drag the **Tutorial_SmallDefaultButton** representation onto your window > Place it to the left of the Back button.
2. Double-Click the button. *The Component Properties Dialog is invoked.*
3. Go to the **Events** Tab of the Component Properties Dialog. *Make sure that the Event is OnClick.*
4. Open the General Methods tree and scroll down the Activate Method list until you see the **Tutorial_AddUpdateButton SetText_OnClick** Method.

* *Note :* For a detailed explanation of ***Events*** - see the 'How It Works' section at the end of the Main Menu window exercise.

5. **Select the Method** > Click the **Link** button.
6. In the **Event** combobox > choose the **OnDisplay** event.
7. Scroll down the Activate Method list until you see the **Tutorial_AddUpdateButtonSetText_OnDisplay** Method.
8. **Select the Method** > Click the **Link** button.

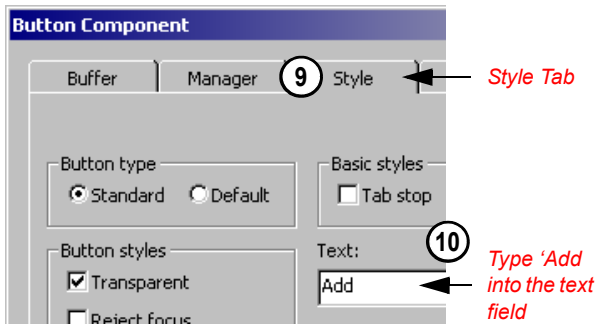


Drag the Tutorial_SmallDefaultButton representation onto your window and place it to the left of the Back button.



In the Events Tab, attach the Tutorial_AddUpdateButtonSetText_OnClick Method to the OnClick event and the Tutorial_AddUpdateButtonSetText_OnDisplay Method to the OnDisplay event.

9. Go to the **Style** Tab of the Component Properties Dialog.
10. Type **'Add"** into the Text Field.
11. Click **OK** to come out of the Component Properties Dialog.



Go to the Style Tab and type 'Add' into the text field.

4. ADDING FIELDS TO THE WINDOW

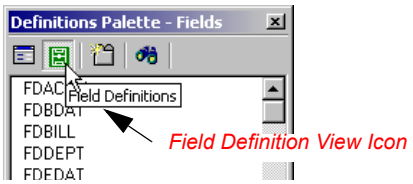
In this step, you will add Representation Definitions to the window by dragging fields onto your window from ***Field Definitions View*** of the *Definitions Palette*. You will then choose the representations associated with the field from a "Short List".

Click the *Field Definitions View Icon* in the Definitions Palette to see the list of fields defined in the KnowledgeBase.

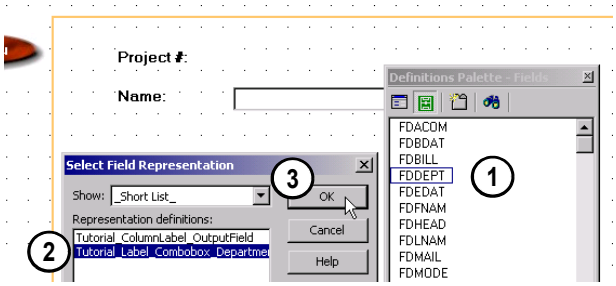
To add the remaining Representations to your Subapplication, refer to the table below:

Window Fields	Representations
FDPNUM	Tutorial_Label_OutputField
FDPNAM	Tutorial_Label_InputField
FDDEPT	Tutorial_Label_Combobox_Department
FDBDAT	Tutorial_Label_DateControl
FDEDAT	Tutorial_Label_DateControl
FDPCOM	Tutorial_Label_InputField

1. **Drag the fields** in the *Window Fields* column onto the window, from the Definitions Palette. *The Select Field Representation Dialog appears.*
2. **Select the representation** in the *Representations* column, from the _Short List_ in the Select Field Representation Dialog.
3. **Click OK** to come out of the dialog. *Your representation is added to the window. It is associated with the field that you dragged from the Definitions Palette.*



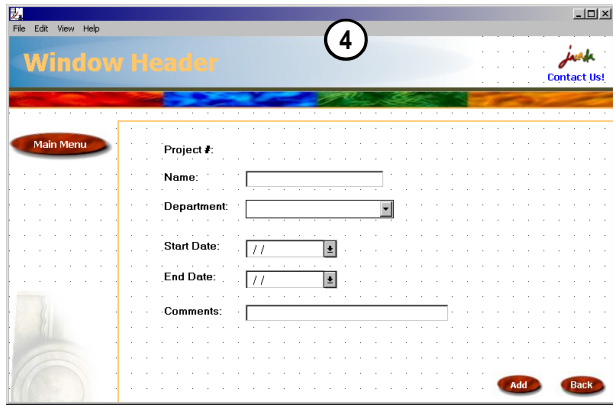
Click on the Field Definitions View Icon in the Definitions Palette to see the list of fields defined in the KnowledgeBase.



Drag fields onto the window, then choose the correct representation from the Short List.

Arrange the Position of the Added Representations

Arrange the position of the added representations according to the image to your right. *For more information on control manipulation, please refer to the IDK Walk-through section of this tutorial.*



Arrange the position of the added representations like this.

5. ERROR HANDLING ON THE HOST

Before we see how error handling is performed on the host, we need to add a couple of variable definitions to our window.

Add Variable Representations to the Window

In the last exercise, we saw and implemented the usage of the VariableMode variable (attached to FDMODE field). In this exercise, we use the **VariableFocus** variable **(attached to JSTFOC field)** and the **VariableMessage** variable **(attached to JSTMSG field)** to perform error handling on the host.

* *Note :* The VariableMessage variable and JSTMSG field already exist in your Subapplication - they were brought in by your window layout

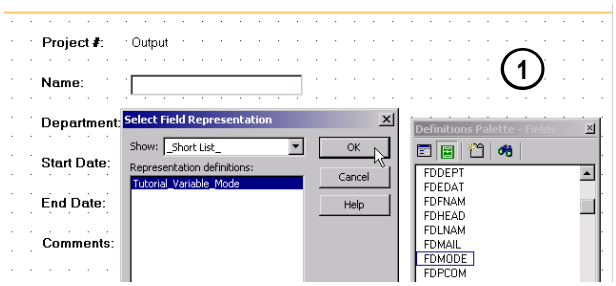
Don't forget, we still need to add the FDMODE variable to our Subapplication. It is used by the SetText method to set the value of the 'Add 'and 'Update' buttons.

1. Add the ***Variable*** Representation Definitions in the table below to the window by dragging ***Window Fields*** onto your window. Then, choose the ***Representations*** associated with the field from the "Short List" of favorites. Drag the following fields onto your Subapplication window from ***Field Definitions View*** of the Definitions Palette:

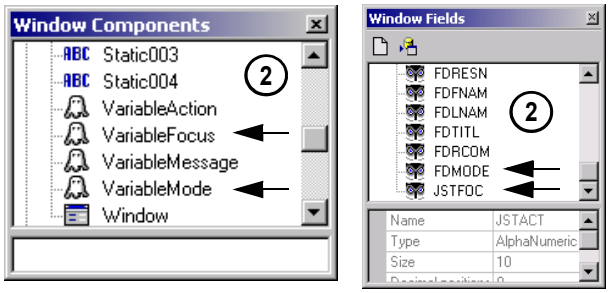
Window Fields	Representations
JSTFOC	Tutorial_Variable_Focus
FDMODE	Tutorial_Variable_Mode

* *Note :* When you drag these fields onto the window, it will seem as if nothing has happened, and no representation will appear. This is because these fields are attached to ***hidden*** variable representations.

2. Make sure that these fields were added to the window: Check that the fields **JSTFOC** and **FDMODE** reside in the list of fields in the Window Fields Palette. Check that the ***VariableFocus*** and ***VariableMode*** variables exist in the list of window components in the Window Components Palette. *(they will have an icon with a picture of a ghost)*



Drag fields onto the window, then choose the correct representation from the Short List.



Check the contents of the Window Components and Window Fields palettes to ensure the variable representations have been added to your Subapplication.

About Host Code Error Handling

In the last exercise, error handling was performed server-side. In this exercise, let's look at an example of how error handling can be performed on the host. To see how we did it, open the **PADDPR** program on the host and take a look. Heres how it works:

Let's look at the block of code to the right, taken from the program logic written for the PADDPR Subapplication of the ITUTOR demo. This example shows how to display a message, and how to set the focus on a given field.

In this example, "Project Added" was moved to the message field in the buffer. *(You can see MSG2 defined on page 1 of the program listing.)* The Method **Tutorial_ErrorFocus_OnDisplay**, which executes at display time, will display the message in your client's Dynamic Information Line (DIL) during runtime. The preceding was done if the project was added successfully.

Otherwise, if the project was not added successfully:

The field name FDBDAT *(connected to StartDate on the screen)* was moved to the field **JSTFOC**, when the window is displayed; and MSG4 was moved to the DIL *(You can see MSG4 defined on page 1 of the program listing.)*.

* *Note :* See the 'How It Works' section at the end of the exercises for a detailed explanation of the runtime behavior between client, server and host.

```
C*****************************************************
C*  Add Record - User Clicked Add Button             *
C*****************************************************
C*
C* Send "Project Added" to DIL
C                     MOVELMSG2       JSTMSG
C*
C*Error - Start Date > End Date
C                     ELSE
C                     MOVEL'FDBDAT'   JSTFOC
C                     MOVELMSG4       JSTMSG
C                     ENDIF
C*
```

Sample Code taken from the program logic written for the PADDPR Subapplication of the ITUTOR application.

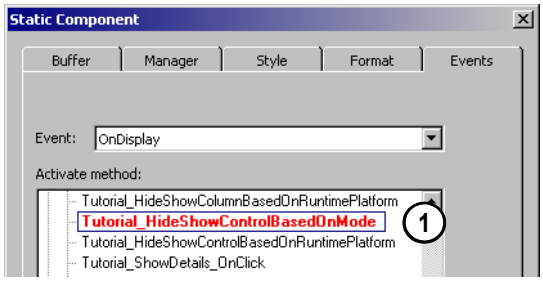
6. THE "PROJECT #" REPRESENTATION

According to our project requirements, the Project Number controls must appear on the window if it is accessed in 'Edit' mode, but not if it is accessed in 'Add' mode. As you did in the previous exercise, use the ***Tutorial_HideShowControlBasedOnMode*** hide/show Method to control this interface variation. To do this:

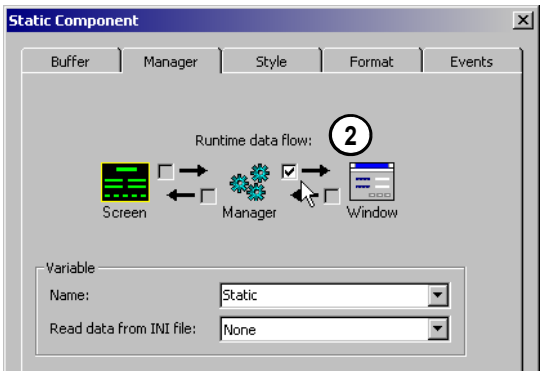
1. Attach the **Tutorial_HideShowControlBasedOnMode** Method to the OnDisplay event of both the Static and the Output components.

* *Note :* Make sure you attach the method to **both** the Static and Output components. For further details, see *Step 5 of Exercise 3* to refresh your memory on how this is done.

2. Check the **Runtime data flow** checkbox in the *Manager Tab* of the Static component's *Properties Dialog*.



Attach the Tutorial_HideShowControlBasedOnMode Method to the OnDisplay event of both the Static and the Output components.



Check the Runtime data flow checkbox in the Manager Tab of the Static component.

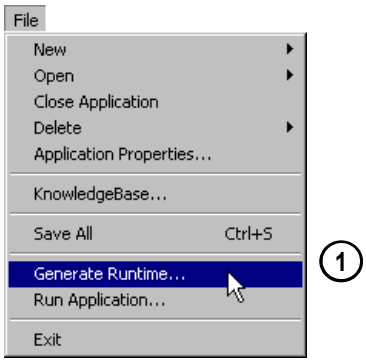
7. GENERATE RUNTIME AND TRANSFER FILES

Compile the Subapplications into an executable, and transfer the DDS physical files to the host via the Generate Runtime Wizard. The Shell programs will not be transferred, since the host code has been prepackaged for you and already exists in the MYTUTORIAL host library. To generate a Runtime:

1. From the **File** menu > choose **Generate Runtime...** *The Generate Runtime Wizard is invoked.*

2. In the **Generate Runtime Wizard**, click **Next** to accept the following default settings:

Runtime Type:	**Java** and **XHTML**
Jacada Server Platforms:	Windows NT(2000) x86
Subapplications to Include:	All
Subapplications to Process:	**Only new and modified**

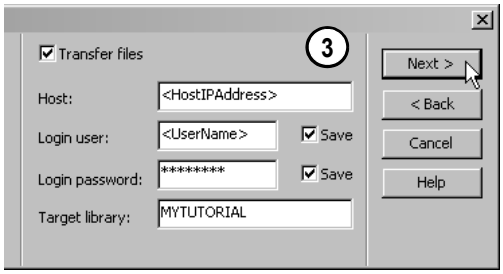


Generate a Runtime in order to compile your SubApplication into an executable

* *Note :* The **Only new and modified** option allows you to only compile the Subapplications that were modified. Only the modified DDS physical file and RPG program of subapplication PADDPR are FTP'd back to the host, and only they need to be compiled.

3. In the **File Transfer** screen, specify the following information, then click **Next**:

Transfer files:	<Checked>
Host:	<YourHostIPAddress>
Login User:	<YourUserNameOnHost>
Login Password:	<YourPasswordOnHost>
Target Library:	**MYTUTORIAL**

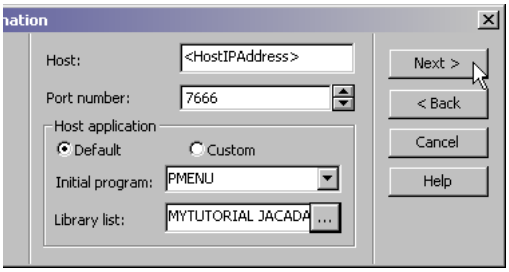


Fill out the File Transfer screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, specify your respective Library (i.e. TUTORIAL01) as the Target Library.

4. In the **Specify Host Connection and Application Information** screen, specify the following information, then Click **Next**:

Host:	<YourHostIPAddress>
Port Number:	7666
Initial Program:	PMENU
Library List:	**MYTUTORIAL JACADA**



Fill out the Host Connection and Application Information screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, remember to have your respective Library (i.e. TUTORIAL01) be the first library in the Library List entry and include the JACADA library after your library.

5. Click **Finish** to come out of the Generate Runtime Wizard, and commence with the compilation process. *The Generating the Runtime dialog appears.*

   Look for the message "Runtime was successfully generated" at the end of the compilation process, this is an indication that all is well and you can safely go on to the next step.

6. Click **Close** to come out of the Generating the Runtime dialog.

8. COMPILE DDS AND PROGRAM FILE ON HOST

Verify that the Jacada Library is included in your library list.

Compile the DDS physical file, **PADDPR$D** and the RPG program file **PADDPR** in your **MYTUTORIAL** library.

9. RUN APPLICATION WITH A JAVA CLIENT

In order to run your Application, you must first verify that the Jacada monitor is active.

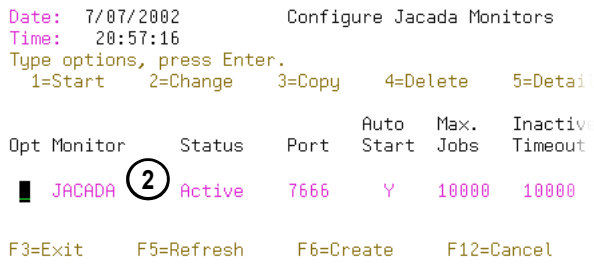
Ensure the Jacada Monitor is Active

To ensure that the Jacada Monitor is active:

1. Type **CFGJACMON** in the iSeries command line.
2. In the Configure Jacada Monitors screen, make sure that the word '**Active**' Appears in the *Status* column to the right of the Jacada Monitor.

If the Jacada monitor is not active:

Type **1** in the *Opt* column to start the Jacada Monitor.



In the Configure Jacada Monitors screen, make sure that the word 'Active' Appears in the Status column to the right of the Jacada Monitor

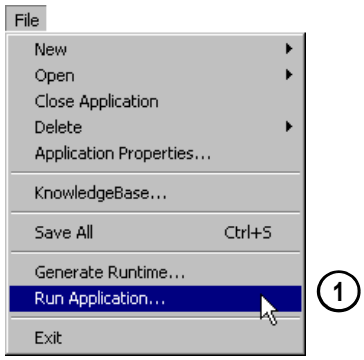
Run your Application with a JAVA Client

Run the executable created during the Runtime Generation process from within the development environment:

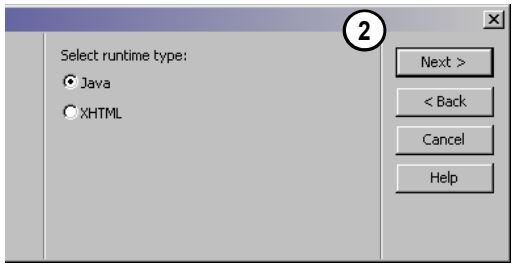
1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*
2. In the **Run Application Wizard**, agree to the default Runtime properties, by clicking the **Next** button, when prompted for:

Runtime Type:	Java
Web Server:	Integrated HTTP Service
Application URL:	http://localhost:8080/ MYTUTOR.html

3. Click **Finish** to come out of the Run Application Wizard. *The Jacada Server is activated and your Default Browser window is opened to the Jacada <ApplName>.html page.*
4. Type your iSeries **Username** and **Password** into the appropriate fields.
5. Click the **OK** button to run your Java client Application.



Run your application through the development environment by choosing File > Run Application



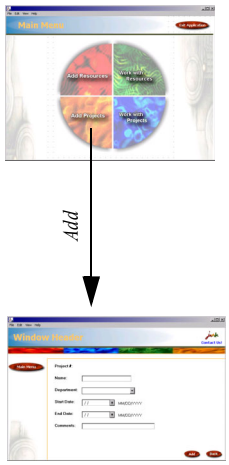
Choose Java as your runtime type in the Run Application Wizard

Navigate to the Add / Edit Projects Window

Navigate to the PADDPR Subapplication by using the diagram provided for you to the right.

* *Note :* You can only navigate to this Subapplication in 'Add' mode at this time. This is because the PPROJ Subapplication, from which you can access this window in 'Edit' mode, has not been built yet. You will build the PPROJ window in the next exercise. For now, navigate to this window in 'Add' mode only. After you complete the next exercise, navigate to this window in 'Edit' mode to see the final product of your efforts in this exercise.

PMENU
Main Menu

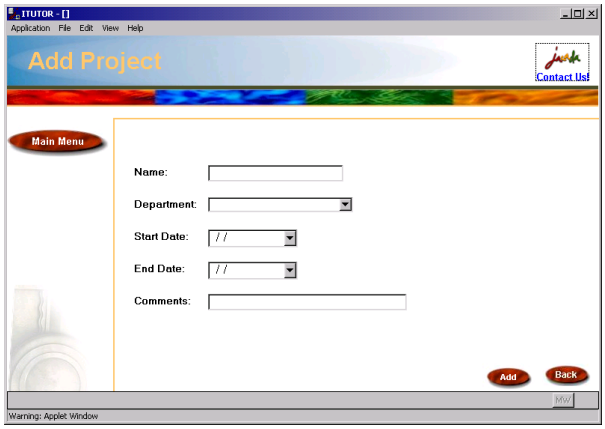


PADDPR
Add Project

Navigate to the PADDPR SubApplication Add mode. You will be able to access the window in Edit mode after completing Exercise 5.

The Finished Product

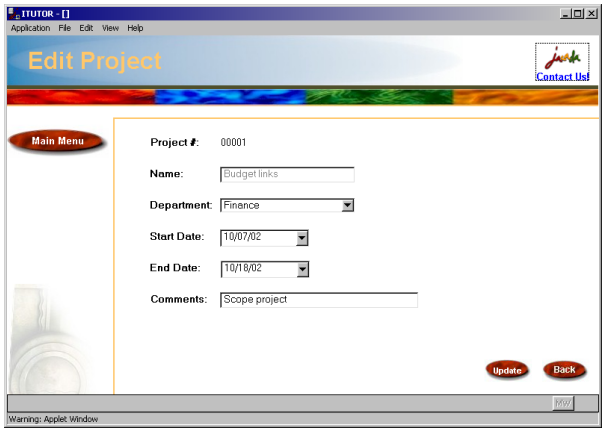
This is what your window should look like when accessed in 'Add' mode in the Java client.



The final product of your efforts in this exercise. The look of the PADDPR SubApplication, when accessed through 'Add' mode in a Java runtime.

This is what your windows should look like when accessed in 'Edit' mode in the Java client.

* *Note :* Remember, you will only be able to access the window in Edit mode after completing Exercise 5.



The final product of your efforts in this exercise. The look of the PADDPR SubApplication, when accessed through 'Edit' mode in a Java runtime.

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by navigating back to the Main Menu window with the 'Back' button and clicking the 'Exit Application' button in the Main Menu window.
2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes if a message appears.
3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.
4. Close your browser window.

10. EXTEND THE HTML CODE TO INCLUDE DATE CONTROLS

In the XHTML client runtime, it is still possible to improve your Application's look and feel after developing your Application in the IDK. Such improvements are made outside of the IDK and are manually incorporated into the Jacada Studio Application. This is done by creating user HTML extensions. During runtime, the Jacada Server merges these HTML extensions with your runtime Application. User HTML extensions enable you to incorporate Java Scripts, VB Scripts and various other HTML features into runtime-generated XHTMLs.

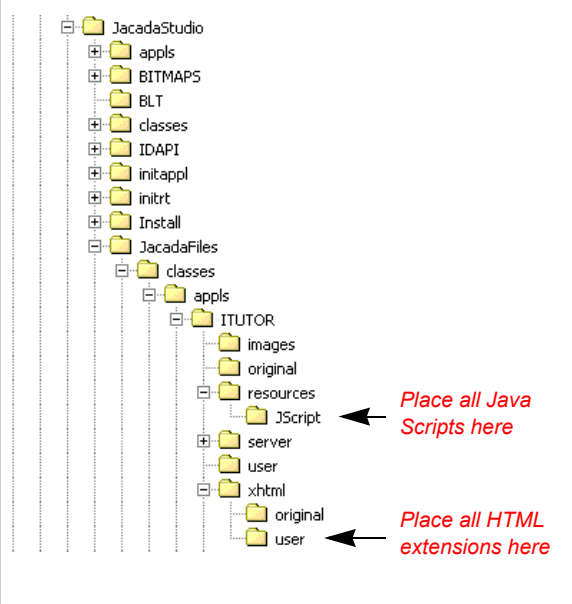
For the sake of this example, the requirement is that both the Java and XHTML runtimes have date controls. HTML does not support date controls, so we have extended the HTML code with a prepackaged extension. This extension will allow you to implement the date control globally in the HTML runtime.

FYI: Naming and Placing a User HTML Extension File

It is important to correctly name and position your user HTML extension file. The file's name has a direct bearing on the level at which the extension file is incorporated into the Application. Extension files can exist on different levels: Subapplication level extensions take the highest priority and override Application level extensions.

***Application Level** - To merge a user extension file with the whole Application, you must give it the name "appl.html" and save it in the **\JacadaFiles\classes\appls\<applname>\xhtml\user** directory. The extension file is then incorporated into the whole Application. It effects all Subapplications in the Application.*

***Subapplication Level** - To merge a user extension file with one specific Subapplication, you must give it the name of the Subapplication. If the Subapplication's name is "PADDPR", then you must name your extension file "**PADDPR.html**" and save it in the Application or library's **user** directory. The extension file is then incorporated only into that one screen.*



The Date Control Extension

The javascript file **APS.js** is used to create the date controls and the GUI calendar functionality. It is called from the HTML file **appl.html**. This file is used as a global extension and it effects all Subapplications in the Application.

These extension files are included with Jacada Studio for iSeries, and can be used to create sophisticated GUI date controls in all of your HTML client runtimes.

To view the prepackaged extension:

1. Go to the directory **\JacadaStudio\JacadaFiles\classes \appls\MYTUTOR\xhtml\user**
2. Open the **appl.html** file in any text editor

To view the prepackaged javascript:

1. Go to the directory **\JacadaStudio\JacadaFiles\classes \appls\MYTUTOR\resources\JScript**
2. Open the **APS.js** file in any text editor to view the code.

* *Note :* In addition to adding the date controls and the GUI calendar, the APS.js javascript file also sizes the windows and removes the up/down scrolling buttons assigned to tables by default.

```
<html>
<head>
<script src="http://localhost:8080/classes/appls/
ITUTOR/resources/JScript/APS.js">
</script>
<title></title>
</head>
<body onLoad="onLoad_();">
<form name="jacadaform" id="jacadaform">
</form>
<form name="myForm">
<script>
  checkDateControls();
</script>
</form>
</body>
</html
```

Contents of the appl.html global extension used in the HTML runtime. Extension calls the APS.js javascript

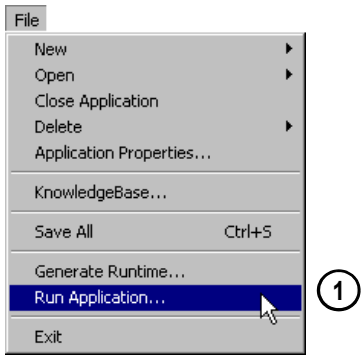
11. RUN APPLICATION WITH AN XHTML CLIENT

Run the executable created during the Runtime Generation process from within the development environment:

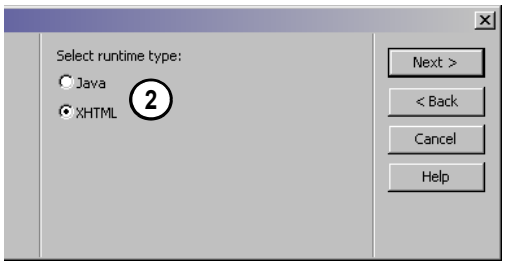
1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*
2. In the **Run Application Wizard**, agree to the default Runtime properties by clicking the **Next** button, when prompted.

Runtime Type:	**XHTML**
Port Number:	8080
Web Server:	Integrated HTTP Service
Application URL:	http://localhost:8080/ MYTUTOR-xhtml.html

3. Click **Finish** to come out of the Run Application Wizard. *The Jacada Server is activated and your Default Browser window is opened to the Jacada <ApplName>.html page.*



Run your application with an XHTML client by choosing File > Run Application in the development environment.



Choose XHTML as your runtime type in the Run Application Wizard

4. Type your iSeries **Username** and **Password** into the appropriate fields.

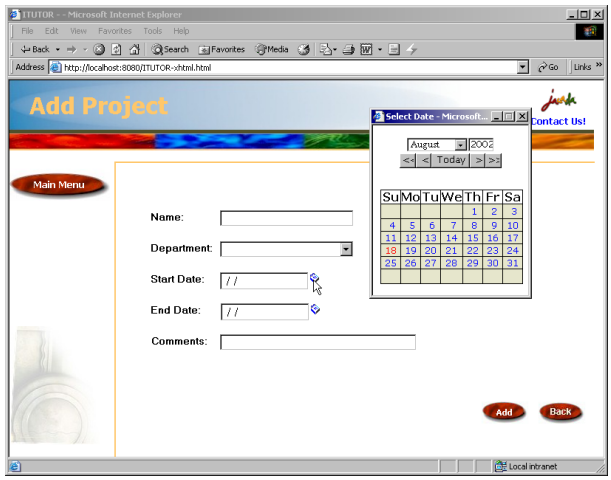
5. Click the **OK** button to run your HTML client Application.

Navigate to the Add / Edit Projects Window

Navigate to the PADDPR Subapplication. Navigate to the Subapplication in both Add and Edit mode to see the final product of your efforts in this exercise.

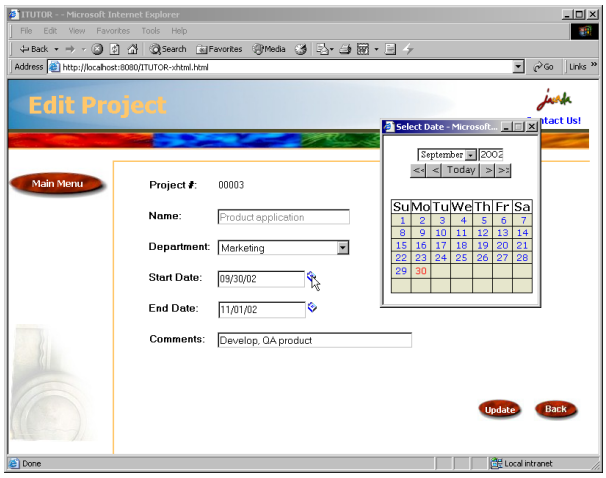
The Finished Product

This is what your window should look like when accessed in 'Add' mode in the HTML client.



The final product of your efforts in this exercise. The look of the PADDPR SubApplication, when accessed through 'Add' mode in an XHTML runtime.

This is what your window should look like when accessed in 'Edit' mode in the HTML client.



The final product of your efforts in this exercise. The look of the PADDPR SubApplication, when accessed through 'Edit' mode in an XHTML runtime.

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by navigating back to the Main Menu window with the 'Back' button and clicking the 'Exit Application' button in the Main Menu window.

2. Type **quit** in the Jacada Server command window to close the Jacada Server.
   or
   Use the shortcut **Ctrl+C** and answer yes to the message that appears.

3. Close the Jacada Server command window.

4. Close your browser window.

Exercise 4 - Jacada Studio For iSeries

How It Works: The Add / Edit Project Window

If you feel comfortable with the level of detail provided in this exercise, feel free to skip this section and go on to the next exercise. If you'd like to find out a bit more about how it all works, keep reading. In this section you learn about how everything comes together during runtime. Are you ready?

Error Handling On The Host

In the last exercise, error handling was performed server-side. In this exercise, error handling was performed on the host, with a little help from an IDK Method. To see how this works, open the PADDPR program on the host and take a look. In this example, we run validity checks to make sure that none of the fields are blank. Heres how it works:

In the IDK:

1. In the IDK, the Method **Tutorial_ErrorFocus_OnDisplay**, is attached to all input fields on PADDPR.

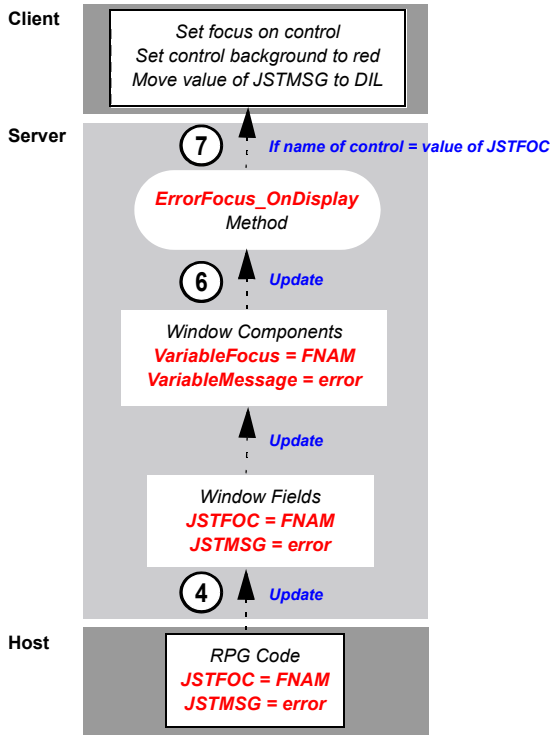
* *Note :* You can double click any of the controls that require input and go to the Events tab to see this association.

On the Host:

2. The nested IF statements check the valid fields to determine if the field is blank.
3. If the field is blank:
   - The name of the field is moved to the JSTFOC field
   - An error message from the "Application Messages" I specifications is moved to the JSTMSG field.

* *Note :* For example, if the value of the FDPDES field is equal to blanks... notice the ELSE statement (towards the bottom of the code) - the field name FDPDES is moved to the JSTFOC field and a message is moved to the JSTMSG field.

4. When the RPG program gets to a WRITE statement, the screen is sent to the Jacada Server with the value of the JSTFOC field and the JSTMSG field.



On the Jacada Server:

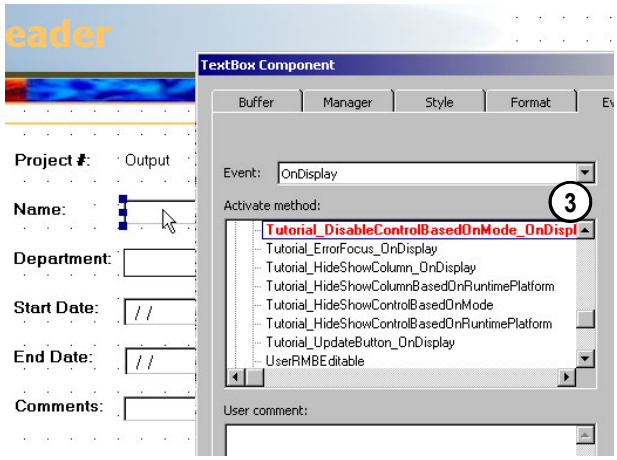
5. All OnDisplay Methods are executed
6. The **ErrorFocus_OnDisplay** Method gets the name of the current control and compares it to the value of the JSTFOC field *(Variable Representation name is VariableFocus)*.
7. If the value of the JSTFOC field matches the name of the current control, the Method:
   - Puts the focus on the current control
   - Sets the control's background color to red
   - Sends the Application message in the JSTMSG field to the Dynamic Information Line (DIL).

* *Note :* All Subapplications with required input have a variable representation called VariableFocus which is attached to the JSTFOC field in the buffer for the error handling to work.

Disabling the 'Name' Textbox in Edit Mode

In this project, the requirement is that the textbox containing the project name be disabled in *Edit* mode. To do this, the ***Tutorial_DisableControlBasedOnMode_OnDisplay*** Method was pre-attached to the *OnDisplay* event of the textbox component of the representation used to create the name field. This Method checks the value of the ***VariableMode*** variable (attached to the *FDMODE* field). If the variable's value is 'EDIT', the textbox is disabled. If the variable's value is anything other than 'EDIT', the textbox is enabled. To view this Method:

1. **Double-click the textbox** next to the 'Name' label on your window. *The Component Properties Dialog appears.*
2. Go to the **Events Tab**.
3. **Double-click** on the ***Tutorial_DisableControlBasedOnMode_OnDisplay*** Method.



Double-click the textbox next to the 'Name' label on your window. Go to the Events Tab and double click on the Tutorial_DisableControlBasedOnMode_OnDisplay Method

Exercise 5 - Jacada Studio for iSeries

Create the Work with Projects Window

Objectives:

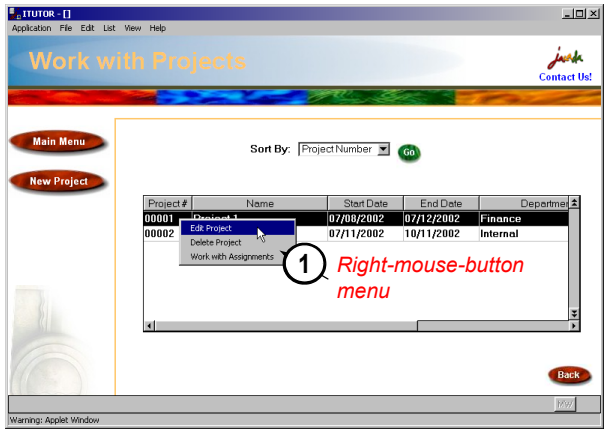
- To provide an exercise that replicates the Work with Projects window in the pre-packaged iTutor application
- To build a window that illustrates the graphical alternative to Subfile behavior through the use of a graphical table control
- To provide a brief explanation on how to construct and manipulate a table within the IDK
- To provide a capability to re-sequence or reload a table based on a Combobox selection of logical sort sequences
- To show the use of previously used fields with new short-list representations and the difference in use within a table control
- To expose the developer to the use of palette filters to improve usability of the IDK
- To go through the process of creating a new field and assigning an appropriate representation to that field when it is added to the display
- To add several lines of RPG code to implement one of the Jacada Studio table level APIs
- To differentiate Jacada Studio for iSeries table processing from iSeries green-screen Subfile processing
- To explain the different runtime behaviors of record selection between the Java and XHTML clients

In this exercise, you create the Work with Projects window of the ITUTOR Application. Your MYTUTOR Application should already contain seven of the eight Subapplications in the ITUTOR demo, four that were prepackaged for you and the Main Menu, Add/Edit Resource and Add/Edit Projects windows that you created in the previous exercises. In this exercise, we will create a new field that will be used to output a count of the total records in the table. The table record count is a value that is automatically maintained by the Jacada table API architecture. This is a handy feature that has no equivalent Subfile instruction in the green-screen world. The API document completely describes all the API modifiers and their implementations.

Tutorial Exercise Overview	IDK Walk-Through	Your First Application Exercise	Main Menu Exercise	Add / Edit Resource Exercise	Add / Edit Project Exercise	Work With Projects Exercise

The major steps to this exercise are:

1. **Window Design Specifications**
2. **Create the PPROJ Subapplication**
3. **Add the 'Sort by' Combobox to your Window**
4. **Add a Table to your Window**
5. **Add Representations to the Table**
6. **Create a Field in the KnowledgeBase**
7. **Associate Representation with Field and Create a Short List**
8. **Generate Runtime and Transfer Files**
9. **Modify The RPG Program**
10. **Compile DDS and Program File on Host**
11. **Run Application with a Java Client**
12. **Run Application with an XHTML Client**



The final product of your efforts in this exercise. The look and behavior of the PPROJ Subapplication, in a Java Runtime.

1. WINDOW DESIGN SPECIFICATIONS

First, you must define the differences in the Subapplication's interface, between when it is run with a Java client and when it is run with an XHTML client. The differences are:

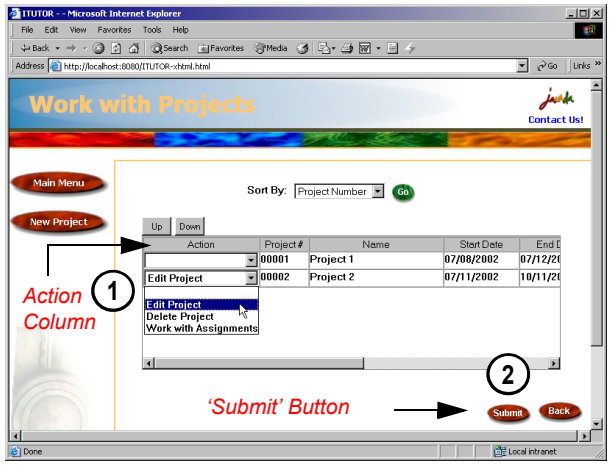
In the Java Runtime:

1. There is a *Right Mouse Button menu* in the table. Via this menu, user can perform actions on table rows.

In the XHTML Runtime:

1. There is an *Action Column* in the table. Via this column, user can perform actions on table rows.
2. There is a 'Submit' Button, to the left of the 'Back' button in the bottom right corner of the window.

In this Subapplication, you must take into account limitations stemming from the use of multiple client languages. For the sake of this example, the requirement is that actions be performed on table records in both the Java and XHTML runtimes. Since the HTML client is a browser window and browsers have their own right-mouse-button menus, it is not recommended to override the right-mouse-button browser functionality. In the HTML runtime you will create an action column, similar in functionality to the action columns that you are used to seeing on the iSeries. In the Java runtime, you will exploit Java's ability to allow for a right-mouse-button menu within the table.



The final product of your efforts in this exercise. The look and behavior of the PPROJ Subapplication, in an XHTML Runtime.

The native behavior of the Java runtime is to move to the next window as soon as an action is selected from the right-mouse-button menu. The native behavior of the HTML runtime is to move to the next window, only when given the command to "Submit". Therefore, a "Submit" button will only be needed in the HTML runtime.

2. CREATE THE PPROJ SUBAPPLICATION

To create the PPROJ Subapplication, you must first open both the Jacada Studio Interface Development Kit (IDK) and the MYTUTOR Application.

The New Subapplication Wizard

In this step, you create the PPROJ Subapplication in your MYTUTOR Application. In this exercise, you apply a prebuilt Window Layout called *Tutorial_LayoutforPPROJ* to your Subapplication.

1. From the **Subapplication Menu** > **New ...** *The New Subapplication Wizard is invoked.*
2. In the **New Subapplication Wizard**, specify the following Subapplication properties:

Subapplication Name:	**PPROJ**
Popup Window:	Unchecked
Window Layout:	**Tutorial_LayoutforPPROJ**

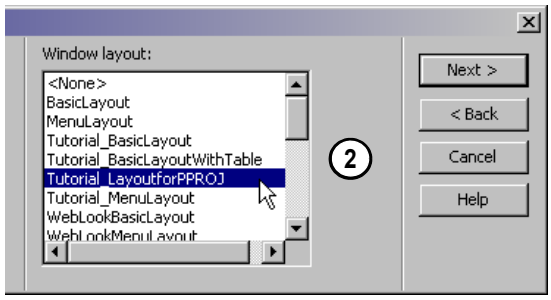
* *Note :* All other Subapplications in the ITUTOR Application containing tables were built using the *Tutorial_BasicLayoutWithTable* window layout. The window layout for PPROJ is special in that it contains *list controls* used to create the right-mouse-button menu functionality of the Java runtime.

Subapplication Description:	None

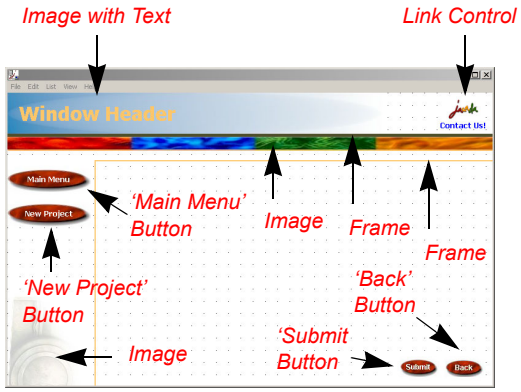
3. Click **Finish** to exit the New Subapplication Wizard.

Elements added to the Window by the Tutorial_ LayoutforPPROJ Window Layout

Did you notice the various elements that were brought into your Application when you selected the *Tutorial_LayoutforPPROJ* Window Layout in the New Subapplication Wizard?



In the Select Window Layout step of the New Subapplication Wizard, choose the Tutorial_LayoutforPPROJ Window Layout.



Elements added to the window by the Tutorial_LayoutforPPROJ Window Layout

Checking the Contents of the Window

Check the contents of the Window Components Palette to see the GUI components brought in by the Window Layout.

Check the contents of the Window Fields Palette to see the Window Fields brought in by the Window Layout.

* *Note :* Notice that the main difference between the previous window layouts and this one is in the number of window fields and variable window components. This window layout has fields and window components that were not present in the other window layouts. In order to perform actions on tables, table columns and table rows, a broader selection of fields and variable window components is needed.

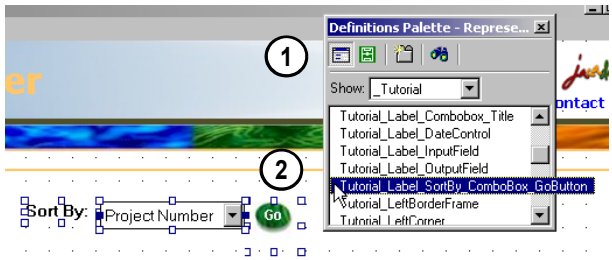
3. ADD THE 'SORT BY' COMBOBOX TO YOUR WINDOW

In this step, you will add the ***Sort by:*** combobox definition to the window. The *Sort by:* combobox definition will allow the user to sort records in the table according to preset criteria.

To Add the "Sort by" Combobox

1. Go to Representation Definition view of the Definitions Palette:
2. Drag the **Tutorial_Label_SortBy_ComboBox_GoButton** representation onto your window.

* *Note :* This particular combobox is fully functional because it has been preformatted in the KnowledgeBase, and code to support it's functionality exists in the prepackaged MYTUTORIAL host library. In the optional exercises at the end of this section, you will create another criterion for sorting, format it into this combobox and write the host code to support it.

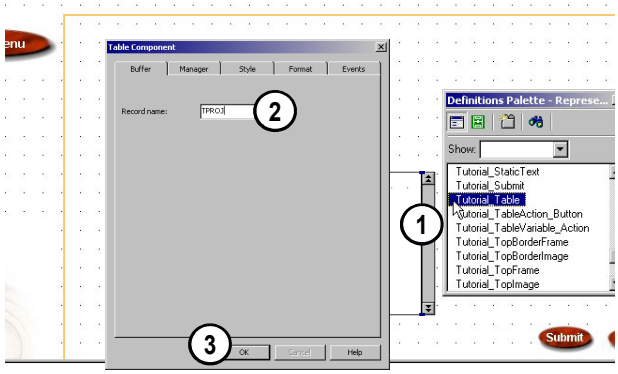


Drag the Tutorial_FunctionKey representation onto your window from the Definitions Palette.

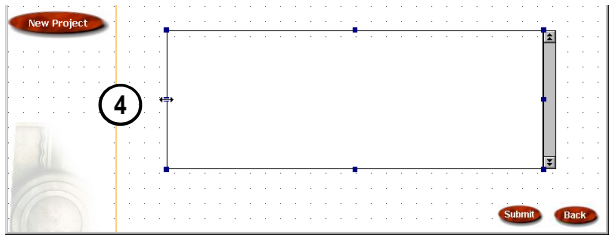
4. ADD A TABLE TO YOUR WINDOW

To add the table to your Subapplication, from Representation Definitions view of the *Definitions Palette*:

1. Drag the representation **Tutorial_Table** onto your window. *The Table Component Dialog appears.*
2. Type **TPROJ** in the *Record name* field of the Table Component Dialog.
3. Click **OK** to exit the Table Component Dialog.
4. Select the table representation in the window, resize it by stretching the table via the control handles at its side.



Drag the representation Tutorial_Table onto your window. Type 'TPROJ' in the Record name field.



Select the table and use the control handles at its sides to stretch it until it looks like this diagram

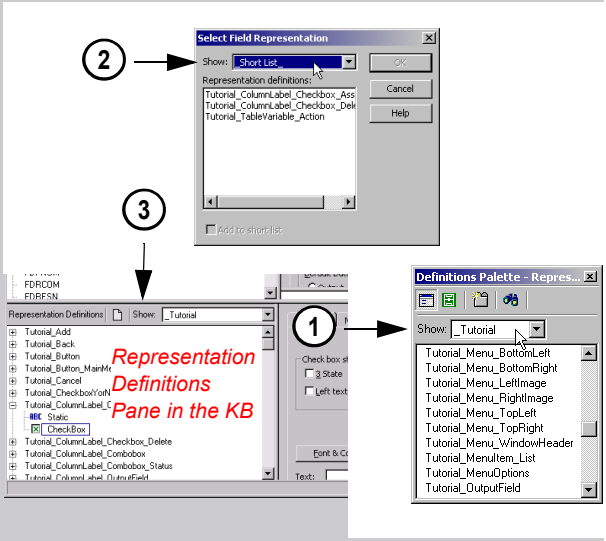
5. ADD REPRESENTATIONS TO THE TABLE

In this step, you will first use the **Show** filter to select a representation from the full list of representations in the ***Select Field Representation*** dialog that pops up when you drag a field onto the window. You will then add the remaining Representation Definitions to the table by dragging fields onto your table from ***Field Definition View*** of the Definitions Palette.

FYI: The Show Filter

*The **Show** filter can be found preceding lists of representation definitions that exist in the KnowledgeBase. The purpose of the Show filter is to allow the developer to view Representation Definitions according to specific criteria. For example, only representations connected to a certain field or representations with a certain prefix can be viewed in the list. Using this filter shortens the list of representations and allows the developer to focus on the representations that he is working on. The Show filter criteria are fully customizable by the developer and can be set to include as many criteria options as you choose. The **Show** filter exists in three places in the IDK interface:*

1. *Representation Definitions View* of the ***Definitions Palette*** in *Design View*
2. The ***Select Field Representation*** dialog in *Design View*
3. The ***Representation Definitions Pane*** of the *KnowledgeBase* interface (Lower Left Pane)



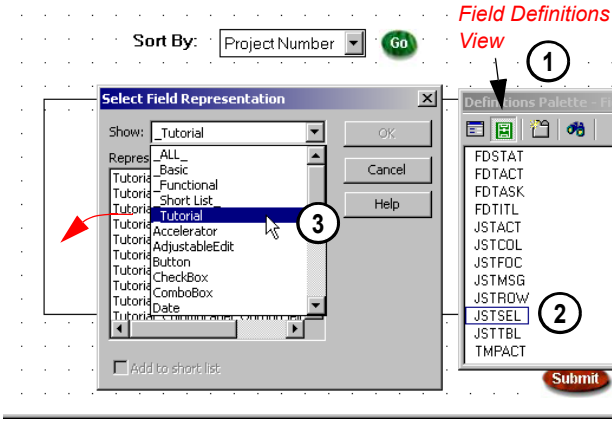
** Note :* Use the button with the picture of an owl (on your standard toolbar) to access the KnowledgeBase.

Add a Field to the Table and Use the Show Filter to Find the Right Representation

1. Click on Field Definition View of the Definitions Palette.
2. Drag the **JSTSEL** field onto the ***table***. *The Select Field Representation Dialog appears.*

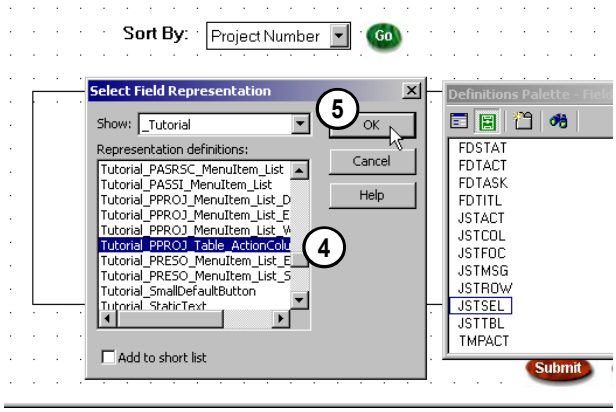
** Note :* Make sure you drag the field onto the **TABLE** and not the window.

3. Choose the **_Tutorial** filter from the **Show** filter combobox in the Select Field Representation dialog. *Only representations with the prefix 'Tutorial' are shown in the list of representations.*



Drag the JSTSEL field onto the table. Choose the -Tutorial filter from the Show filter combobox.

4. Select the **Tutorial_PPROJ_Table_ActionColumn** representation from the list of representations definitions.
5. **Click OK** to exit the dialog. *Your representation is added to the table. It is associated with the field that you dragged from the Definitions Palette.*



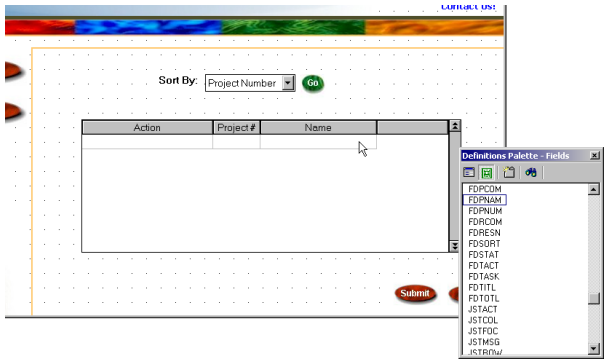
Select the Tutorial_PPROJ_Table_ActionColumn representation from the list of representations definitions. Click OK to add the representation to your table.

Add Remaining Representations to the Table

Add the following Representation Definitions to the table by dragging fields onto your table from ***Field Definition View*** of the Definitions Palette. Choose the representations associated with the field from the "Short List" of favorites.

The place in the table onto which you drag the representations is not important. Columns will be created in the order in which the representations are dragged onto the table. If the column order is not to your liking, you will learn how to change it later in this step. To add the remaining Representations to your Subapplication, refer to the table below:

Window Fields	Representations
FDPNUM	Tutorial_ColumnLabel_OutputField
FDPNAM	Tutorial_ColumnLabel_OutputField
FDDEPT	Tutorial_ColumnLabel_OutputField
FDBDAT	Tutorial_ColumnLabel_OutputField
FDEDAT	Tutorial_ColumnLabel_OutputField
FDPCOM	Tutorial_ColumnLabel_OutputField



As Fields are dragged onto the table, the table headers become populated with the label definitions assigned to the representations.

* *Note :* In the last exercise, we used these same data fields. We chose short list representations appropriate for displaying a label to the left of the field control. In this exercise, we are using the same fields with short list representations suitable for a table. In this case, you will see the label at the top of the column.

Testing the Table's Functionality

Because of the amount of representations added to the table, the columns in the table now exceed the table length. In Design View, you can not see all of the columns added to the table. To see all of the columns added to the table and verify the functionality of the horizontal scrollbar:

1. Go to **Test View** by clicking the  *Test View Icon* in the Standard Toolbar.
2. Click and drag the **Table Scrollbar** to scroll through the table columns and check that the correct columns were added in the correct order. *You can also click the arrow buttons to the left and right of your scrollbar to scroll the table in each direction.*

Manipulation of Table Columns

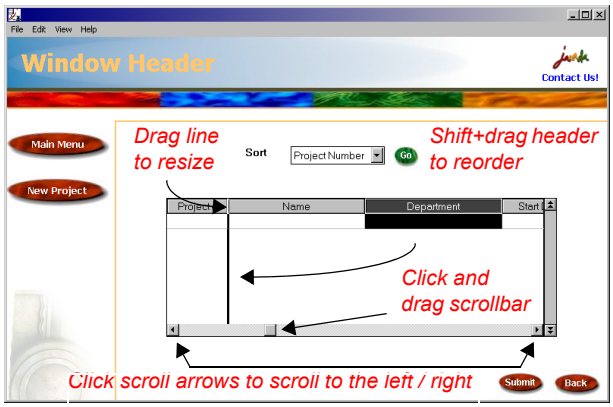
In Test View you can manipulate table columns. You can change the order in which the different columns are placed, and you can also change the width of each column.

To reorder table columns:

1. **Select a Table Column** by clicking on its header.
2. In **Test View** > Hold down the **Shift key** on your keyboard.
3. **Drag** the table column to the left or right to move the column to its new location.

To change the width of a table column:

1. In **Test View** > Place your **Cursor** on the line that divides between two table headers. *The cursors form.will change.*
2. **Drag** the line between the table columns to the left or right to resize the adjoining columns.



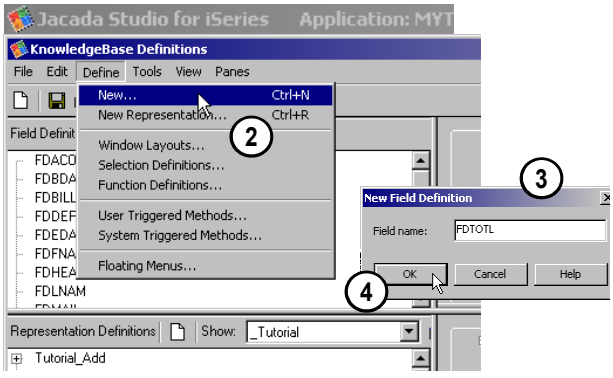
In Test View you can scroll through the table, resize and reorder table columns.

6. CREATE A FIELD IN THE KNOWLEDGEBASE

In this step, we create a field that will be used to output a count of the total records in the table. We will then define the field's properties. We will define this field globally, in the KnowledgeBase.

To create the field:

1. Click the KnowledgeBase icon  in the Standard Toolbar. *The KnowledgeBase window opens.*
2. From the **Define** menu > choose **New...** T*he New Field Definition dialog appears.*
3. **Type FDTOTL** into the *Field name* area of the New Field Definition dialog.
4. **Click the OK button** to exit the New Field Definition dialog. *The new FDTOTL field is added to the list of fields in the Field Definitions Pane. The Field Definition Properties Pane becomes active.*



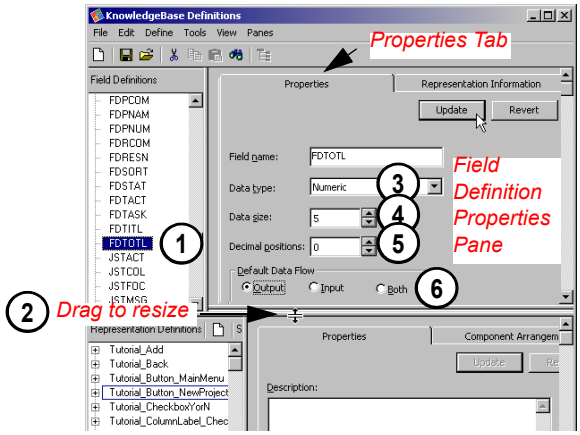
From the Define menu > choose New... > Type FDTOTL into the Field name area of the New Field Definition dialog > Click the OK button to exit the dialog.

To define the field's properties:

1. Select the FDTOTL field in the ***Field Definitions Pane***.
2. **Drag the interface divider downward** until you can see the three types of Default Data Flow (Output, Input, Both) in the ***Properties Tab*** of the ***Field Definition Properties Pane***.

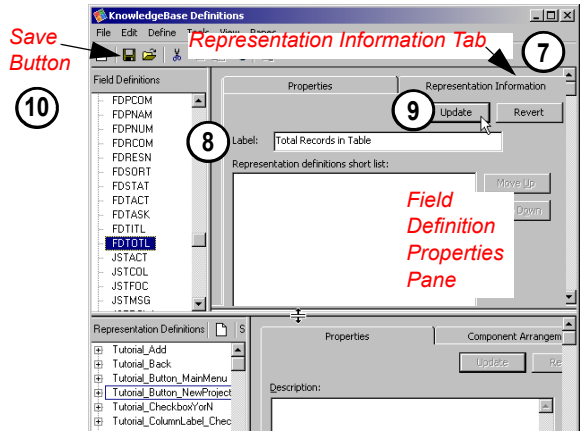
In the ***Properties Tab*** of the ***Field Definition Properties Pane:***

3. Set *Data Type* to **Numeric**
4. Set *Data Size* to **5**
5. Set *Decimal Positions* to **0**
6. Set *Default Data Flow* to **Output**



Set field properties in the Properties Tab of the Field Definition Properties Pane.

7. Click the ***Representation Information Tab*** in the *Field Definition Properties Pane*.
8. Type **Total Records in Table** into the *Label* field.
9. Click the **Update** button to update the field's properties in the KnowledgeBase.
10. Save your KnowledgeBase settings by clicking the **Save** button in the KnowledgeBase interface.



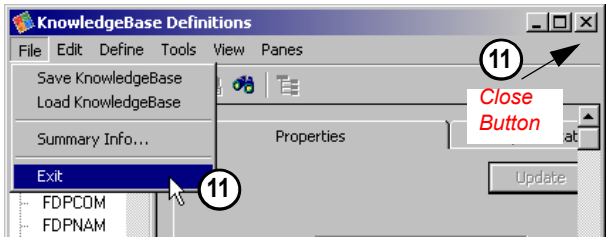
Type 'Total Records in Table' into the Label field in the Representation Information Tab of the Field Definition Properties Pane.

11. To **exit the KnowledgeBase**:

From the **File** menu > choose **Exit**.

or

Click the Close button on the upper right corner of the window.



To exit the KnowledgeBase, choose 'Exit' from the File menu or click the Close button on the upper right corner of the window

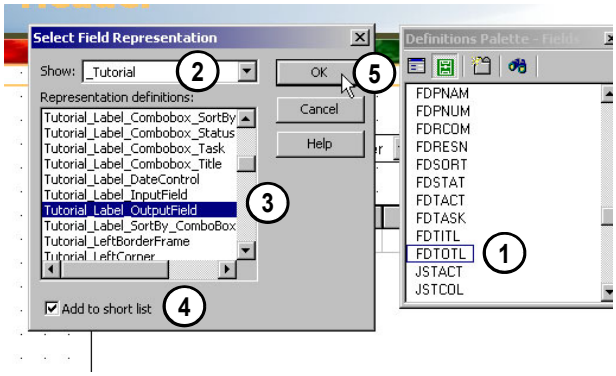
7. Associate Representation with Field and Create a Short List

In this step, you add the new FDTOTL field to the window. You will then associate a representation definition with this field, and put the chosen Representation Definition in the short list of favorite representations for this field. These changes will be saved to the KnowledgeBase, and will allow you to reuse this field, with its new short list of favorites, throughout the application. Doing this will ensure that whenever this field is dragged onto a window, the associated Representation Definition will show up in the field's short list. If the Representation Definition is chosen from the fields short list, it will be automatically connected to that field in the buffer.

Add FDTOTL to the Window

To add the new field that you created to the window:

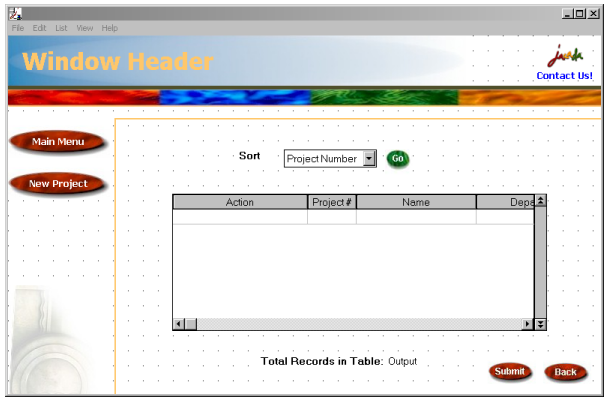
1. Drag **FDTOTL** onto your Window ***(not Table)*** from Field Definitions View of the *Definitions Palette*. Drag the field onto the area under the table. *The Select Field Representation dialog appears.*
2. Choose the **_Tutorial** filter from the **Show** filter combobox in the Select Field Representation dialog. *Only representations with the prefix 'Tutorial' are shown in the list of representations.*
3. Select the **Tutorial_Label_OutputField** representation from the list of representations definitions.
4. Check the **Add to short list checkbox**.
5. **Click OK** to exit the dialog. *Your representation is added to the window. It is now associated with the field that you dragged from the Definitions Palette.*
6. Click the **Apply Design Changes** button in your Standard Toolbar to apply the changes of your design.



Drag FDTOTL onto the window. Select Tutorial_Label_OutputField as the associated representation definition and check the 'Add to short list' checkbox.

What your Window Should Look Like

This is what your windows should look like at the end of the last step. Notice that the label of the representation that you added, automatically received the text 'Total Records in Table'. If the representation is not placed correctly, select the representation components and place them according to the diagram to the right. You are now ready to generate a runtime.



What your Window Should Look Like at the end of the last step.

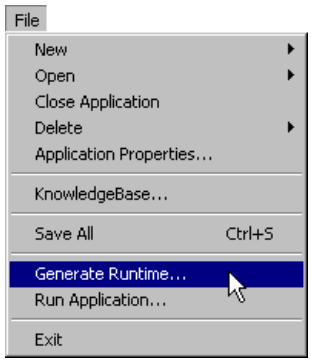
8. GENERATE RUNTIME AND TRANSFER FILES

To generate a Runtime and transfer the files to host:

1. From the **File** menu > choose **Generate Runtime...** *The Generate Runtime Wizard is invoked.*

2. In the **Generate Runtime Wizard**, click **Next** to accept the following default settings:

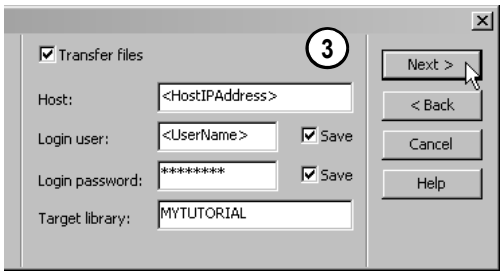
Runtime Type:	**Java and XHTML**
Jacada Server Platforms:	Windows NT(2000) x86
Subapplications to include:	All
Subapplications to Process:	**Only new and modified**



Generate a Runtime in order to compile your Subapplication into an executable

3. In the **File Transfer** screen, specify the following information, then click **Next**:

Transfer files:	**<Checked>**
Host:	**<YourHostIPAddress>**
Login User:	**<YourUserNameOnHost>**
Login Password:	**<YourPasswordOnHost>**
Target Library:	**MYTUTORIAL**

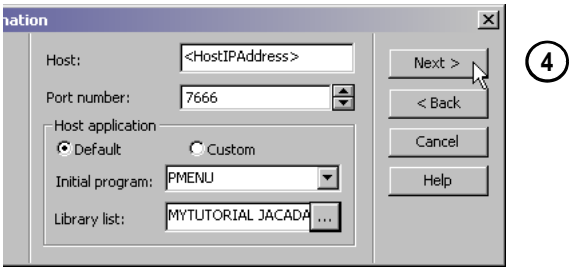


Fill out the File Transfer screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, specify your respective Library (i.e. TUTORIAL01) as the Target Library.

4. In the **Specify Host Connection and Application Information** screen, specify the following information, then Click **Next**:

Host:	**<YourHostIPAddress>**
Port Number:	**7666**
Initial Program:	**PMENU**
Library List:	**MYTUTORIAL JACADA**



Fill out the Host Connection and Application Information screen of the Generate Runtime Wizard

* *Note :* If you are working in a multi-evaluator environment, remember to have your respective Library (i.e. TUTORIAL01) be the first library in the Library List entry and include the JACADA library after your library.

5. Click **Finish** to exit the Generate Runtime Wizard, and commence with the compilation process. *The Generating the Runtime dialog appears.*

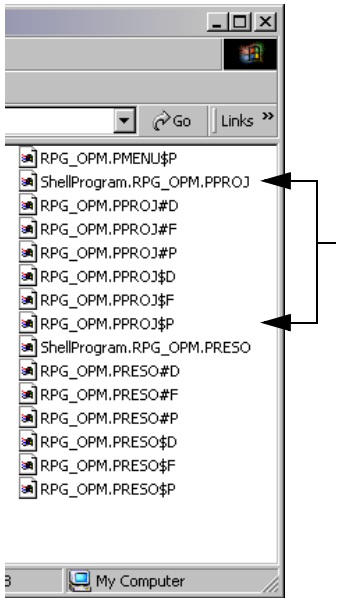
6. Wait for the Runtime Generation process to come to an end.

7. Click **Close** to exit the Generating the Runtime dialog.

Files Created By the Generate Runtime Process on the Development PC

Subapplications containing tables contain two separate records: a window record and a table record. Look in the **JacadaStudio\appls\MYTUTOR\gds** directory - **7 files** were created on the development PC for this Subapplication:

1. **RPG_OPM.PPROJ$D** - *The Window Record DDS physical file*
2. **RPG_OPM.PPROJ#D** - *The Table Record DDS physical file*
3. **RPG_OPM.PPROJ$P** - *The Parameter List Copybook for the Window Record*
4. **RPG_OPM.PPROJ#P** - *The Parameter List Copybook for the Table Record*
5. **RPG_OPM.PPROJ$F** - *The File Specification Copybook for the Window Record*
6. **RPG_OPM.PPROJ#F** - *The File Specification Copybook for the Table Record*



The files created on the development PC as a result of the Runtime Generation Process

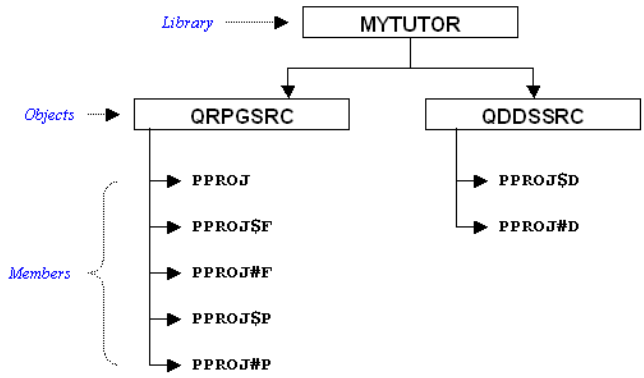
* *Note :* The dollar sign ($) is used in the name of the files generated for window definitions. The pound sign (#) is used in the name of the files generated for table definitions.

7. **ShellProgram.RPG_OPM.PPROJ** - *The Shell Program*.

* *Note :* Only one Shell Program is generated per Subapplication.

Libraries Objects and Members Created by the Generate Runtime Process

When you choose to transfer the files created by the Runtime Generation process to the host by checking the *Transfer files* checkbox in the *Transfer Files* screen of the Generate Runtime Wizard, the library structure in the diagram to the right is created on the host in the Target Library that you specified. Because the RPG code of the PPROJ subapplication was prepackaged for you, the shell program is the only file that is not transferred.



The libraries, objects and members of the PPROJ window on the iSeries.

9. MODIFY THE RPG PROGRAM

In this section we will add RPG code to retrieve a count of the records in the table and move that value into the field we created earlier (FDTOTL) in order for it to be displayed. The Jacada Studio table management API automatically maintains a table record counter. In order to retrieve this counter, we will insert a small piece of RPG code that will set the API Modifier Field (GDSMOD) to a value of "GDSGL" (get line), and then execute a READ to the table record. The READ will retrieve the current count of table records and place that value in an API field called GDSEC. From there we'll move that value to the FDTOTL window field. The WRITE instruction that actually sends the window is already in place.

* *Note :* For a full description of all the API fields and their uses, please refer to the API document.

In the screen-shot to the right, lines 87.01 through 87.04 were added to the prepackaged PRG code.

Manually insert the following lines of code before line 88 in the prepackaged PRG program. Line 88 contains the comment "Write and Display Window"

```
C* Get Record Count in Table
C                MOVE   GDSGL        GDSMOD
C                READ   TPROJB                        99
C                MOVE   GDSEC        FDTOTL
```

```
FMT C  .....CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqCo
0085.00      C           FDSORT    IFEQ 'STAR'
0086.00      C                     EXSR LODLS2
0087.00      C                     ENDIF
0087.01      C*GET RECORD COUNT IN TABLE
0087.02      C                     MOVE GDSGL     GDSMOD
0087.03      C                     READ TPROJB                  99
0087.04      C                     MOVE GDSEC     FDTOTL
0088.00      C*Write and Display Window
0089.00      C                     WRITEPPROJB
0090.00      C                     READ PPROJB                  99
0091.00      C*
0092.00      C           JSTACT    IFNE 'EXIT'
 Prompt type . . .    *      Sequence number . . .  0088.00

 Data area
 . 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+
 Write and Display Window

 F3=Exit   F4=Prompt   F5=Refresh          F11=Previous record
 F12=Cancel            F23=Select prompt   F24=More keys

MA    a
```

The modified RPG program with the added lines of code.

10. COMPILE DDS AND PROGRAM FILE ON HOST

In our previous exercises all the window data was defined in a single Physical File DDS member and was managed with a single Special File. Table processing brings another dimension to the development process. Windows that implement Tables are conceptually similar to Display Files that implement Subfiles. The Window itself can be related to a Subfile Control Record Format while the Table portion can be related to a Subfile Record Format. Just like I/O to a Subfile Record Format takes place in local program memory on the host, Table I/O is also a local memory operation. Similarly, I/O to the sub-application Window that contains the Table actually gets sent to the client for display, just like I/O to a Subfile Control Record Format.

In order to differentiate between Table I/O and Window I/O, Jacada Studio generates separate Physical File DDS members and Special File definitions for the Window record and the Table record. This separation at the file level instead of a format level provides some very advanced flexibility in determining where and how a table can be loaded at the host. (Advance table management is not discussed in this tutorial.)

As you compile the files and program on the host, remember that two Physical File DDS members must be compiled because there are two pieces to the display. Also remember to add the JACADA library to your library list.

Compile the DDS physical files **PPROJ$D** and **PPROJ#D** as well as the RPG program file **PPROJ** in your **MYTUTORIAL** library.

11. RUN APPLICATION WITH A JAVA CLIENT

In order to run your Application, you must first verify that the Jacada monitor is active. Only then can you run your Application.

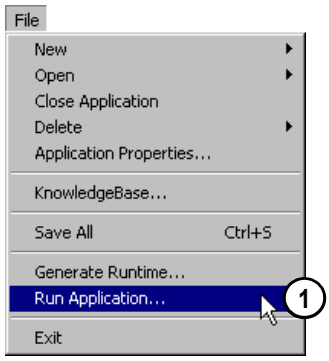
Run your Application with a JAVA Client

Run the executable created during the Runtime Generation process from within the development environment:

1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*
2. In the **Run Application Wizard**, agree to the default Runtime properties, by clicking the **Next** button, when prompted for:

Runtime Type:	**Java**
Web Server:	Integrated HTTP Service
Application URL:	http://localost:8080/ MYTUTOR.html

3. Click **Finish** to exit the Run Application Wizard. *The Jacada Server is activated and your Default Browser window is opened to the Jacada MYTUTOR.html page.*
4. Type your iSeries **Username** and **Password**.
5. Click the **OK** button.

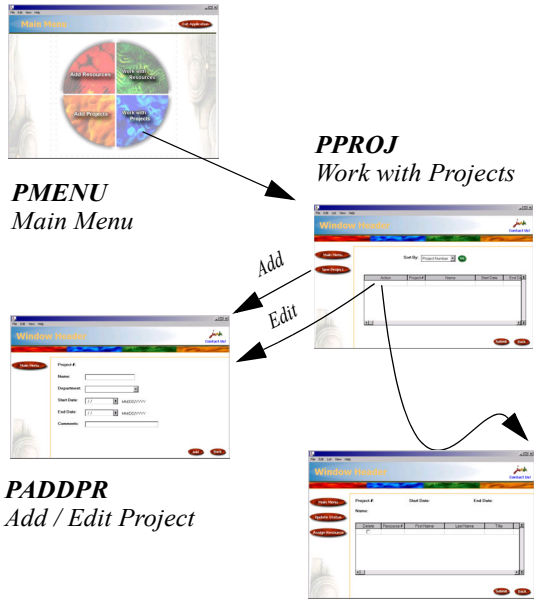


Run your Application through the development environment by choosing File > Run Application

Navigate to the Work with Projects Window

Navigate to the PPROJ Subapplication by using the diagram provided for you to the right.

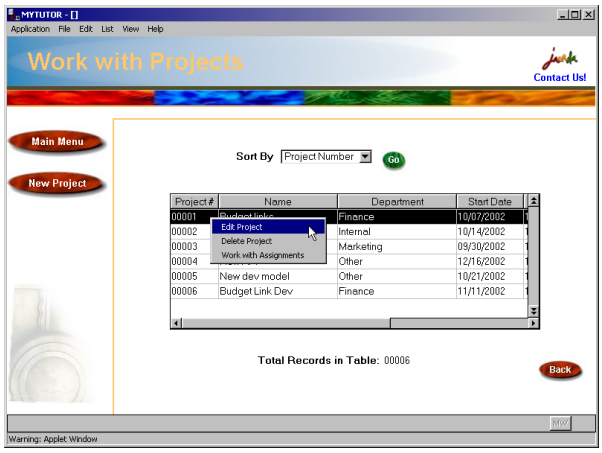
* *Note :* Now that this window is built, you can choose the **Edit Project** option from the action column to navigate to the Add/Edit Projects window (PADDPR) in Edit mode.



Navigate to the PPROJ Subapplication and right-click on the table rows to see the menu options.

What your Window Should Look Like in Java

This is what your windows should look like in the Java client.In the Subapplication, right-click on table rows to invoke the selection options. Select the options to invoke the actions associated with them.



The final product of your efforts in this exercise. The look and behavior of the PPROJ Subapplication, in a Java Runtime.

Close the Jacada Server

When you are done running your Application

1. Exit the Application and end your host session by navigating back to the Main Menu window with the 'Back' button and clicking the 'Exit Application' button in the Main Menu window.
2. Type **quit** in the Jacada Server command window to close the Jacada Server.
3. Type **exit** in the Jacada Server command window to close the Jacada Server command window.
4. Close your browser window.

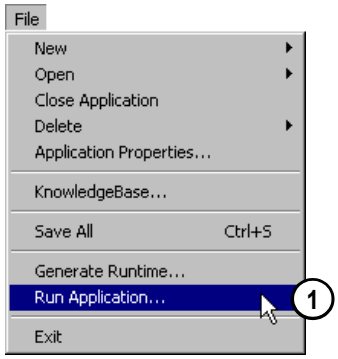
12. RUN APPLICATION WITH AN XHTML CLIENT

Run the executable created during the Runtime Generation process from within the development environment:

1. From the **File** menu > choose **Run Application...** *The Run Application Wizard appears.*
2. In the **Run Application Wizard**, agree to the default Runtime properties, by clicking the **Next** button, when prompted for:

Runtime Type:	**XHTML**
Port Number:	8080
Web Server:	Integrated HTTP Service
Application URL:	http://localost:8080/ MYTUTOR-xhtml.html

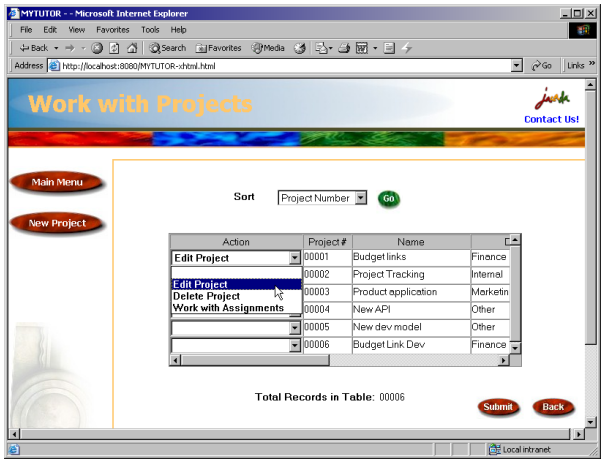
3. Click **Finish** to exit the Run Application Wizard. *The Jacada Server is activated and your Default Browser window is opened to the Jacada MYTUTOR-xhtml.html page.*
4. Type your iSeries **Username** and **Password** into the appropriate fields.
5. Click the **OK** button to run your HTML client Application.
6. Navigate to the Work with Projects Window.



Run your Application through the development environment by choosing File > Run Application

What your Window Should Look Like in HTML

This is what your windows should look like in the HTML client.In the Subapplication, click the *Action Column* in the record row you wish to select. Select the option to invoke the action.



The final product of your efforts in this exercise. The look and behavior of the PPROJ Subapplication, in an XHTML Runtime.

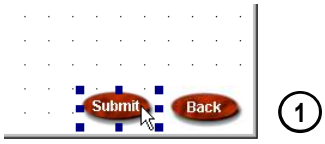
Exercise 5 - Jacada Studio For iSeries

How It Works: The Work with Projects Window

If you feel comfortable with the level of detail provided in this exercise, feel free to skip this section and go on to the next exercise. If you'd like to find out a bit more about how it all works, keep reading. In this section you learn about how everything comes together during runtime. Are you ready?

The 'Submit' Button

A "Submit" button was added to the Subapplication by the window layout. For the sake of this example, we have created a method that shows or hides the "Submit" button according to the client type. In this step, you see how the "Submit" button's functionality was controlled in such a way, that will allow the button to only be shown in the HTML runtime.



Double Click the Submit button to open the Component Properties Dialog.

To see how this was done:

1. **Double-Click** the Submit button on your Subapplication. *The Component Properties Dialog appears.*
2. Go to the **Events** Tab of the Component Properties Dialog.
3. In the **Event** combobox > choose the **OnDisplay** event.
4. Double click the **Tutorial_HideShowControlBasedOn RuntimePlatform** method.

```
#0 = DoMethod: Receiver: `System` Method: GetSharedUserVariable Parms: ( `"SUV_RTFlag"` )
If: Cond: `#0 == "Java"`
  DoMethod: Receiver: `this` Method: HideControl Parms: ( )
EndIf:
```

The Tutorial_HideShowControlBasedOnRuntimePlatform method.

* *Note :* This is a very simple method that checks the value of the *SUV_RTFlag* "shared user variable" (variable from variable pool shared by both client and server). If the value of this variable equals the string "Java", this button is hidden.

5. Click **OK** to exit the Component Properties Dialog.

Actions Performed on Table Records

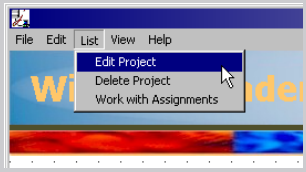
The requirement is that for each table record, three action options be available during runtime. Since this table consists of a list of projects and their associated data, the possible actions for table records in this table are:

Work with Assignments	Takes you to the '*Work with Assignments*' window
Edit Project	Takes you to the '*Add/Edit Projects*' window
Delete Project	*Deletes* table record

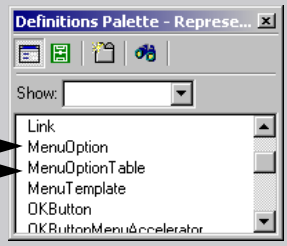
FYI: The MenuOption and MenuOptionTable Window Components

*The **MenuOption** component is a standard generic component that ships with the default Jacada Studio KnowledgeBase (look for it in the Window Definitions Palette). When dragged onto the window or table, it creates a right-mouse-button menu through which the user can access the actions defined in its component properties dialog. It also creates a **Menu Item** in a **Menu** (that you specify) on the Subapplication **Menu Bar**. In the default Jacada Studio KnowledgeBase there are two representations that create right-mouse-button functionality: **MenuOption** and **MenuOptionTable**. The First handles data flow through fields defined in the Window records buffer. The latter handles data flow through fields defined in the table records buffer.*

A Menu with Menu Items in the Subapplication menu Bar



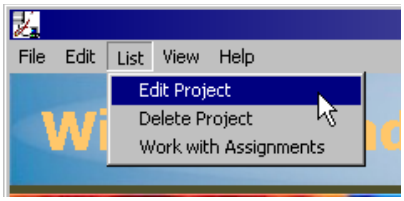
The MenuOption Representations



* *Note :* Remember that a Subapplication with a table consists of two records: The window record and the table record. We will discuss the implications of this later in the *Generate Runtime* step.

Table MenuOption Component in the Java Runtime

To exploit the ability of the Java client language to allow for a right-mouse-button menu within the table, the generic **MenuOptionTable** representation was used to create a ***Menu*** with three ***Menu Items***. Unknowingly, you have already added these items to the Subapplication through the *Tutorial_LayoutforPPROJ* window layout.



In Design View > Click on the 'List' Menu > select and click the Edit Project menu item

Action Column in the XHTML Runtime

In the XHTML runtime, the ***Action** Column* is the first column that you see in the table. The ***Tutorial_TableVariable_Action*** representation used to create this action column consists of two window components: A *Static* (text) component used to display the text on the column header and a *Combobox* component used to display the action options available to the user. During runtime, a combobox will appear next to each record in the table. It will provide the user with a pull-down menu, from which he will choose an action that will be performed on the associated table record when the window is submitted.This representation is connected to a field called **JSTSEL**, through which data will be transferred to and from the host during runtime.

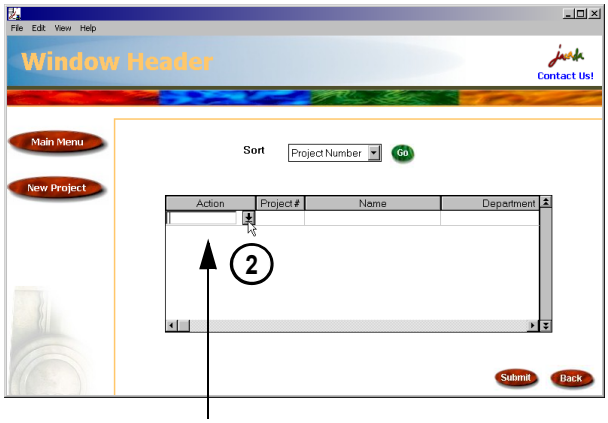
* *Note :* This particular combobox is fully functional because it has been preformatted in the KnowledgeBase, and code to support its functionality exists in the prepackaged MYTUTORIAL host library. In the optional exercises at the end of this section, you will create a combobox option, format its host and window values and write the host code to support it.

Because the table actions available through the Action Column will be accessible in the Java runtime via the MenuOption component, we will hide the action column in the Java runtime to avoid redundancy. In this step, you will be introduced to the KnowledgeBase method used to control the appearance of the Action Column according to the chosen runtime platform. This method hides the Action Column in the Java runtime.

View the XHTML Action Column Combobox in Test View

To see the Action Column Combobox in Test View:

1. Go to **Test** View.
2. **Click** the area *UNDER* the table header with the title - 'Action'.



In Test View, click the area UNDER the table header with the title - 'Action' to see the Action Column Combobox.

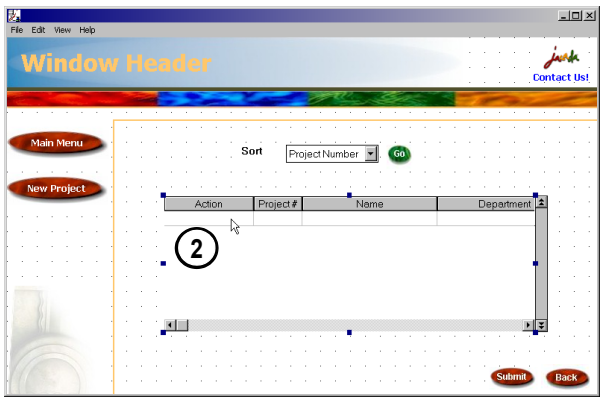
View the XHTML Action Column Properties

In order to pass values to the host during runtime, the combobox component of the ***Tutorial_TableVariable_Action*** representation used to create this action column has already been connected to the JSTSEL buffer field. Since the combobox component sits in a table, we cannot click the representation and see the associated field selected in the Window Fields Palette (we only see information associated with the selected table). Therefore, to view the connection between the combobox component and the buffer field:

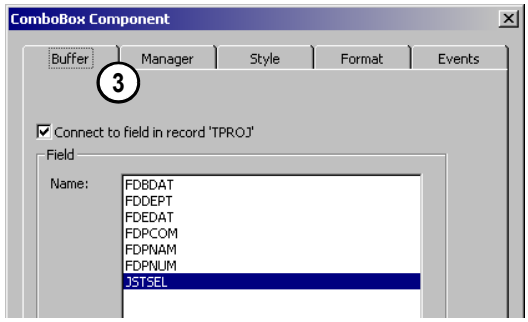
1. Go to **Design View**.
2. **Double-click** the first record row in the area *UNDER* the table header with the title - 'Action'. *The Combobox Component Properties Dialog appears.*

* *Note :* Make sure that the component properties dialog says 'Combobox Component' in the header and not 'Static Component'. If it says 'Static Component', this means that you double clicked the header and not the combobox. Cancel out of the dialog and try again.

3. In the Combobox Component Properties Dialog > go to the **Buffer Tab**. *Notice the JSTSEL buffer field attached to the combobox.*



In Design View, double-click the area UNDER the table header with the title - 'Action' to access the combobox component properties dialog.



Go to the Buffer Tab of the Combobox Component Properties Dialog to view the connection between this component and the buffer field associated with it.

Showing and Hiding the XHTML Action Column According to Runtime Platform

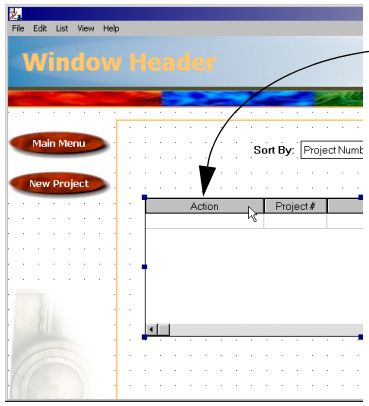
In order to hide the Action Column in the XHTML runtime, a method called ***Tutorial_HideShowColumnBasedOnRuntimePlatform*** was attached to the OnDisplay event of the Static (text) component of the representation used to create the Action Column. This method checks the value of a variable that is populated when the Application is launched. If the value of the variable equals Java, the column is hidden. If the value of the variable equals XHTML, the column is shown. To view the association between the column and the method:

1. **Double-click the Action Column Header.** *The Static Component Properties Dialog appears.*
2. Go to the **Events Tab**.
3. View the ***Tutorial_HideShowColumnBasedOnRuntimePlatform*** method attached to the OnDisplay event of the static component.

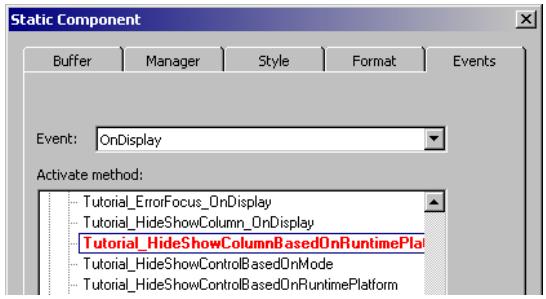
* *Note :* Notice that the method was attached to the column header (Static component) and not the combobox component.

In Both Cases:

The RPG code loops through the table and processes each table entry according to the value set by the user.



Double-click the Action Column header.



Go to the Events Tab of the Static Component Properties Dialog to view the method attached to the Static component's OnDisplay event.

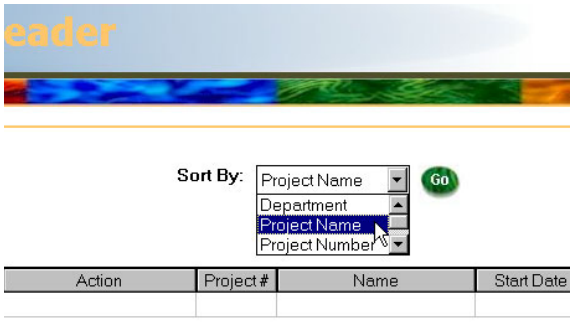
Exercise 5 - Studio for iSeries, Jacada

Optional Exercises: Let's Break it Down - PPROJ

If you're feeling like you need to break it down some more, how about adding a sort option to the 'Sort by' combobox. In this section you will learn all of the steps necessary to sort the table records by project name. You will create a combobox option, format it's host and window values in the IDK and write the host code to support it. Are you ready? Well, what are you waiting for?

The major steps to this exercise are:

1. **Add a Value to the 'Sort by' Combobox**
2. **Associate Screen and Window Values**
3. **Generate a Runtime**
4. **Create Fspec and Modify the RPG Program**
5. **Compile Host Code and Run Application**



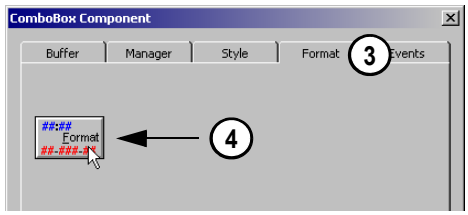
The final look of the Representation that you will build in this exercise.

1. Add a Value to the 'Sort by' Combobox

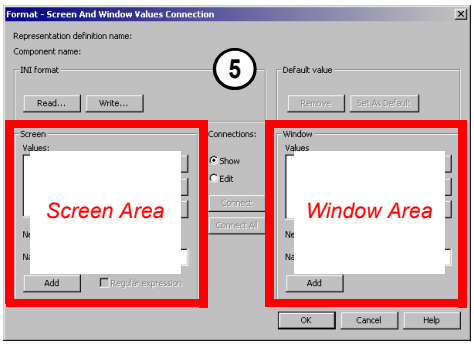
1. Go to **Design View**.
2. **Double-click** the combobox component of the 'Sort by' representation. *The Combobox Component Properties Dialog appears.*
3. Go to the **Format Tab** in the Component Properties Dialog.
4. Click the **Format Button**. *The Format - Screen and Window Values Connection Dialog appears.*

In the Format - Screen and Window Values Connection Dialog:

5. Identify the **Screen** and **Window** areas of the dialog. *See the image diagram to the right for help identifying these areas.*



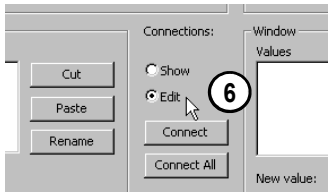
Go to the Format Tab in the combobox component's properties dialog and click the Format Button.



Identify the Screen and Window areas of the dialog.

** Note :* You will define the value passed to the host in the *Screen* area. You will define the string that appears in the combobox during runtime in the *Window* area.

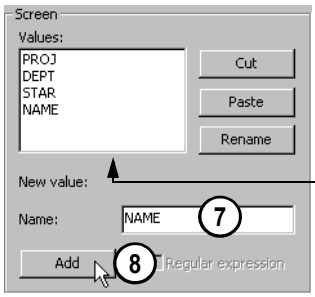
6. In the **Connections** area > select the **Edit** radio button



Select the Edit radio button in the Connections area

7. Type **'NAME'** into the name field in the ***Screen Area*** of the dialog,.

8. Click the **Add** button to add the value to the values list in the ***Screen Area***.

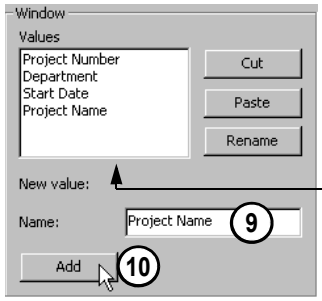


Type 'NAME' into the name field in the Screen Area of the dialog and Click the Add Button

Screen Area Values List

9. Type **'Project Name'** into the name field in the ***Window Area*** of the dialog,.

10. Click the **Add** button to add the value to the values list in the ***Window Area***.



Type 'Project Name' into the name field in the Window Area of the dialog and Click the Add Button

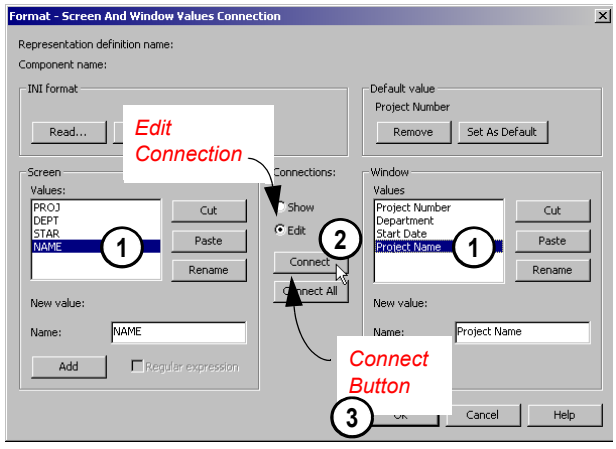
Window Area Values List

2. ASSOCIATE SCREEN AND WINDOW VALUES

1. Select the **NAME** value from the values list in the ***Screen Area*** and the **Project Name** string from the values list in the ***Window Area***.

2. Click the **Connect** button in the ***Connections*** Area.

3. Click the **OK** button to exit the *Format - Screen and Window Values Connection Dialog*.



Select a window value from the window value list and its corresponding screen value from the screen value list > click the Connect button to associate the two values during runtime.

View the Connection Between Screen and Window Values

1. In the **Connections** area > select the **Show** radio button.

2. Select any of the values in either of the values lists to see their associated counterparts.

3. GENERATE A RUNTIME

In the IDK, **Generate a Runtime** to compile the Application, transfer **only new and modified** files to the host.

4. CREATE FSPEC AND MODIFY THE RPG PROGRAM

In order to add make the *Project Name* sorting option functional, the screen values formatted into the combobox must be supported by the RPG code and an F spec must be created. Perform the following steps on the host machine to make the *Project Name* sorting option functional. Use the code samples in the column to the right to guide you through the procedure.

1. **Add an F spec** for the ***LPROJEC4*** logical file sorted by Project Name to the RPG program. On the continuation line of the F spec, rename record PROR in column 54 to ***PROR4***.
2. In the RPG program PPROJ, **add IF logic** after the *DO* loop to check the sort field (***FDSORT***) for the ***NAME*** value that you formatted into the combobox > then call the ***LODLS3*** subroutine (*LODLS3 subroutine will be created in the next step*).
3. **Add the LODLS3 subroutine** to your RPG program by copying the ***LODLST*** subroutine and pasting it after the LODLS2 subroutine.
4. In the LODLS3 subroutine, change SETLLPROR to **SETLLPROR4**. Change PROR to **PROR4**.

```
                                                    1
FLPROJEC4IF  E           K       DISK
F            PROR                       KRENAMEPROR4
```

Add an F spec for the LPROJEC4 logical file sorted by Project Name. Rename record PROR to PROR4.

```
                                                    2
C           FDSORT     IFEQ 'NAME'
C                      EXSR LODLS3
C                      ENDIF
```

Add IF logic after the DO loop to check the FDSORT field for the NAME value and call the LODLS3 subroutine

```
                                                    3
C           *LOVAL     SETLLPROR
C                      READ PROR                88
```

The LODLST subroutine that must be copied in order to create the LODLS3 subroutine.

```
                                                    4
C           *LOVAL     SETLLPROR4
C                      READ PROR4               88
```

In the LODLS3 subroutine, change SETLLPROR to SETLLPROR4. Change PROR to PROR4.

5. COMPILE HOST CODE AND RUN APPLICATION

You're all done! So you want to run your Application? Its easy, just:

1. On the host machine, **Compile** the PPROJ RPG program.
2. Make sure the **Jacada Monitor** is Active.
3. From the IDK, **Run the Application**.

Congratulations!