Some Background Information

This chapter covers the following topics:

- If you Change the Adapter
- Name Binding between Controls and Adapter
- Data Exchange at Runtime
- Files and their Locations
- Application Project Management
- Usage of Enhanced Development Tools

If you Change the Adapter

You can now add more functions to your "Hello World!" application by yourself. This means that you will both work with the layout description inside the Layout Painter as well as change the code of the adapter.

Changing the adapter’s code requires that you replace your existing class files in the *<installdir>/tomcat/webapps/cis/cisyourfirstproject/appclasses/classes* directory by new ones. What does this mean for the runtime environment? The runtime does not see these recompiled classes because it already loaded the existing classes. So there is no difference how the application performs by default.

However, you can explicitly force the server to reload all application classes and to use the newest class version. The server is clever enough to keep old classes for existing sessions, which means that users who are already logged on will not be disturbed. But new sessions will get the newest classes.

The task to update the classes is directly triggered from the Layout Painter by choosing the save button. This tells the runtime to use new classes for new sessions. Since the preview area always opens a new session, it will be up-to-date whenever you choose the preview button.

Therefore, you can develop continuously, without restarting the GUI-server Tomcat environment.

For fast readers:

The class loader management mentioned above is designed to be used at design time. At runtime, especially if running in non-Tomcat environments, you should use the application server’s web class loader to load the adapter classes. Details are provided in other sections of the Application Designer documentation.

Name Binding between Controls and Adapter

Which are the critical parts when building the "Hello World!" application?

- The PAGE control in the layout points to the class name of the adapter (property `model`).

- The FIELD control in the layout points to the property name of the adapter (property `valueprop`).
- The BUTTON control in the layout points to the method `sayHello()` of the adapter (property `method`).

There is a name binding between the layout definition and its corresponding adapter. This is the simple and effective approach of the Application Designer’s development process: The adapter represents a logical abstraction of what the page displays. All layout definitions are kept in the page - all the logic is kept in the adapter. (Or better: behind the adapter. The adapter itself should only be a facade to the "real" application logic.)

Data Exchange at Runtime

What happens at runtime?

- The browser accesses (within the preview area of the Layout Painter) the intelligent HTML page generated from the XML layout description. The HTML page is loaded. Depending on the settings of your browser, the HTML page is picked from the page buffer of the browser after the first access.
- The page is loaded and JavaScript statements are executed to send a command to the server, requesting the data content of the page.
- The server’s session management finds out that a new instance of the adapter class has to be created. It calls the default constructor - without any parameters - of the adapter class and registers it within its session management.
- The server calls all `get` methods of the adapter and collects the results in a streamed string. The string is sent back as a response to the browser.
- The page inside the browser receives the response and distributes the new content to the controls. The controls are updated using JavaScript statements working with the DOM interface.
- The user provides some input, for example, enters the name. The content change is registered inside the page.
- The user does something which causes a flush of the changes (for example, the user chooses a button). Therefore, all registered data changes are packaged as a streamed string and are sent to the server including the information that a method has to be called.
- The server receives the request and finds by its session management the corresponding adapter object. First, it pushes all data changes using the `set` methods into the adapter class and calls the method.
- The method changes the data inside the adapter.
- The server calls all `get` methods of the adapter and collects the result into a streamed string. The string is sent back as a response to the browser.
- And so forth.

With a standard HTTP connection, only the changed content of the screen is passed when operating on one page. The layout is kept stable in the browser. Consequently, there is no flickering of the page due to page reloading.

All steps described in the list above are done completely transparent to your adapter; i.e. you do not have to cope with session management, stream parsing, error management, building up HTML on the server, etc. You just have to provide an intelligent HTML page by defining it in the Layout Painter and an adapter class.

Files and their Locations

Have a look at the files created for your "Hello World!" application and take notice of the directory in which they are located.

All files are located in the directory *<installdir>/tomcat/webapps/cis/cisyourfirstproject*. The */cis* directory is the directory of the web application instance. The */cisyourfirstproject* directory is the directory that has been created for your new project

- The XML layout definition is kept in the */xml* directory.
- The generated HTML page is kept directly in the project directory. There are also some other files inside this directory that start with "ZZZZ". These files are temporary files used when previewing pages inside the Layout Painter.
- The Java class files are kept in the directory */appclasses/classes*. There is also a directory */appclasses/lib* where you can find *jar* files to be imported into your application.
- In the directory */accesspath*, "access restriction" files are generated. If you view these files inside a normal text editor (such as Notepad), you see that one file is maintained for each page; it holds the information about which properties are accessed by the page.

The directory locations described in this section are the default locations. See also *Directories* in the *Development Workplace* documention.

Application Project Management

Use the project "cisyourfirstproject" only for your first steps. For real development, create your own projects. Working inside your own projects, you can be sure that your project files will never be touched by any Application Designer updates. Therefore, patches or enhancements to Application Designer can be installed by just copying them over the existing Application Designer installation.

Keep all your files inside the project directory during the development cycle. Even if this might work correctly: for example, do not put *jar* files into the server (Tomcat) directories, but place them into the corresponding */appclasses/lib* directory of your project.

There are some rare cases in which you have to position *jar* files explicitly inside the Tomcat environment: for example, if classes of a *jar* file access native libraries (such as DLLs) by the JNI interface. They can only be loaded once by the class loader. Because both Tomcat and Application Designer have class loaders which may load a class multiple times, you have to position these classes, for example, into the */tomcat/lib* directory. However, these are the exceptions. Do not put any classes outside of the Application Designer project directories unless there is a reason for it.

Usage of Enhanced Development Tools

You may already have noticed that there are various other tools inside the development workplace. See the *Development Workplace* documentation for detailed information on how to use these tools.