Modular development

In This Chapter

Introduction

With Servoy you can develop your application by using a modular design. This means that you can separate objects from eachother by storing them into different solutions and attaching them to one another. Basically, the idea and advantage of this concept is that it allows you to reuse resources instead of having to define and maintain them more than once. Servoy allows you to reuse:

- Forms
- Variables
- Methods
- Relations
- Value lists
- Media

Solution types

Servoy distinguishes several solutions types of which *solution* and *module* are the most important terms. However, not all of them allow sharing of resources and are meant to be executed standalone.

Type	Meaning	Allows resource sharing
Normal	Can be used and started as a regular solution or attached to another solution and/or module as a module.	Yes
Module	Can only be used as an attached module to a solution and/or module.	Yes
Web client only	Can be used as a regular solution, but can only be deployed into the Web Client or attached to another solution and /or module.	Yes
Smart client only	Can be used as a regular solution, but can only be deployed into the Smart Client or attached to another solution and/or module.	Yes
Login	Can only be as an attached module specifically for the login UI.	Yes
Authenticator	Can only be used as a standalone module which executes security logic server-side.	No
Pre-import hook module	Can only be used for automatically executing business logic before the top solution is imported into the Application Server.	Yes
Post-import hook module	Can only be used for automatically executing business logic after the top solution is imported into the Application Server.	Yes

Note

For information about the solution types *login* and *authenticator*, check out the *Enhanced Security* section.

You can define the solution type by selecting one at property *solutionType* at the property sheet of the active solution.

Solutions and modules

To attach modules to a solution, you can select all applicable modules at property *moduleNames* at the property sheet of the active solution. By doing so you can now expand the *Module* node of the active solution in the Solution Explorer and access all objects of each attached object. You can also maintain objects from modules and copy and move objects between solutions and modules.

Note

Solution properties will be ignored if the solution is accessed as a module. Only settings at the top solutions are applicable, except for property *moduleNames*.

Accessing objects from modules from the active solution works in the same was as accessing them from the active solution itself. The only difference is that in most cases the name of the module is shown behind the name of the objects which can be selected (from a list).

> **Note**
>
> It is not possible to access objects of a solution or module which are not directly or indirectly attached to the current solution or module. Therefore, it's also not possible to access objects from a solution which lies higher in hierarchy than the current one, as long as it has not been attached.

> **Note**
>
> When a module is attached to a solution and that solution has been attached to that same module, then you have created a cycle. This is not allowed and results in an error in the Developer.

> **Note**
>
> The way how your application has been designed by using modules is not relevant anymore during deployment. At this point the collection of solutions and modules have become one flat solution.

Form inheritance

Another way of reusing objects is by form inheritance. Servoy allows you to extend forms by other forms. As forms contain UI as well as variables and methods, all of this can be inherited and thus reused.

> **Note**
>
> For more information about inheritance, check out the *Inheritance* section.

Best practices

Servoy allows you to use its modular design in any way that you want. However, there are some best practices which can help you as a guideline to make better use of it.

Basic core

When you develop multiple projects then there is most likely some functionality you want to use for all your projects which does not depend on any database connection. Think of logic like e.g. math function, date calculations, procedures for sending notifications and perhaps some general icons. This kind of logic is very suitable to store in a core module. Whenever a new project is started, this module will be the main building block on which the rest of your solutions and modules which be based upon.

Project core

Bigger projects are usually developed by using multiple solutions which are executed independently and/or connected by a main solution. It's most likely that these solutions share certain objects like e.g. relations, calculations and project-specific media. These objects can be stored in a project core module and can be attached to all the other project solutions.