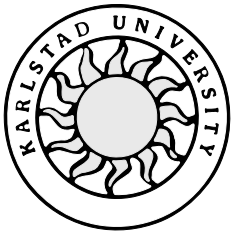


Computer Science

**Magnus Gustafson, Göran Skantz**

# iWarf - a Service Creation Environment

Bachelor's Project

2001:14

# iWarf - a Service Creation Environment

**Magnus Gustafson, Göran Skantz**

This report is submitted in partial fulfilment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Magnus Gustafson, Göran Skantz

Approved, 2001-06-05

Advisor: Hannes Persson

Examiner: Stefan Lindskog

# Abstract

This C-grade Bachelor's project is written for the Department of Computer Science at Karlstad University and for the Karlstad company Incomit, spring term 2001. The background for this work is the need of a certain service creation environment for making development of telecom and Internet services easier and the main goal is to create a prototype for this. Service developers, who will create the telecom services of tomorrow, will use this service creation environment. The environment is based on the modular open source IDE (Integrated Development Environment) NetBeans. New features have been added to the environment to make it easier to create telecom and Internet services. The development has been done mostly in XML (eXtended Mark-up Language) and with the JavaBeans™ components architecture.

# Acknowledgement

We would like to start by thanking our University supervisor, Hannes Persson, for helping us writing this document and pushing us to really start in time. Special thanks goes to our company supervisor, Pär Larsson, together with Marwan Semaan and Håkan Spjuth at Incomit for giving us the support we needed and making this Bachelor's project joyful.

# Contents

# List of Figures

# 1 Introduction

In these days, telecom and Internet are really common things in everyone's life. We use a lot of services and take them for granted. The most common services available today are simply WAP (Wireless Application Protocol), messaging and calling services but the possibilities of tomorrow are far beyond these. What about setting up your own conference calls, receiving information depending on where you are or choosing to call the nearest taxi by just clicking the taxi button on your PDA (Personal Digital Assistant). What we really do not think of is how these services are created. Today at the Karlstad company Incomit, services are created from scratch with a common editor. What happens when Incomit will ship their products and their customers need to create a service? Probably it will take hours to create even a simple one and this is not really a good sell argument. Therefore it would be a lot easier with some sort of service creation environment with certain Incomit features. This Bachelor's project is about creating a prototype for that environment.

**Chapter two** explains the background of this Bachelor's project. In **chapter three** the purposes of this Bachelor's project and the requirements of the software are listed. This Bachelor's project includes a preliminary investigation about choosing an IDE, which can be read in **chapter four**. The prototype design, described in **chapter five**, can be divided into graphical and functional design. **Chapter six** brings up the implementation, which is the main part of this Bachelor's project. Here the reader will be given a thorough description of how to modify the IDE. Furthermore, **chapter seven** contains a use case that shows the prototype usage. Then in **chapter eight**, experiences and recommendations are discussed and then finally in **chapter nine** conclusions are presented. After these chapters follows a section with terminology and abbreviations and then a section with references.

# 2 Background

Incomit AB is a producing company with their main office in Karlstad, Sweden. Incomit is a growing company with approximately 35 employees. Incomit's business concept is to integrate the mobile telephone network and the PSTN (Public Switched Telephone Network) with the Internet. Their two flagship products are iSluice and iSea. iSluice is a server that runs within the PSTN to give iSea servers access to the telecom network. iSea can be classified as an application server that can execute different types of services, such as conference call or positioning mobile phones, and in a safe way communicate with iSluice. As seen in Figure 1, it is possible to communicate with iSea using WAP or HTTP (Hyper Text Transfer Protocol) through a portal.

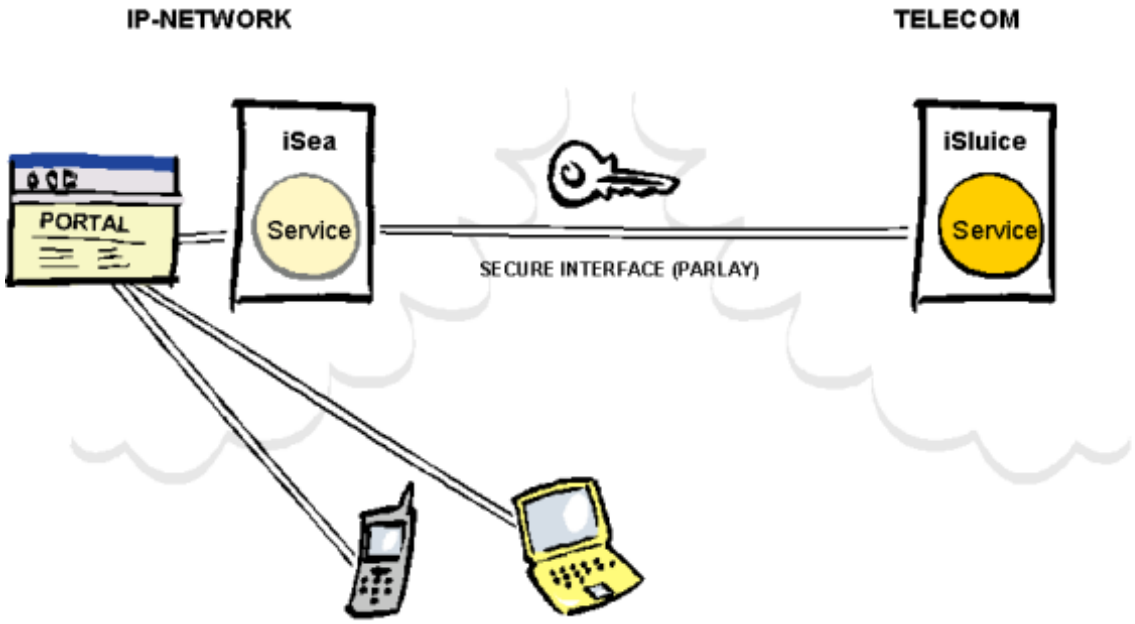


*Figure 1: iSea and iSluice overview*

## 2.1 iSea

iSea is a platform that enables service developers to quickly create advanced telecom-based services and make them accessible in an Internet environment. iSea consists of many different components, both hardware and software. Incomit develops some of the components and some are bought and integrated into iSea. This way of working is possible thanks to products built on open systems and commercial standardized interfaces.

## 2.2 iSluice

iSluice is the “sluice” an operator must have in the network to permit secure and reliable routing of externally initiated iSea services (as from Internet portals). iSluice enables today’s operators to increase the usage of their networks by attracting Internet portals and other companies to develop services. iSluice allows iSea servers to access the telecom network in a secure way.

## 2.3 SLEE

The SLEE (Service Logic Execution Environment) is an execution environment. Both iSea and iSluice are based on a SLEE. The difference between iSea and iSluice is the services running in the SLEE. For more information see the SLEE API specification [5].

## 2.4 Problem

Today all iSea services are developed in Java™. This works fine but it demands a lot of competence, both in iSea API's and Java™. An important issue for making profit from iSea and iSluice is to offer Incomit’s customers a tool that has features to simplify the making of iSea services. How do one measure the simplicity of a tool? This is a problem, because it is difficult to measure simplicity. An important factor is the time it takes for a developer to make a service from scratch to have it running in iSea. Our assignment was to make a prototype for such a tool. The prototype was based on an existing development tool, which was changed to fulfil Incomit’s needs better.

# 3 Purpose and requirements

Below follows the purpose of this Bachelor's project and the requirements specific to the prototype. The prototype will be the foundation of a commercial product that is to be developed the summer of 2001.

## 3.1 Purpose

The main purpose of this Bachelor's project is to create a service creation environment, called iWarf, which makes it easier to develop telecom and Internet services. This means that it will be possible to create services faster than without the environment, which might be the small difference between coming in first or second place on the market because of the exaggerated tempo. It also means that fundamental knowledge of the Java™ language will still be necessary but not all about the underlying structure and this will simplify the developing.

## 3.2 Requirements

The requirements for this Bachelor's project are gathered in the Requirements specification in Appendix A. For more specific information, please read that Appendix.

### 3.2.1 iWarf requirements

Below follows the requirements specific to iWarf.

1. Create an Incomit service project
2. Create a specific Incomit servlet project
3. Create a specific Incomit client project
4. Investigate which development tool to use
5. Create an ESPA (Easy Service Provider API) palette.
6. Create an Utils palette.
7. Drag 'n' drop Incomit iSea Call Control service
8. Drag 'n 'drop Incomit iSea Messaging service
9. Drag 'n' drop Incomit iSea Positioning service
10. Drag 'n' drop Incomit iSea DB service
11. Drag 'n' drop Incomit iSea Log service
12. Drag 'n' drop Incomit iSea Trace service

13. Incomit menu
14. JAR (Java™ ARchive) Packager
15. Deploy to iSea
16. Help files
17. Create a service using iWarf

### 3.2.2 Other requirements

Some documents had to be written to accomplish this Bachelor's project and these are:

- Requirements specification.
- Technical documentation. A SDS (System Design Specification) containing information specific for the Incomit crew.

There were no delimitations of this Bachelor's project other than the requirements specified above or in the requirements specification.

# 4 Preliminary investigation about choosing an IDE

The recommendation from the company supervisor was that Sun's Forte™ should be used as underlying foundation when creating the service creation environment. We first built iWarf upon Forte™. When problems occurred no useful information about Forte™ could be found but since Forte™ is based on the NetBeans IDE, everything needed could be found at NetBeans website [4]. When working further with Forte™ and NetBeans, reason came up to question the choice of underlying IDE. Was Forte™ really the better of the two? To answer this question, a comparison between the tools had to be done. The differences between NetBeans and Forte™ are very small. Their GUI (Graphical User Interface) looks almost the same, so the user will not know on which IDE iWarf would be based. An interesting question was found at Forte's FAQ website [1]:

*"What's the difference between Forte for Java software and NetBeans software?"*

*"The NetBeans Tools Platform supports a large community of developers centered at netbeans.org. NetBeans software is primarily a platform for developers (including the Sun Forte Tools group), who want build to development tools. Each version of Forte for Java software is Sun's tested, supported, commercial implementation of the open standard NetBeans Tools Platform. The Forte for Java product also includes optional, 'closed-source' modules that are offered for a price."*

The most interesting sentence from that answer is, *NetBeans software is primarily a platform for developers (including the Sun Forte Tools group), who want to build development tools* (grammatically corrected by author of this document). This is exactly what is to be done, so NetBeans might be a better choice. To make this decision on stronger grounds, more facts had to be found. When enough facts was found, they were listed as positive/negative attributes, which can be seen below.

Forte

+ Tested by Sun.
+ Support by Sun

+ CE (Community Edition) version is free but other versions are very expensive.
- Current version 2.0 does not include layers (easy creating menus and palettes, using XML) and wizards.
- Forte is based on NetBeans, and because of that, improvements will be made to NetBeans before they are put into Forte.

NetBeans

+ A platform for developers, who want to build development tools.
+ Core is updated more frequently as seen in Figure 2 below.
+ Absolutely free.
+ Current version 3.1 includes layers and wizards
+ The idea with NetBeans is to modify it to fit your needs.
+ Mailing lists (frequently used).
- No support.

This conclusion was made:

Building an Incomit service creation environment based directly on NetBeans seems like a much better idea because the purpose with NetBeans is to modify the environment to fit the user's needs. Forte is based on NetBeans and that means that iWarf indirectly will be based on NetBeans anyway. Below is a figure of a roadmap that shows that new versions of NetBeans are released more frequently than Forte™.

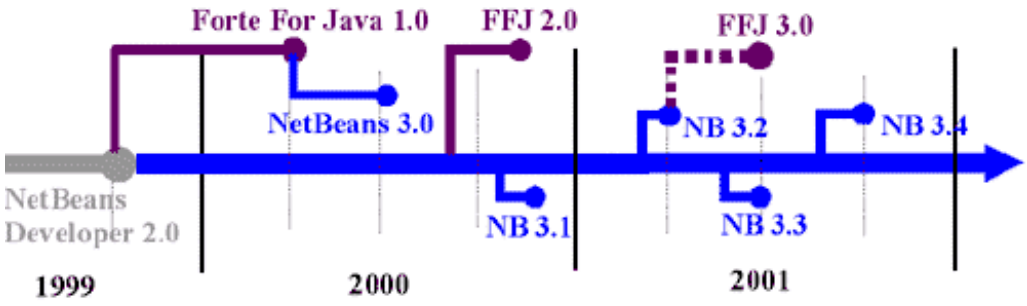


*Figure 2: NetBeans and Forte roadmap*

# 5 Description of the system design

The system design can be divided into graphical design and functional design. The graphical design is what the end user will use as an interface to communicate with the functional design. Another way of describing it: The graphical design is what is shown on screen. Behind the surface is the functional design. The functional design describes what action should be performed when a graphical object is activated.

## 5.1 Graphical User Interface design

The graphical user interface was built on the already existing IDE, NetBeans (version 3.2). This is what NetBeans GUI looks like:

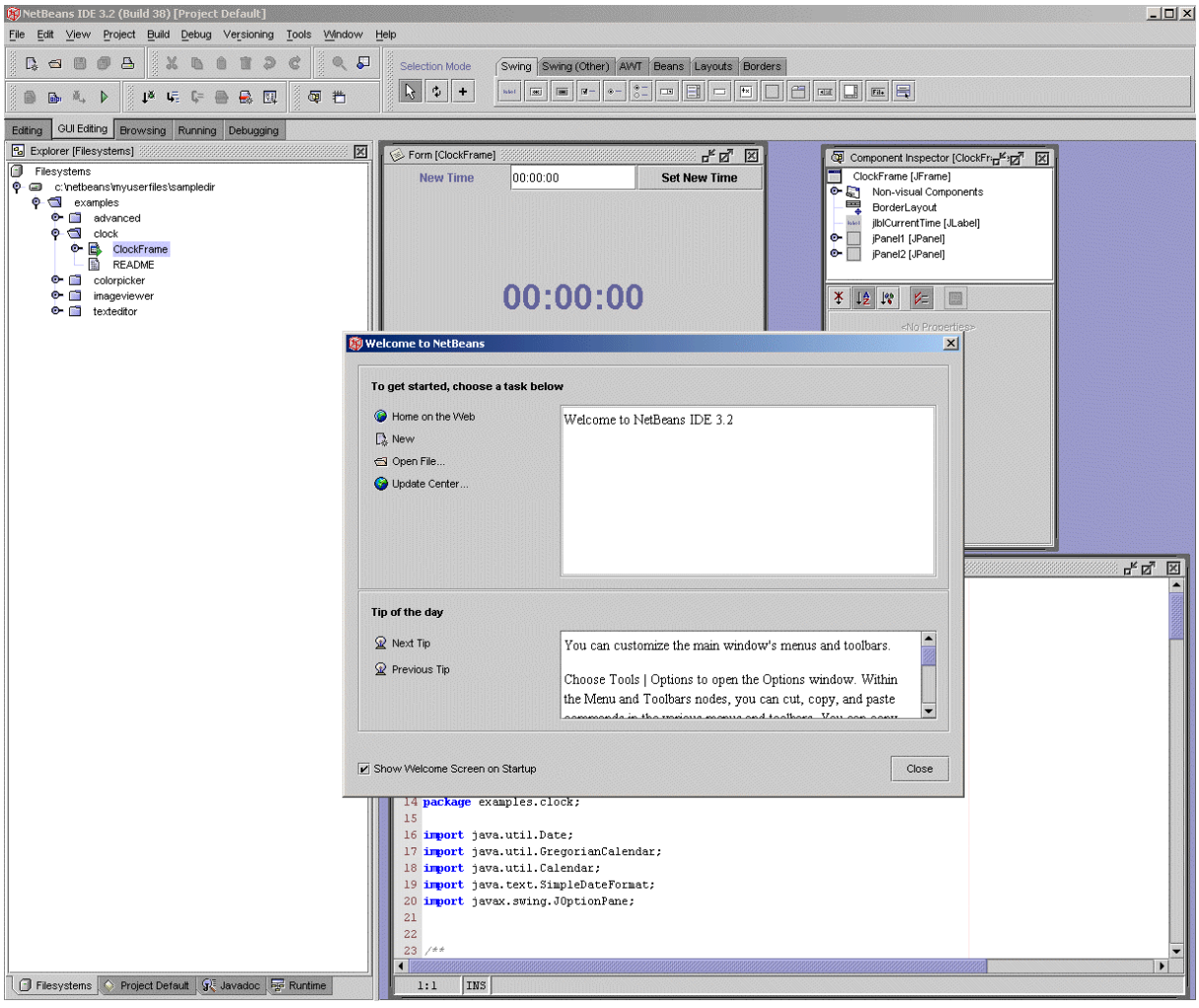


*Figure 3: NetBeans GUI*

### 5.1.1 Incomit menu

In the top menu bar a menu with the label, “Incomit” will be inserted (see Figure 4). This menu will include menu items in the following order:

- “Descriptor wizard”
- “Deploy to iSea”
- A line that separates the menu items (separator)
- “Incomit web”

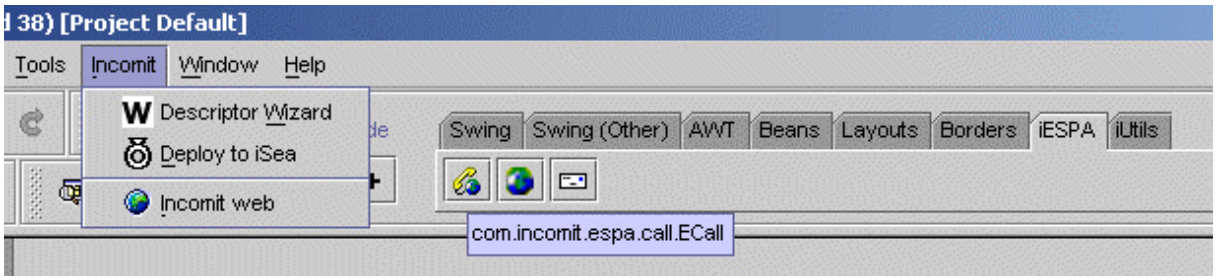


*Figure 4: iWarf – Menu and palette*

### 5.1.2 Palettes

Two palettes, “iESPA” and “iUtils” (see Figure 4), will be inserted beside the already existing ones. These palettes consist of iSea services that can be dragged and dropped to the current project. The services will be displayed as an icon and a tool tip text. The services will be explained in chapter 5.2.2.

The iUtils will include the following services:

- “Db”
- “Log”
- “Trace”

The iESPA will include:

- “ECall”
- “Positioning”
- “Messaging”

### 5.1.3 Templates

The specific Incomit templates will show up when creating a new project file by choosing “File->New”. All templates will be placed in a sub directory called “Incomit”.

The following templates will be available:

- “iSea SLEE Client”

- “iSea SLEE Service”
- “iSea non-SLEE Client”
- “iSea non-SLEE Service”

### 5.1.4 Other graphical design parts

Below follows other graphical design parts.

- The text in the title bar of the IDE will be changed to “Incomit iWarf [version]”.
- The start up screen will be changed to the Incomit iWarf picture seen below.


*Figure 5: iWarf splash screen*

## 5.2 Functional design

Each graphical object represents an underlying functional task. Those functional tasks are explained in this chapter.

### 5.2.1 Incomit menu

- **“Descriptor wizard”**

  When creating an iSea service, a descriptor file is needed. This file, `srv_depl.xml` contains attributes of the service and is needed by the SLEE. Since these are SLEE specific attributes, and not directly part of the iWarf prototype, they will be briefly described in Appendix C. A wizard is needed to make it easy for the user to write this file. By filling out fields, the user will go through this wizard with four steps (also see chapter 7.5).

- **“Deploy to iSea”**

  This will start up the Incomit Deployment tool that lets the user install a service (a JAR-file) into iSea. A slightly adjustment will be made to the tool, so it will fit better into iWarf.
- **“Incomit web”**

  When activated the Ice browser shows the Incomit website online. The Ice browser is a web browser integrated in NetBeans.

### 5.2.2 Palettes

- **“Db”, ”Log”, ”Trace”, “Messaging” and “Positioning”**

  These will not be fully implemented. The main reason for this is that they are very similar to `ECall` (ESPA call control). So more effort will be put in making that service well implemented and documented. Another reason is that the ESPA and the Utils API’s, which are used by all ESPA and Utils services, are not yet implemented.
- **ECall**

  The `ECall` API is defined, and that is what will be used to write a dummy class. The `ECall` dummy class will only do printouts as seen in Appendix B.2. For example the `createCall(String number)`, will make the printout, `“createCall: [the number string]”`. When an `ECall` object is dropped onto the form, two lines of code will be generated:

```
1. eCall1 = new com.incomit.espa.call.ECall();

2. private com.incomit.espa.call.ECall eCall1;
```

# 6 Implementation – NetBeans development

This chapter illustrates how a developer can add extra features to the NetBeans IDE. The information below is based on NetBeans 3.2 and might change in later releases. Some of the features from chapter 5 were too complicated to explain in this chapter, therefore some features were replaced by more trivial examples.

## 6.1 NetBeans architecture

NetBeans is an open source, modular, standard-based IDE, totally written in Java™. It is based on different modules and this makes it really easy to modify to suit your needs. To add features to the IDE, creation of modules can be done (see chapter 6.2). To simplify the modification of the IDE, it is based on a virtual file system, in which files can be added just by writing some XML.

## 6.2 Modules

It is possible to create every feature directly within the NetBeans IDE, but this is not the right way of working if the features would be distributed to someone else. To do the later, a module that contains the features has to be created. There are no restrictions of how many features that can be added to one module, so there is no need for more than one module when customizing the IDE. The module will be packed into a JAR-file, called `mymodule.jar`, and then inserted into NetBeans.

### 6.2.1 The manifest file

The first thing to do when creating a module is to write a manifest file. This simple text file specifies the version and name of the module and the path to the layer file (see chapter 6.2.2). There are no restrictions to the manifest filename but the manifest file will be named `Manifest.mf` and placed in the folder `[JAR-file root]/meta-inf/` when packing the JAR (see chapter 6.2.3). Below is an example of how a module manifest might look like.

```
1.  Manifest-Version: 1.0
2.  OpenIDE-Module: com.incomit.iwarf.modules
3.  OpenIDE-Module-Name: Incomit iWarf module
```

```
4.  OpenIDE-Module-Specification-Version: 0.1
5.  OpenIDE-Module-Implementation-Version: (xmagu/xgosk build 0001)
6.  OpenIDE-Module-Layer: com/incomit/iwarf/modules/Layer.xml
7.  Created-By: Incomit iWarf crew
```

| Line | Attribute | Description |
|---|---|---|
| 1. | *Manifest-Version* | Should always be 1.0. |
| 2. | *OpenIDE-Module* | Specifies the path (within the JAR-file) to the directory that contains the layer file and all other files in the module. Separate directories with dots. |
| 3. | *OpenIDE-Module-Name* | This is the name of the module and will appear in Tools->Options in NetBeans. |
| 4. | *OpenIDE-Module-Specification-Version* | The current specification version of the module. |
| 5. | *OpenIDE-Module-Implementation-Version* | The current implementation version of the module. Here the current build of the module can be specified. |
| 6. | *OpenIDE-Module-Layer* | Path to the layer file. Separate directories with slashes. |
| 7. | *Created-By* | User or company that has created this module. |

### 6.2.2 Layer file

The layer file is the heart of the module. It contains a lot of information describing features that will be inserted into the IDE. With the layer file, a lot of code writing will not be necessary, instead a few lines of XML will be enough. The layer filename is often set to `Layer.xml`, but has actually no restrictions. Below is an example of a simple layer file that creates a new empty menu and places it between the `Window` and `Help` menus in NetBeans, as seen in Figure 6.

```
1.  <filesystem>
2.     <folder name="Menu">
3.        <attr name="Window/MySubFolder" boolvalue="true"/>
4.        <folder name="MySubFolder">
5.        </folder>
6.        <attr name="MySubFolder/Help" boolvalue="true"/>
7.     </folder>
8.  </filesystem>
```

| Line | Description |
|---|---|
| 1. | The filesystem tag should always be put on the first line of the layer file. |
| 2. | Specifies the virtual folder changes are made to, in this case the `Menu` (the menu bar). |
| 3. | Forces NetBeans to place the `MySubFolder` menu after the Window menu. |
| 4. | Creates a new menu named `MySubFolder`. |
| 5. | Ends the folder tag specified in line 4. |
| 6. | Forces NetBeans to place the `MySubFolder` menu before the `Help` menu. |
| 7. | Ends the folder tag specified in line 2. |
| 8. | Ends the filesystem tag specified in line 1. Always at the last line in the layer file. |

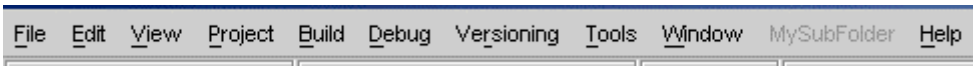


*Figure 6: Result from the layer file*

### 6.2.3 Installation

First the directories, specified in the manifest file, has to be created. In this case something like this (note that the `Netbeans` directory in the figure below has nothing to do with the NetBeans installation directory where the program files are stored):


*Figure 7: Setting up module directories*

Put the manifest file created in chapter 6.2.1 into the `manifests` directory and name it `module-mf.txt`. Then save the layer file, from chapter 6.2.2, as `Layer.xml` into the `code\com\incomit\modules` directory. Now enter the `code` directory and type the following to pack the JAR-file:

```
jar cvfm0 mymodule.jar ../manifests/module-mf.txt com/incomit/iwarf/modules
```

Then simply move the `mymodule.jar` file into the `[NetBeans installation directory]\modules` directory and start NetBeans. The changes will now be applied automatically. This makes it really easy to distribute the changes to someone else, by just copying the file. In later chapters, nothing is said about recompiling the JAR-file, when a new feature will be added. That is taken for granted.

## 6.3 Templates

This chapter describes how to create a template and then put it into the module created in the previous chapter. Templates can be created with or without group files. When using a group file one or more files can be added to the template, without a group file only one single file can be added. Because it is a lot easier to add another file to a template based on a group file, non-group file based templates will not be discussed here. The following example is the iSea SLEE Client template described in chapter 7.2.

### 6.3.1 Template files

Start with creating some new directories to store the template files in (see figure below).


*Figure 8: Setting up template directories*

These directories are not necessary but this is done for making it look nice and easy. Then three files are needed in the `iSeaClient` directory to be able to create the template.

**iSeaClient_java**

This file includes the code that is going to be static when creating an iSea SLEE Client. Macros can be used in this file to get the current date and time, the Microsoft Windows username and the filename. The source can be found in Appendix B.1. Since this file is not going to be compiled call it `iSeaClient_java` (this will not mix it up with `.java` files). To be able to compile the Java™ source file that is generated from this file in NetBeans,

installation of the `com.incomit.slee` class library is required. How to install external class libraries is explained in chapter 6.3.5.

**iSeaClient_form**

This file is used for getting a form in the NetBeans IDE that enables components to be added by dragging ‘n’ dropping. This file will always have the appearance as described below:

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.
3.  <Form version="1.0"
    type="org.netbeans.modules.form.forminfo.FrameFormInfo">
4.  </Form>
```

**iSeaClient_group**

The last file is the group file that contains the path to all files that are to be included in the template. The path is the one found in the layer file as virtual folders (see chapter 6.3.2 for further details). The files that will be included are the two files above. This is how the file looks like:

```
1.  Templates/Incomit/SLEE/__NAME_sleeClient__.java
2.  Templates/Incomit/SLEE/__NAME_sleeClient__.form
```

### 6.3.2 Layer file – create the template

Now some text has to be inserted into the layer file to really create the template in NetBeans. Put this in the `Layer.xml` below the `<filesystem>` tag (see chapter 6.2.2):

```
1.  <folder name="Templates">
2.     <folder name="Incomit">
3.        <attr name="templateWizardURL" urlvalue="nbresloc:/com/incomit/
    iwarf/modules/templates/resources/incomit.html"/>
4.        <folder name="SLEE">
5.           <attr name="templateWizardURL" urlvalue="nbresloc:/com/
    incomit/iwarf/modules/templates/resources/slee.html"/>
6.           <file name="iSeaClient.group"
    url="templates/slee/iSeaClient/iSeaClient_group">
7.              <attr name="template" boolvalue="true" />
```

```
8.              <attr name="SystemFileSystem.localizingBundle"
    stringvalue="com.incomit.iwarf.modules.templates.Bundle" />
9.              <attr name="templateWizardURL" urlvalue="nbresloc:/com/
    incomit/iwarf/modules/templates/resources/iSC.html"/>
10.             <attr name="SystemFileSystem.icon"  urlvalue="nbresloc:
    /com/incomit/iwarf/modules/templates/resources/iSC.gif"/>
11.         </file>
12.         <file name="__NAME_sleeClient__.java"
    url="templates/slee/iSeaClient/iSeaClient_java">
13.             <attr name="template" boolvalue="false" />
14.         </file>
15.         <file name="__NAME_sleeClient__.form"
    url="templates/slee/iSeaClient/iSeaClient_form">
16.             <attr name="template" boolvalue="false"/>
17.         </file>
18.       </folder>
19.    </folder>
20. </folder>
```

| Line | Description |
|---|---|
| 1. | Specifies the virtual folder in where to make the change, in this case `Templates`. This is the first entry in the group file path (see the `iSeaClient_group` file in chapter 6.3.1). |
| 2. | Creates a new folder named `Incomit`. This is the second entry in the group file path (see the `iSeaClient_group` file in chapter 6.3.1). |
| 3, 5, 9. | Sets the paths to the help files displayed when selecting the `Incomit` directory (3), `SLEE` directory (5) or the iSeaClient template (9). See chapter 6.3.3. |
| 4. | Creates a new folder named `SLEE`. This is the third entry in the group file path (see the `iSeaClient_group` file in chapter 6.3.1). |
| 6-11. | Adds the group file to the template. Use dot instead of underscore in the name so that NetBeans associates right. URL is relative to the layer file. |
| 7. | Tells NetBeans that this file is a template. |
| 8. | Path to the bundle file (`Bundle.properties`). See chapter 6.3.4. |
| 10. | The icon displayed in NetBeans. See chapter 6.3.3. |
| 12-14. | Adds the Java™ file to the template. Use dot instead of underscore in the name. |

| | |
|---|---|
| 13, 16. | Tells NetBeans that this file is NOT a template. |
| 15-17. | Adds the form file to the template. Use dot instead of underscore in the name. |

### 6.3.3 Resource files

Resource files are html files and gif files that represent help files and icons that appear at different places. Icons have the size of 16x16 pixels and can have transparent background. The help files consist of common html code.

Create the help files `incomit.html`, `slee.html`, `iSC.html` and the icon `iSC.gif` and put them into the `resource` directory to get the template work properly.

### 6.3.4 Bundle file

Bundle files are very common in the NetBeans IDE. They provide an easy way of changing names, paths or whatever that can be written in text. This makes it easier to change a path without changing the code. Just change the text in the bundle, re-pack the JAR and the changes are done.

The bundle has the filename `Bundle.properties` but the path to it, in the layer file, is in this case `com.incomit.iwarf.modules.templates.Bundle`, without the `.properties`. Create a `Bundle.properties`, put it in the `templates` directory and insert the following text:

```
1.  Templates/Incomit/SLEE/iSeaClient.group=iSea SLEE Client
```

The path above is based on the virtual folders and file, created in the layer file (see chapter 6.3.2). iSea SLEE Client is the name that will appear in NetBeans, when selecting the template.

### 6.3.5 Adding external libraries to NetBeans

If there exists an `xwing.class` in the class library `starwars.jar`, and the xwing class is needed by the Star Wars GUI you are currently working on, then the `starwars.jar` needs to be added to the NetBeans class library. To achieve this, start by copying `starwars.jar` into the `[NetBeans installation directory]\lib\ext` directory. After that add the code

below to the layer file. If the `starwars.jar` class library is not added to the NetBeans class library, a class not found error would occur during compilation.

```
1.  <folder name="Mount">
2.      <folder name="java">
3.          <file name="starwars.xml">
4.              <![CDATA[
5.                  <!DOCTYPE JavaLibrary PUBLIC "-
    //NetBeans IDE//DTD JavaLibrary//EN"
    "http://www.netbeans.org/dtds/JavaLibrary-1_0.dtd">
6.                  <Library>
7.                      <Archive
    name="lib/ext/starwars.jar"/>
8.                  </Library>
9.              ]]>
10.         </file>
11.     </folder>
12. </folder>
```

| Line | Description |
|---|---|
| 1. | Specifies the virtual folder in where to make changes, in this case `Mount`. |
| 2. | The virtual folder for Java™ classes. |
| 3. | Here a virtual XML file that includes data about the library is created. Choose a filename, but it is a good idea to have the same name on the XML as on the JAR. |
| 4. | Here is the start of the data in the virtual XML file. |
| 5. | Always begin the data with this line. |
| 6-8. | Here is the library added to the NetBeans class library. The archive path is relative to the NetBeans installation directory. |
| 9. | Ends data. |

## 6.4 Palette – create the drag ‘n’ drop feature

This chapter describes how to create a palette tab and put a JavaBean component, ECall, into it that can be dragged ‘n’ dropped into a form window (see chapter 7.3).

### 6.4.1 Layer file – create the palette

To create the palette a few lines have to be added to the layer file in the module. Just add the following lines after the `<filesystem>` tag (see chapter 6.2.2).

```
1.  <folder name="Palette">
2.      <folder name="iESPA">
3.          <file name="com-incomit-espa-call-ECall.instance"/>
4.      </folder>
5.  </folder>
```

| Line | Description |
|---|---|
| 1. | Specifies the virtual folder in where to make changes, in this case `Palette`. |
| 2. | Creates a new folder named `iESPA`. |
| 3. | Creates an instance of the component class `ECall` in the package `com.incomit.espa.call`. This is the component that can be dragged 'n' dropped into the form. See chapter 6.4.3 for further details. |

### 6.4.2 The manifest file – specifying the palette component

A new manifest file has to be written for the components. This is because of the need for a new JAR-file to put the component(s) in. The component class file should not be included into the module but instead as an external library (see chapter 6.3.5). Name it `palette-mf.txt` and put it in the `manifests` directory. It looks like this:

```
1.  Manifest-Version: 1.0
2.  Created-By: Incomit iWarf crew
3.
4.  Name: com/incomit/espa/call/ECall.class
5.  Java-Bean: True
```

### 6.4.3 The component

Now the component, that was mentioned in chapter 6.4.1, has to be created. The component consists of a common JavaBean class and a bean info class. The bean info class consists of information about the common class, like properties and methods but also the path to the icon shown in the palette. The easiest way to do this is to use NetBeans to create the bean info class, but first the common class.

**JavaBean class**

First the two new directories introduced in line 3 in chapter 6.4.1, have to be created. The `com\incomit` already exists so only the `espa` and the `call` directories have to be created. A `resources` directory can also be added, for help files and icons.


*Figure 9: Setting up component directories*

Now the JavaBean class (`ECall.java`) has to be created and put in the `call` directory. It is a common class that only does printouts. The source can be found in Appendix B.2.

**Bean info class**

Now create the bean info class in NetBeans. Start by making a 16x16 pixels icon (gif-file) and put it in the `resource` directory. This icon will later be seen in the palette. Then open the `ECall.java` file in NetBeans by selecting File->Open File… Browse to the `Ecall.java` file and hit the accept button. The file will now be mounted in the Explorer view. Expand all directories if necessary until you can access `Bean Patterns` seen in Figure 10.

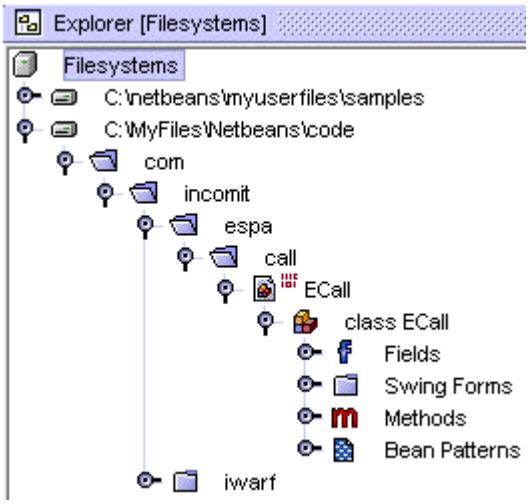


*Figure 10: Explorer view*

Now right-click `Bean Patterns` and choose `BeanInfo Editor…` (see Figure 11).

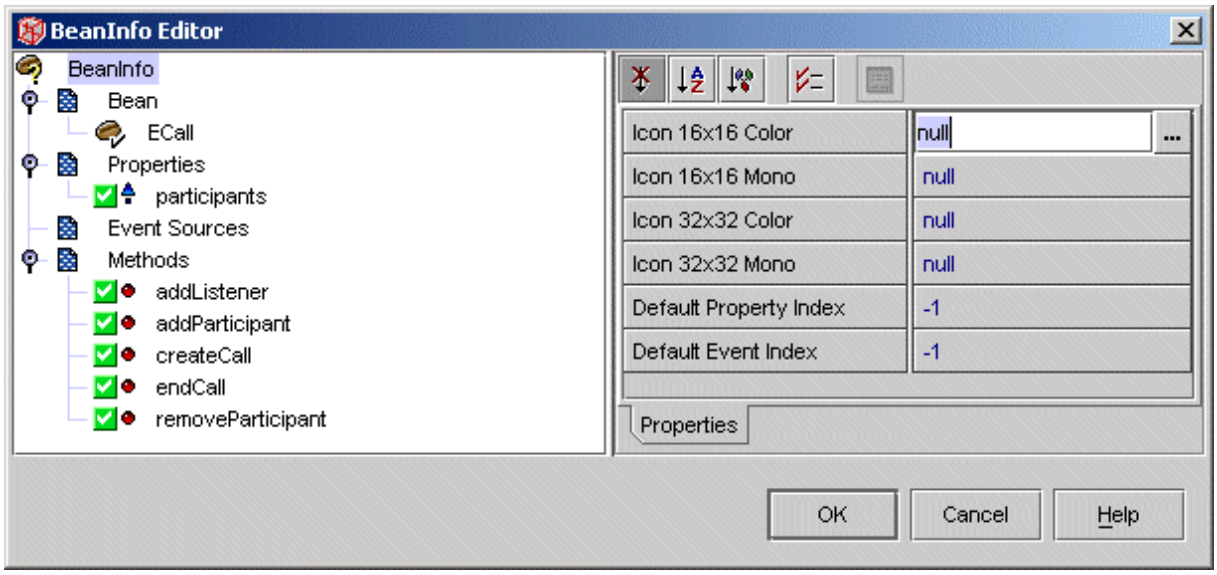


*Figure 11: BeanInfo editor*

Then click the `BeanInfo` and browse for the `Icon 16x16 Color`. Select `Classpath` and browse for the gif file. Hit ok a few times. Now the `ECallBeanInfo` source appears in NetBeans. Save the file and exit NetBeans.

### 6.4.4 Final installation

Enter the `code` directory, compile the `.java` files and pack the JAR-file like this:

```
javac -deprecation -g com/incomit/espa/call/*.java
jar cvfm0 incomitbeans.jar ../manifests/palette-mf.txt com/incomit/espa
```

Move the JAR-file into the `[NetBeans installation directory]\lib\ext` directory. Now the library (JAR-file) has to be added into NetBeans, exactly as in chapter 6.3.5. Add the following in the `Mount/java` folders in the layer file.

```
1.  <file name="incomitbeans.xml">
2.      <![CDATA[
3.          <!DOCTYPE JavaLibrary PUBLIC "-//NetBeans IDE//DTD
    JavaLibrary//EN" "http://www.netbeans.org/dtds/JavaLibrary-1_0.dtd">
4.          <Library>
5.              <Archive name="lib/ext/incomitbeans.jar"/>
6.          </Library>
7.      ]]>
```

```
8.   </file>
```

Now the `ECall` component can be dragged 'n' dropped into the form window.

## 6.5 Wizards

This chapter describes how to create a simple wizard in NetBeans. Wizards can be used for making difficult tasks easier for the user. In NetBeans there exist some help classes for building wizards that makes it a little easier. In this tutorial the two-step Hello World wizard seen below, will be created.

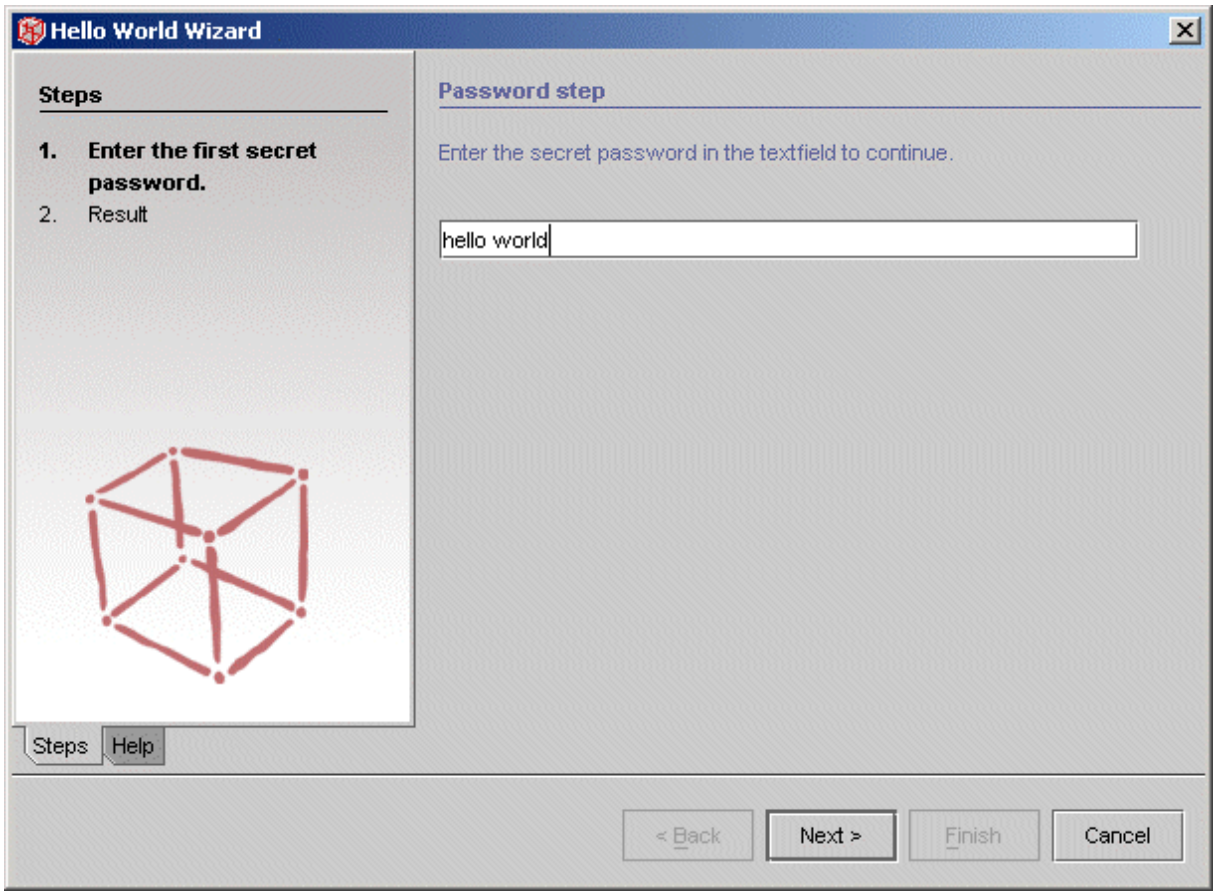


*Figure 12: The complete step 1 in the wizard*

### 6.5.1 Wizard files

Start by creating a `wizards` directory in the `modules` directory. In that directory create the two directories `helloworld` and `resources` (see Figure 13).

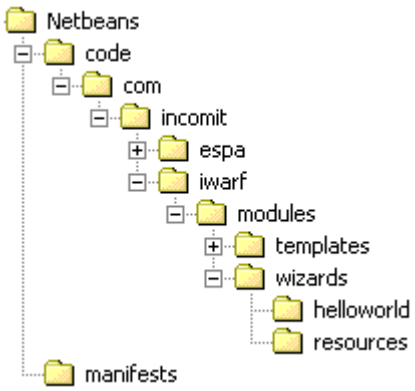


*Figure 13: Wizard directories*

A wizard consists of a main class and one panel class for each step, in this case two. Each panel class extends JPanel on which swing components or common awt components can be added. It also implements an interface to handle common wizard features. The main class manages the panels and all communication is controlled from this class.

**HelloWorldPanel1.java**

Figure 12 illustrates the first step that will appear to the user. HelloWorldPanel1.java is the panel seen in Figure 14. In this step the user has to enter a secret password in the text field. When this is done, the next button (see Figure 12) will be enabled.

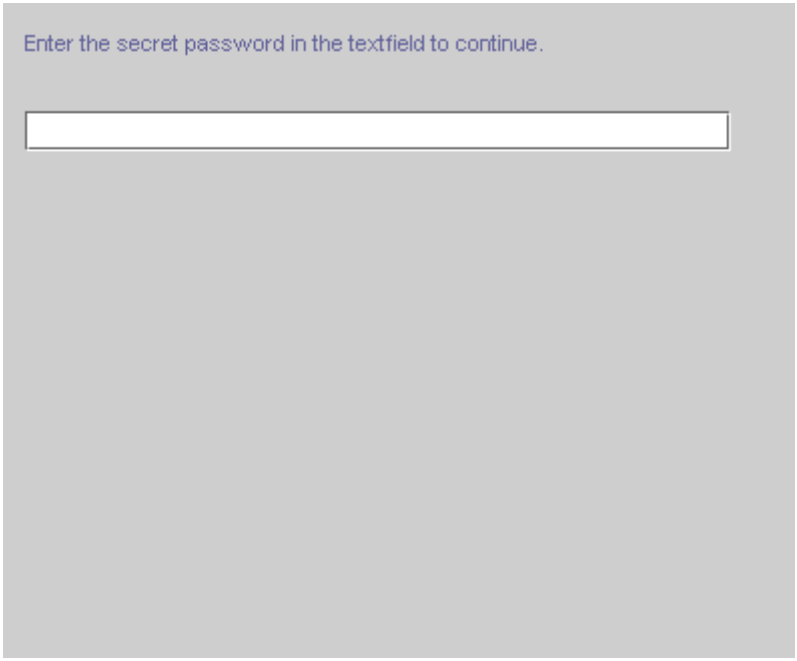


*Figure 14: Wizard Panel 1*

Create the first panel by saving the file `HelloWorldPanel1.java` (see Appendix B.3) in the `helloworld` directory.

**HelloWorldPanel2.java**

This is the second step that will appear when the user has entered the correct password and hit next in the first step. This panel contains a label with the text, “Congratulations!!! You have entered the secret password!!!". In this step the user can either go back or click Finish to close the wizard. This class is much simpler than the `HelloWorldPanel1` class. For example there are no listener or event handlers. Put the source in the file `HelloWorldPanel2.java` (see Appendix B.4) in the `helloworld` directory.

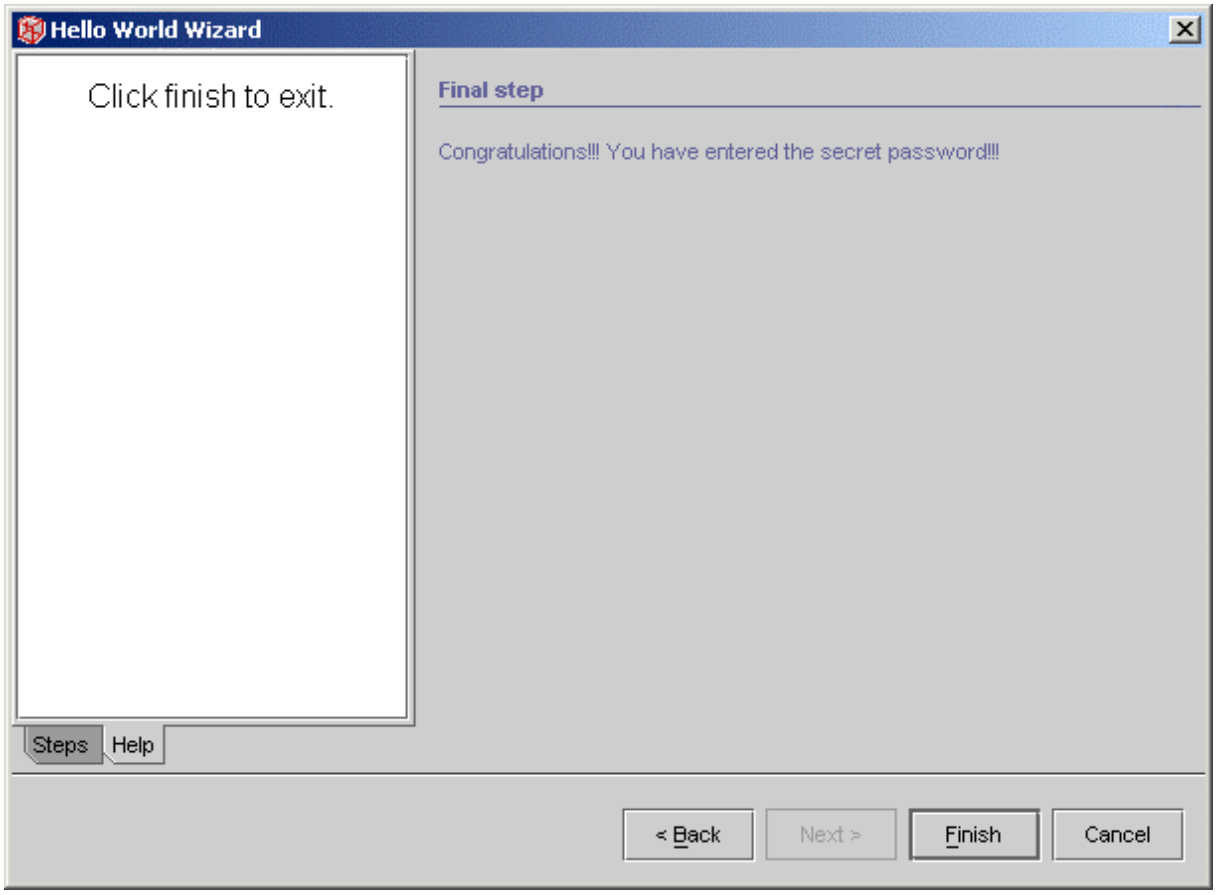


*Figure 15: Complete step 2 in the wizard*

**HelloWorld.java**

This is the main class where communication with the panels can be handled. For example, buttons can be set enable or disable, texts in the wizard can be set and much more. The class includes a `WizardDescriptor` class that is a basic wizard GUI system. The panels will then be added to the `WizardDescriptor`. Simply put the source in the file `HelloWorld.java` (see Appendix B.5) in the `helloworld` directory.

**helloworld_panel1.html**

This is the file specified in the help file URL in the `HelloWorldPanel1` class. Put it in the `resources` directory.

```
1.  <HTML>
2.  <BODY>
3.       <CENTER>Try typing 'hello world' in the textfield.</CENTER>
4.  </BODY>
5.  </HTML>
```

**helloworld_panel2.html**

This is the file specified in the help file URL in the `HelloWorldPanel2` class. Put it in the `resources` directory.

```
1.  <HTML>
2.  <BODY>
3.       <CENTER>Click finish to exit.</CENTER>
4.  </BODY>
5.  </HTML>
```

Now the Java™ files have to be compiled. Add all files to the JAR-file and it is done. To be able to start this wizard from a menu, an action has to be created. This will be done in chapter 6.6.1.

## 6.6 Menus and actions

This chapter describes how to create menus and actions. Menus make it easy for the user to reach specific tasks. An action is called when the user clicks on a menu item in a menu. Create a `menus` directory in `modules` and `incomit` and `resources` in the `menus` directory (see Figure 16).

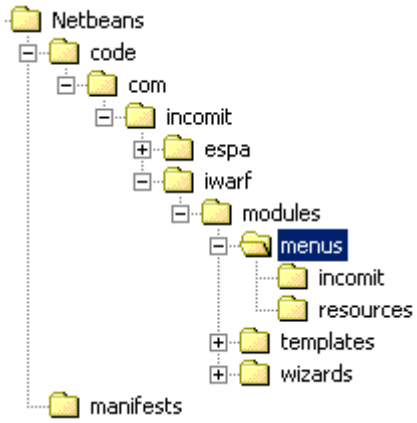


*Figure 16: Menus and actions directories*

### 6.6.1 Actions

There are different types of actions where the `CallableSystemAction` is the most suitable for menu actions. To be able to start the wizard, created in chapter 6.5, from a menu item, an action class is needed.

**HelloWorldWizardAction.java**

This is the action class for the wizard. Its three main purposes are:

- Fetch the menu name associated with the wizard.
- Fetch the icon associated with the wizard.
- Instanciate the HelloWorld class.

Put the `HelloWorldWizardAction.java` (see Appendix B.6) file in the `incomit` directory.

**Bundle.properties**

In the bundle file the name of the menu item can be specified. The & sign in row 2 below means that the letter after it will be underscored in the menu. Hello World Wizard is the name of the menu item. Save this file in the `incomit` directory.

```
1.  # MenuActions
2.  LBL_HelloWorldWizardAction=&Hello World Wizard
```

### 6.6.2 Menus

To a menu, either action classes or URLs can be added. To finally be able to start the wizard from the menu a few things have to be done.

**The layer file – create the menu**

As was done in chapter 6.2.2, a few lines have to be added to the layer file to create a menu. Add the lines below to the layer file. Put it after `<folder name="Menu">` (see chapter 6.2.2).

```
1.  <attr name="Tools/Incomit" boolvalue="true"/>
2.  <folder name="Incomit">
3.       <attr name="SystemFileSystem.localizingBundle"
    stringvalue="com.incomit.iwarf.modules.menus.incomit.Bundle"/>
4.       <file name="com-incomit-iwarf-modules-menus-incomit-
    HelloWorldWizardAction.instance"/>
5.  </folder>
```

| Line | Description |
|---|---|
| 1. | Puts the `Incomit` directory after the `Tools` menu. |
| 2. | Creates the `Incomit` menu. |
| 3. | Specifies the path to the bundle file, in this case used for specifying which letter to underscore in the menu name. |
| 4. | Path to the action class created in chapter 6.6.1. Put `.instance` at the end of the path instead of `.class`. |

**Bundle.properties**

To specify which letter to underscore, a line has to be added into the bundle file. Add the following lines to the bundle file in the `incomit` directory where line 2 specifies that the I letter will be underscored.

```
1.  # Menu keys
2.  Menu/Incomit=&Incomit
```

**Creating an URL in a menu**

Start by adding the following lines after the `<file name="com-incomit-iwarf-modules-menus-incomit-HelloWorldWizardAction.instance"/>` attribute in the layer file.

```
1.  <file name="incomit-web-link.url">
2.       <![CDATA[http://www.incomit.com/]]>
3.       <attr name="SystemFileSystem.localizingBundle"
    stringvalue="com.incomit.iwarf.modules.menus.incomit.Bundle"/>
```

```
4.       <attr name="SystemFileSystem.icon" urlvalue="nbresloc:/com/
     incomit/iwarf/modules/menus/resources/weblink.gif"/>
5.   </file>
```

| Line | Description |
|---|---|
| 1. | The filename of the virtual URL file. |
| 2. | Enter the URL inside the []. |
| 3. | Specifies the path to the bundle file, in this case used for specifying which letter to underscore in the menu item name. |
| 4. | Path to the 16x16 icon that will appear in the menu. |

Then add the following to the bundle file (`Bundle.properties`) in the `incomit` directory.

```
1.   Menu/Incomit/incomit-web-link.url=Incomit &Web
```

This will make the W letter underscored.

**Creating a separator in a menu**

Just add the following line after `<file name="com-incomit-iwarf-modules-menus-incomit-HelloWorldWizardAction.instance"/>` in the layer file.

```
1.   <file name="Sep1[javax-swing-JSeparator].instance"/>
```

If multiple separators are wanted in the same menu, name them Sep1, Sep2, …

**Ordering menu items**

If the following order is wanted of the menu items in the menu: wizard menu item, the separator and last, the URL. Then add the following two lines to the layer file after the separator tag above.

```
1.   <attr name="com-incomit-iwarf-modules-menus-incomit-
     HelloWorldWizardAction.instance/Sep1[javax-swing-
     JSeparator].instance" boolvalue="true"/>
2.   <attr name="Sep1[javax-swing-JSeparator].instance/incomit-web-
     link.url " boolvalue="true"/>
```

| Line | Description |
|---|---|
| 1. | This attribute makes NetBeans put the wizard menu item before the separator. Put this line between the wizard virtual file and separator virtual file in the layer file. |
| 2. | This attribute makes NetBeans put the separator before the URL. Put this line between the separator virtual file and the URL virtual file in the layer file. |

## 6.7 Filesystem example

Instead of accessing files through pure Java™ API's, NetBeans offers the package `org.openide.filesystems`, where the FileSystem and the Repository classes can be accessed. The Repository is a singleton class that corresponds to the Explorer window in NetBeans. The Repository consists of all mounted filesystems. The NetBeans crew describes the Filesystems API as follows [2]:

*"The FileSystems API permits module authors to access files in a uniform manner: e.g. you may be unaware of whether a file you are using is stored on local disk in the user's repository, or stored in an auxiliary directory, or stored in a JAR archive. Alternately, you may have reason to implement a custom file system--e.g. a vendor tool being integrated into the IDE may have its own local or remote storage of files in a special fashion; using this API, the rest of the IDE will be able to seamlessly view your files."*

### 6.7.1 Simple example

Consider the following. A local file, `c:\myfolder\myfile.txt`, should be mounted so it will be accessible in the Explorer window in the NetBeans IDE. This is a quite simple task to perform. If the folder, `c:\myfolder` does not exist, it will be created. If the file, `myfile.txt` does not exist, it will be created. First the following directory structure has to be created:

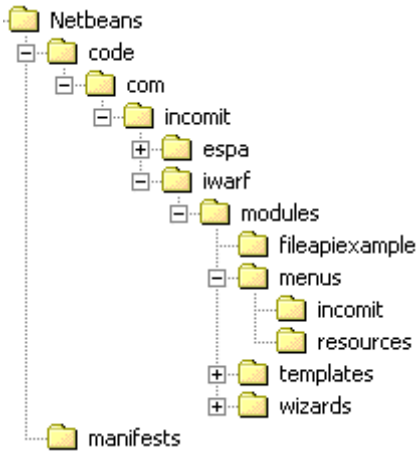


*Figure 17: Filesystem directories*

**Layer.xml**

The `Layer.xml` file is used to add the menu item into the Incomit folder.

Add the following line, to the virtual folder Menu -> Incomit in the layer file.

```
1. <file name="com-incomit-modules-menus-incomit-
   FileExampleAction.instance"/>
```

**Bundle.properties**

The bundle file is called from the `FileExampleAction` class. It is used to name the menu item “File Example”. Add this line to the bundle file in the `filesystem` directory:

```
1. LBL_FileExampleAction=&File Example
```

**Example.java**

This class mounts `c:\myfolder\myfile.txt` to the NetBeans Explorer window. Put Example.java (see source in Appendix B.7) in the directory `fileapiexample`.

**FileExampleAction.java**

This file extends `CallableSystemAction` and its main purpose is to start the `Example` class. It also holds a function to fetch its name from the bundle file. This is the name that will be visible for the user in the Incomit menu. Put the file (see source in Appendix B.8) in the directory `incomit`. Create the 16x16 pixels icon, `FileExampleAction.gif`, and put it in the `resource` directory.

Compile the files using:

```
  javac -deprecation -g
c:/netbeans/lib/openide.org;c:/netbeans/lib/openide-
util.org;c:/netbeans/lib/openide-fs.org
com/incomit/iwarf/modules/fileapiexample/*.java
com/incomit/iwarf/modules/menus/incomit/*.java
```

## 6.8 Other

This chapter will explain how to change the start-up logotype and the window title, and how to enable code completion.

### 6.8.1 Changing the start-up logotype

To change the start-up logotype, create a `c:\core\jar` temp directory and unpack the `[NetBeans installation directory]\lib\core.jar` (easiest with WinZip) into that directory. Copy the `\meta-inf\Manifest.mf` to the `core` directory. Open, edit and save the logotype file `c:\core\jar\org\netbeans\core\resources\splash.gif`. Then pack the JAR by entering the `jar` directory and enter the command `jar cfm0 core.jar ../Manifest.mf *`. Move the new `core.jar` file into the `c:\netbeans\lib` and overwrite the existing file.

### 6.8.2 Changing the coordinates of the text in the start-up logotype

To alter the loading text coordinates in the splash screen, assuming that the `core.jar` is unpacked as in chapter 6.8.1, open the bundle property file `c:\core\jar\org\netbeans\core\Bundle.properties` and change the SplashRunningTextBounds attribute. The numbers are x-coordinate, y-coordinate, width and height separated by comma. Save the file and pack the JAR-file like in chapter 6.8.1.

### 6.8.3 Changing the window title.

To set the NetBeans window title, edit the very same Bundle.properties as in chapter 6.8.2. Change the currentVersion attribute to whatever is preferred. The bundle property file `c:\core\jar\org\netbeans\core\windows\Bundle.properties` also has to be edited. Here change the CTL_MainWindow_Title and CTL_MainWindow_Title_Project to whatever is preferred. Save the files and pack the JAR as in chapter 6.8.1.

### 6.8.4 Add code to the parser database (code completion)

To enable code completion for objects, install the JAR file containing the objects into NetBeans as was done in chapter 6.3.5. Then start NetBeans and choose File->Mount Filesystem and choose to add the JAR file. Hit Ok to mount it. Right-click the mounted JAR file in the Explorer window and choose Tools->Update Parser Database… Choose a name and select which classes/fields/methods to include in the database and hit Ok. Two new files will be created in the `c:\netbeans\system\ParserDB` or the `[NetBeans installation directory]\myuserfiles\system\ParserDB` with the filename decided and the extensions `.jcb` and `.jcs`. These files can be moved between different installations of NetBeans if put in the `[NetBeans installation directory]\system\ParserDB` directory.

# 7 Use case – Create and deploy iSea SLEE Client

Not much testing has been done in this Bachelor's project, partly because iWarf is a prototype, but also because there are no good ways of testing it. Instead a use case will be described in this chapter. When going through this case, all features added to NetBeans will be used. This chapter describes how to create a simple iSea client with the iWarf tool.

## 7.1 Environment

For this prototype, only an iSea SLEE Client can be created. The example client will create a conference call with three participants using the ESPA ECall. At this point the ESPA ECall exists only as an API, so a dummy class was made, that only does printouts. The creation of an iSea SLEE Client starts with choosing a template. Thereafter items from the palette can be dropped to the project. Then the user must write some trivial code. When the client is complete the user will create a descriptor XML file, needed by the SLEE. The client files will be packed in a JAR-file, using the NetBeans JAR packager. Finally the JAR-file will be deployed into iSea using the integrated Incomit deployment tool.

## 7.2 Templates

First choose to create a new project file (ctrl + n). Browse to the iSea SLEE Client template (see Figure 18) and then continue by clicking next.


*Figure 18: Choosing the template*

The next step is to name the project and to determine which package it will belong to. For this example choose to name the project `MyTestClient` and place it in the package `com.incomit.testing` (see Figure 19 and Figure 20).


*Figure 19: Project name*


*Figure 20: Project package*

When done choosing the template, press finish to complete this wizard. This will make NetBeans think for a few seconds, so just stand by for a while. Now the two windows, Form [MyTestClient] and Source Editor [MyTestClient] will pop up. The Source editor now contains prewritten code that matches the selected template, in this case the iSea SLEE Client.

## 7.3 Palettes

The form window is needed to drop palette components into. Choose to drop an ESPA ECall to the project. Do this by clicking once on the palette component (see Figure 21) and then click anywhere in the form window.

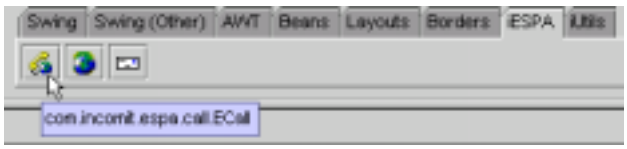


*Figure 21: iESPA palette*

Two lines of code will be generated in `MyTestClient.java`, a declaration line and an initialisation line. The name of the object will be [classname] followed by an incrementing number, in this case, `eCall1`.

## 7.4 Coding

The created object can be seen in the component inspector (see Figure 22). With the component inspector many nice features can be accessed, such as renaming the object and setting properties of the object.

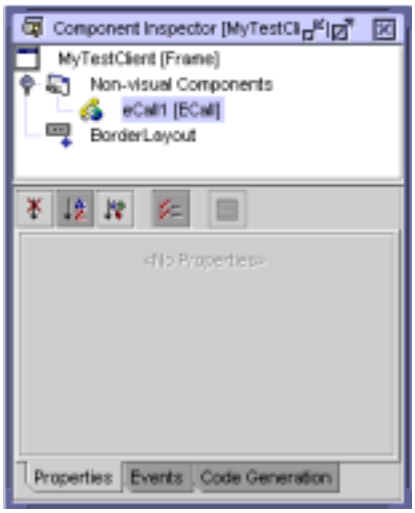


*Figure 22: Component inspector*

Now it is time to create the conference call. To make it easier for the user, the ECall class library has been added to NetBeans parser database (see chapter 6.8.4). This enables code completion as seen in the figure below.

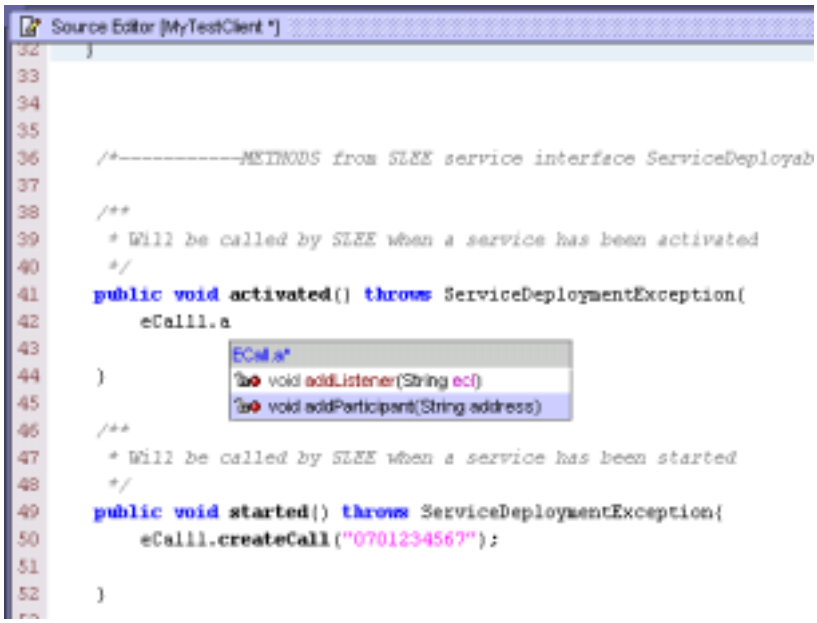


*Figure 23: Code completion*

To create the call, write the following code:

```
1.  eCall1.createCall("0701234567");
2.  eCall1.addParticipant("0707894561");
3.  eCall1.addParticipant("0701223344");
4.  eCall1.endCall();
```

In this example put line 1 in the method `started()`, line 2 and 3 in the method `activated()` and line 4 in the method `deactivated()`. Save and compile the file by hitting F9.

## 7.5 Descriptor Wizard

To be able to deploy this client to iSea, the descriptor file, `srv_depl.xml`, is needed. This file contains information about the client, and is needed by the SLEE. If this file is missing or is incorrect the client will not be accepted by the SLEE. In the Incomit menu, there is a Descriptor wizard that can be used to create the descriptor file in four easy steps (see Appendix C):

- **Basic service definition**

  In the first step the only required field to fill in is Name. This is the name the client will have in the SLEE. Set the name to "MyClient". The other fields are optional, and can be left out without change.

- **Service classes definition**

  In this step there is one required field, Deployable. Since there is only one file in the project, `MyTestClient.java`, and this file implements the Deployable interface, fill in the deployable field like this: `com.incomit.testing.MyTestClient`. The remaining fields cannot be filled in, since there are no classes implementing any of the mentioned interfaces.

- **Choose descriptor path**

  Now that all required fields are filled out, specify where to save the descriptor file. Select the directory, either writing it by hand or browse for the directory of your desire. Later on the descriptor file will be placed into the root of a JAR-file as well as the client package, so it might be a good idea to save it with this structure at this stage already (see Figure 24).


*Figure 24: File structure*

- **Manual edit**

This step gives the opportunity to quickly view and change the content of the descriptor file. If it looks ok, press finish to create the `srv_depl.xml`. Shortly after, the `srv_depl.xml` will appear in the Explorer window, as shown in the figure above.

## 7.6 JAR-packaging

NetBeans has a built-in JAR-packager. The usage of this packager needs to be discussed a bit, because it might be a bit confusion of how it works. Create a JAR contents object from a template (see Figure 25).


*Figure 25: JAR contents*

In the next step enter a name for the JAR contents, and where to place it. Name the JAR contents `myjarcontents`. This object is not a JAR-file, but merely an object that describes the JAR-file. To create the JAR-file, specify which files that should be included in the JAR-file (see Figure 26) and then compile the JAR contents. The output of this compilation is the JAR-file. Press next to add files to the JAR contents.

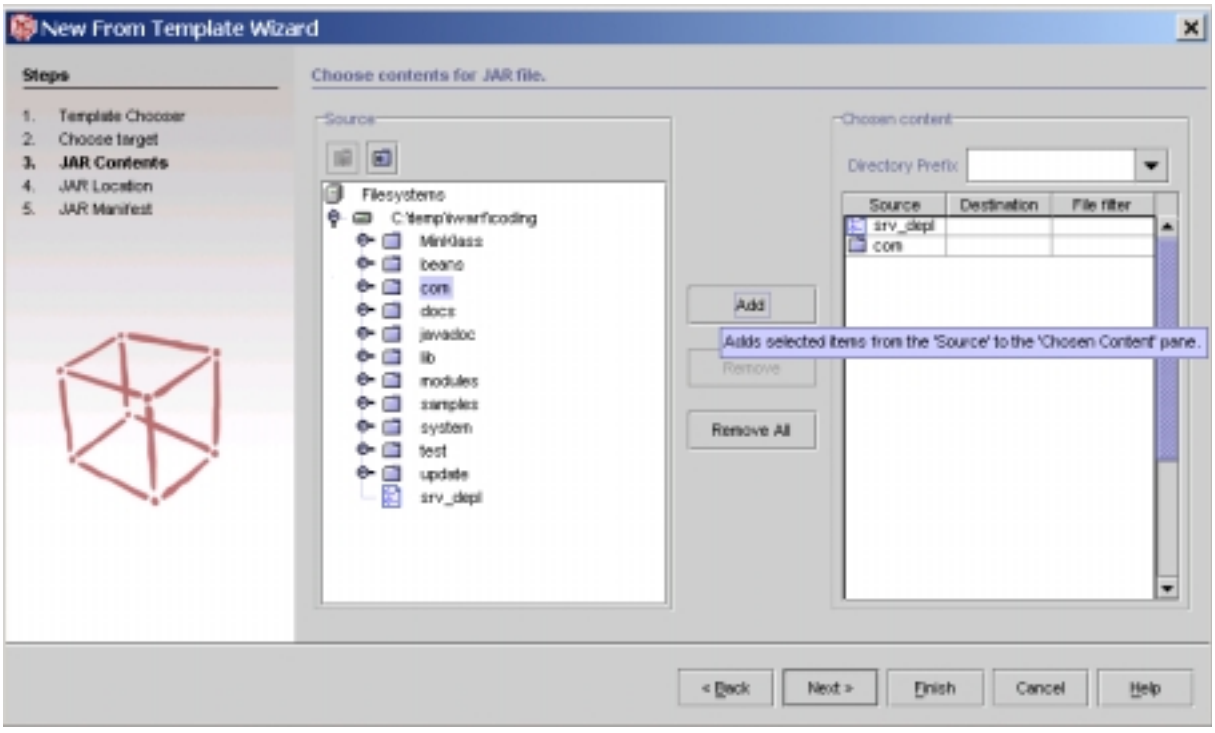


*Figure 26: Adding files and directories to the JAR contents file*

Press next to choose the JAR-file location. The final step is to write the manifest file, which is optional. Press finish to complete the wizard.

Locate `myjarcontents` in the Explorer window and right-click on it. If additional properties need to be set, choose properties in the popup menu (see Figure 27). When satisfied press compile in the popup menu to create a JAR-file from `myjarcontents`.

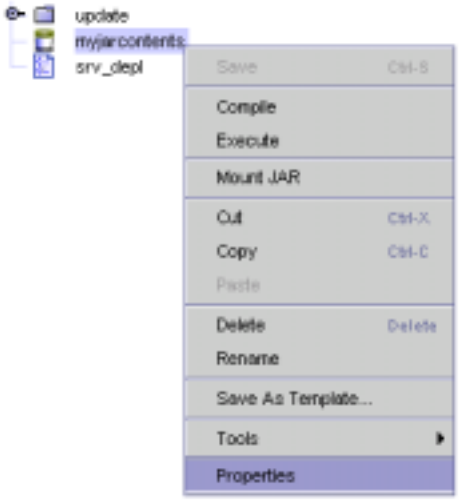


*Figure 27: JAR contents properties*

## 7.7 Deploy to iSea

Access to iSea is needed to complete this step. To deploy the created JAR-file, choose `Deploy to iSea` in the `Incomit` menu. The window that pops up is the standard Incomit deployment tool, just slightly adjusted to fit into iWarf. How the deployment tool, iSea and iWarf communicate is illustrated in Figure 28. Since the deployment tool is under development and the authors of this document is not engaged with that project, details about its usage will not be given. The ECall class will do printouts on the iSea machine.

When starting it, `createCall: 0701234567` is printed on the screen.

When activated, `addParticipant: 0707894651 [newline] addParticipant: 0701223344` is printed.

When deactivated, `endCall` is printed.

An iSea service has step by step successfully been created using the extra features added to NetBeans.

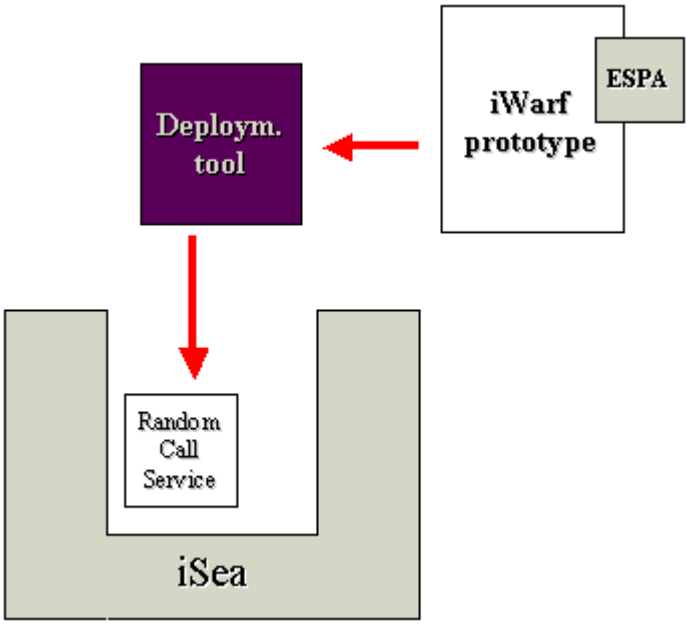


*Figure 28: How iWarf and iSea are related*

# 8 Problems and experiences

Since this Bachelor's project was about making a prototype, focus was intentionally on finding a way to achieve as many goals as possible, without adding features not included in the requirements specification. Even though code completion was not listed as a requirement, it was a simple, but powerful feature to include.

## 8.1 Problems

Along the way some problems of different character were encountered. The problems can be divided into NetBeans related, implementation and Incomit related problems.

### 8.1.1 NetBeans related problems

NetBeans is an IDE in change. At first NetBeans 3.1 was used as foundation. When writing the descriptor wizard, problems occurred. The wizard GUI did not work properly. After some research on the NetBeans website, we came to the conclusion that the fault was embedded in NetBeans. The only solution to this problem was to update NetBeans. An early 3.2 beta release was installed. When trying to install the iWarf module to this version of NetBeans, a lot of exceptions were thrown when starting up the IDE. For some reason (probably a good one) the NetBeans class library files had been split into several smaller files. There were also changes made to the form editor, so adjustments to the code had to be done. The JAR packager was changed too. In NetBeans 3.1 it was located in the tools menu, but was now a template! Upgrading to the 3.2 beta required several days of work. When the wizard was completed, we discovered that the JAR packager in this release was not working properly. This forced another upgrade. This upgrade went smoother, but some changes had been made to the form editor again! They probably were not happy with the last changes. For every new upgrade, changes in the iWarf module had to be done to fit in NetBeans.

### 8.1.2 Implementation problems

The making of the descriptor wizard was not easy. The first problem, described above, was caused by an old version of NetBeans. The wizard implements the `org.openide.WizardDescriptor.Iterator` interface, which was not easy to understand. There were also some problems concerning creating templates groups, in other words a

template containing more than one file. Through mailing lists and examples on NetBeans website, we finally got it to work.

### 8.1.3 Incomit related problems

As mentioned before in this document, the ESPA and Utils classes were not implemented. The solution to this problem has been to write a dummy class, ECall that was used for testing. Another problem that made it hard to test services, created in iWarf, is that Incomit temporarily lost the connection to the PSTN.

## 8.2 Experiences

Before we started designing iWarf we wrote a requirement specification document that was approved by Pär, our company supervisor. We tried to follow it from the start to the end, but some requirements where not achieved. The requirements R1 to R17 can be found in Appendix A. R1 and R2 are not functionally implemented due to lack of experience in CORBA (Common Object Request Broker Architecture). R3 actually has lower priority then R1 and R2, but R3 was easier to achieve. R8 – R12 is not fully implemented (see 8.1.3). R14 was split into two issues, creating XML descriptor file and packing the JAR file. The requirements R3-R7 and R13-R17 were fully implemented.

New experience was gained in XML, Java™ and GUI design but also in the design of telecom systems of tomorrow and what kind of telecom services that will be present in a near future. Since NetBeans was used as foundation of iWarf, we have learned the architecture of NetBeans and how to modify and use the IDE. Through mailing lists we have participated in discussions concerning development of the NetBeans IDE. Beside this we have learned how the modern IT company Incomit develops their products.

We wrote diaries every day, which was of great help when looking back to see what was done earlier. It also helped us to see working hours each day. When problems were solved, we wrote howto’s describing the problems, and how they were solved.

# 9 Conclusions

The purpose with this Bachelor's project was to create a service creation environment that makes it easier to develop telecom and Internet services.

Our conclusion is that service creating has never been easier. The required programming skills for making iSea services using iWarf are far less than writing them in a common editor. To set up a call between two persons, only three lines of code need to be written manually. Actually, that is not completely true, because the code completion helps the user writing those three lines. This means that we have succeeded creating a user-friendly IDE for fast development of telecom and Internet services.

In the preliminary investigation a comparison between Forte™ and NetBeans was made. The choice of using NetBeans was obvious, but it is hard to tell what the result would have been with another IDE. NetBeans is quite easy to modify because it is based on open APIs and the help from the mailing lists is really good. The NetBeans IDE is really easy to use, but even on today's high-end computers NetBeans is a bit slow.

Incomit will develop the commercial version of iWarf this summer. It will be based upon knowledge received from our work with the prototype. What can be done to achieve a commercial product? The main thing to be done is to change all texts and picture to be more Incomit specific, so the customers will not see the NetBeans logo and texts. Then the Incomit specific features can be added, like the requirements R1-R3 and R5-R16. More wizards might be used to make difficult tasks simplified for the users. iWarf is a very modular IDE with open API's so there is no end to the possibilities.

# Terminology and abbreviations

| | |
|---|---|
| CORBA | Common Object Request Broker Architecture. |
| ESPA | Incomit Easy SPA. |
| Forte™ | IDE based on NetBeans, developed by Sun. |
| GUI | Graphical User Interface. |
| HTTP | Hyper Text Transfer Protocol (protocol for transferring files for the web). |
| IDE | Integrated Development Environment. |
| Incomit | An Internet- and telecom-company in Karlstad. |
| ISCE | Incomit Service Creation Environment. |
| iSea | Application server connected to Internet and iSluice. |
| iSluice | Connected to the telecom network and to one or several iSea servers. |
| JAIN | The JAIN program is organized by a number of Expert Groups for Java™ telecom APIs. |
| JAR | Java™ ARchive file format. |
| JavaBeans™ | Java™ object that complies with the JavaBeans™ component architecture. |
| NetBeans | Open-source, modular IDE founded by Sun. |
| PDA | Personal Digital Assistant (a handheld computer). |
| PSTN | Public Switched Telephone Network. |
| SDS | System Design Specification. |
| servlet | Web server side Java™ application. |
| SLEE | Service Logic Execution Environment. |
| SPA | JAIN Service Provider API. |
| Sun | A worldwide company that among other things is the founder of the Java™ language. |
| Utils | Log, trace and database services implemented in iSea. |
| WAP | Wireless Application Protocol. |
| XML | eXtended Markup Language. |

# References


[1] Forte FAQ - http://www.sun.com/forte/ffj/faq/general.html [2001-05-17]

[2] NetBeans FileSystem API - http://www.netbeans.org/download/apis/org/openide/filesystems/doc-files/api.html [2001-05-17]

[3] NetBeans HelpContext - http://www.netbeans.org/download/apis/org/openide/util/HelpCtx.html [2001-05-17]

[4] NetBeans web - http://www.netbeans.org [2001-05-17]

[5] SLEE API specification - http://java.sun.com/aboutJava/communityprocess/jsr/jsr_022_jslee.html [2001-05-29]

# A Requirements specification

## Revision History

| Rev. | Date | Editor | Comments |
|---|---|---|---|
| PA1 | 2001-02-09 | MaGu, GoSk | First version. |
| PA2 | 2001-03-09 | MaGu | Some updates |
| A | 2001-05-04 | MaGu, GoSk | Final |

## Abstract

This document contains the requirements for the Bachelor's project, which will be done at the company, Incomit. The Bachelor's project can be divided to following parts; investigation, service creation environment, iSea service, documentation and presentation.

## Scope

The investigation will include:

- Investigate which development tool to use

The service creation environment will be modified to support these extra features:

- Incomit templates - easy way to create Incomit services
- Incomit palettes - easy way to add Incomit components by drag ‘n’ drop
- Incomit menus - for accessing special options, such as deploy
- Incomit GUI - an Incomit look-a-like environment
- Incomit Depl. tools - deploy service to iSea or file.

The iSea service:

- Create a service of our choice using the service creation environment

The presentation:

- Present the work for the Incomit crew

System creation environment advantages:

- Simple ness - easy way to create and deploy Incomit services
- Flexible - easy to modify

The main goal is to find an easy way to create Incomit services and report the result.

## Functional requirements for the service creation environment

Warf-R1. **Create an Incomit service project**

Done by choosing a specific Incomit service template when a new project is created.

Prio: 1

Warf-R2. **Create a specific Incomit servlet project**

Done by choosing a specific Incomit servlet template when a new project is created.

Prio: 2

Warf-R3. **Create a specific Incomit client project**

Done by choosing a specific Incomit client template when a new project is created.

Prio: 3

Warf-R4. **Investigate which development tool to use**

Compare different development tools to see which suits Incomit best.

Prio: 1

Warf-R5. **Create an ESPA palette**

Create a palette that holds Incomit ESPA services, like Call Control, Messaging and Positioning.

Prio: 1

Warf-R6. **Create an Utils palette**

Create a palette that holds Incomit utilities services, like DB, Log, Trace.

Prio: 1

Warf-R7. **Drag 'n' drop Incomit iSea Call Control service**

Be able to drag 'n' drop Incomit iSea Call Control service from the Incomit ESPA palette to the current project.

Prio: 1

Warf-R8. **Drag 'n' drop Incomit iSea Messaging service**

Be able to drag 'n' drop Incomit iSea Messaging service from the Incomit ESPA palette to the current project.

Prio: 1

Warf-R9. **Drag 'n' drop Incomit iSea Positioning service**

Be able to drag 'n' drop Incomit iSea Positioning service from the Incomit ESPA palette to the current project.

Prio: 1

Warf-R10. **Drag 'n' drop Incomit iSea DB service**

Be able to drag 'n' drop Incomit iSea DB service from the Incomit Utils palette to the current project.

Prio: 1

Warf-R11. **Drag 'n' drop Incomit iSea Log service**

Be able to drag 'n' drop Incomit iSea Log service from the Incomit Utils palette to the current project.

Prio: 1

Warf-R12. **Drag 'n 'drop Incomit iSea Trace service**

Be able to drag 'n' drop Incomit iSea Trace service from the Incomit Utils palette to the current project.

Prio: 1

Warf-R13. **Incomit menu**

Create an Incomit menu containing deploy submenus.

Prio: 1

Warf-R14. **JAR Packager**

Be able to create an XML descriptor (not necessary automatically) and pack the created service into a JAR-file.

Prio: 1

Warf-R15. **Deploy to iSea**

Be able to deploy the created service directly to iSea using Incomit deployment tool.

Prio: 1

Warf-R16. **Help files**

Built in help files for the user. Be able to get help about the Incomit tools.

Prio: 3

## The iSea Service

Warf-R17. **Create a service using iWarf**

Create a simple service that establishes a phone call, by using the service creation environment.

Prio: 1

## Installation

Installation will be done with a simple batch file.

## Platforms and compatibility

Environment:

Windows 2000 platform. For using iSea deployment tool, a connection to iSea is required.

Hardware:

Installation requires 50 MB of hard disk space.

Minimum configuration: 350MHz Pentium II, 128 MB RAM, and 128 MB paging file size.

Recommended configuration: 450MHz Pentium III, 256 MB RAM, and 256 paging file size.

Software:

Development tool and Java™ (TM) 2 SDK, Standard Edition, v. 1.3 (JDK 1.3) for Windows.

## Human Engineering

Uniform look, that is to say buttons and fonts are similar in the program. Incomit style and colours wherever is possible.

## Documentation

Bachelor's project specification

Timetable

Requirements specification

Bachelor's project documentation

Technical documentation (SDS)

Weekly reports

Diaries

# B Code

## B.1 iSeaClient_java

```
1.  import com.incomit.slee.*;
2.
3.  /*
4.   * __NAME__.java
5.   *
6.   * Created on __DATE__, __TIME__
7.   *
8.   * @author  __USER__
9.   */
10.
11. public class __NAME_sleeClient__ implements ServiceDeployable {
12.    public __NAME_sleeClient__() {
13.       initComponents();
14.    }
15.
16.    private void initComponents () {//GEN-BEGIN:initComponents
17.
18.    }//GEN-END:initComponents
19.
20.    public void activated() throws ServiceDeploymentException{
21.    }
22.
23.    public void started() throws ServiceDeploymentException{
24.    }
25.
26.    public synchronized void deactivated() throws
    ServiceDeploymentException{
27.    }
28.
29.    public void stopped() throws ServiceDeploymentException{
30.    }
31.
32.    public void setServiceContext(ServiceContext serviceContext) {
33.       m_serviceContext = serviceContext;
```

```
34.    }
35.
36.    // Components variables declaration - do not modify //GEN-
    BEGIN:variables
37.    // End of components variables declaration //GEN-END:variables
38.
39.    private ServiceContext m_serviceContext = null;
40. }
```

| Line | Description |
|---|---|
| 4. | __NAME__ will be replaced with the chosen class name when using the template in NetBeans. |
| 6. | __DATE__ and __TIME__ will be replaced with the current date and time respectively when using the template in NetBeans. |
| 8. | __USER__ will be replaced with your username when using the template in NetBeans. |
| 11-12. | __NAME_sleeClient__ will be replaced with the same as __NAME__. The extra _sleeClient is there to separate this class file from other template class files. |
| 16-18. | When dragging 'n' dropping a component from the Palette, the initialisation of the component takes place here because of the //GEN-BEGIN and //GEN-END. |
| 36-37. | When dragging 'n' dropping a component from the palette, the variable declaration of the component ends up here. |

## B.2 ECall.java

```
1.  package com.incomit.espa.call;
2.
3.  public class ECall {
4.       public ECall() {
5.       }
6.
7.       public void addListener(String ecl) {
8.                 System.out.println("addListener: " + ecl);
9.       }
10.
11.      public void addParticipant(String address) {
12.                System.out.println("addParticipant: " + address);
13.      }
14.
```

```
15.       public void createCall(String address) {
16.                 System.out.println("createCall: " + address);
17.       }
18.
19.       public void endCall(){
20.                 System.out.println("endCall");
21.       }
22.
23.       public String[] getParticipants(){
24.                 return m_participants;
25.       }
26.
27.       public void removeParticipant(String address){
28.                 System.out.println("removeParticipant: " + address);
29.       }
30.
31.       private java.lang.String m_address;
32.       private String[] m_participants = new String[] {"1","2"};
33. }
```

## B.3 HelloWorldPanel1.java

```
1.  package com.incomit.iwarf.modules.wizards.helloworld;
2.
3.  import java.awt.Component;
4.  import javax.swing.JLabel;
5.  import javax.swing.JPanel;
6.  import javax.swing.JTextField;
7.  import java.awt.event.KeyAdapter;
8.  import java.awt.event.KeyEvent;
9.  import javax.swing.event.ChangeEvent;
10. import javax.swing.event.ChangeListener;
11. import javax.swing.event.EventListenerList;
12. import java.net.URL;
13. import java.net.MalformedURLException;
14. import org.openide.util.HelpCtx;
15. import org.openide.WizardDescriptor;
16.
17. public class HelloWorldPanel1 extends JPanel implements
    WizardDescriptor.Panel {
18.
```

```
19.      public HelloWorldPanel1(){
20.                URL url = null;
21.                try {
22.                          url = new
    URL("nbresloc:/com/incomit/iwarf/
23.  modules/wizards/resources/helloworld_panel1.html");
24.                }
25.                catch (MalformedURLException e) {
26.                }
27.
28.                putClientProperty("WizardPanel_helpURL", url);
29.                putClientProperty("WizardPanel_contentSelectedIndex",
    new Integer(0));
30.
31.                setLayout(null);
32.
33.                addComponents();
34.                addActions();
35.      }
36.
37.      private void addActions() {
38.                jtf_hello.addKeyListener(new KeyAdapter() {
39.                          public void keyReleased(KeyEvent evt) {
40.                                    fireChangeEvent();
41.                          }
42.                });
43.      }
44.
45.      private void addComponents() {
46.                JLabel topInfo = new JLabel("Enter the secret
    password in the textfield to continue.");
47.                jtf_hello = new JTextField("");
48.
49.                add(topInfo);
50.                topInfo.setBounds(0,0,388,20);
51.
52.                add(jtf_hello);
53.                jtf_hello.setBounds(0,44,388,20);
54.      }
55.
56.      public void addChangeListener(ChangeListener l) {
```

```
57.                 listenerList.add (ChangeListener.class, l);
58.       }
59.
60.       public Component getComponent() {
61.                 return this;
62.       }
63.
64.       public HelpCtx getHelp() {
65.                 return HelpCtx.DEFAULT_HELP;
66.       }
67.
68.       public boolean isValid() {
69.                 if
    (jtf_hello.getText().toLowerCase().equals(MYSTRING)) {
70.                           return true;
71.                 }
72.                 else {
73.                           return false;
74.                 }
75.       }
76.
77.       public void readSettings(Object settings) {
78.       }
79.
80.       public void removeChangeListener(ChangeListener l) {
81.       }
82.
83.       public void storeSettings(Object settings) {
84.       }
85.
86.       private void fireChangeEvent() {
87.                 ChangeEvent e = null;
88.                 Object[] listeners = listenerList.getListenerList ();
89.                 for (int i = listeners.length-2; i>=0; i-=2) {
90.                           if (listeners[i]==ChangeListener.class) {
91.                                     if (e == null)
92.                                               e = new ChangeEvent
    (this);
93.
         ((ChangeListener)listeners[i+1]). stateChanged (e);
94.                           }
```

```
95.                 }
96.       }
97.
98.       private JTextField jtf_hello;
99.       private EventListenerList listenerList = new
     EventListenerList();
100.
101.      final static String MYSTRING = "hello world";
102. }
```

| Line | Description |
|---|---|
| 17. | The panel will extend the `JPanel` and implement the NetBeans interface `WizardDescriptor.Panel`. |
| 20-26. | Specifies the path to the help file URL that appears in the step 1 of the wizard. |
| 28. | Adds the URL to the wizard using `putProperty`. |
| 29. | Sets this panel's position in the wizard. In this case it is the first panel and it means that it is at position 0. (Position 1 for step 2, position 2 for step 3, …). |
| 31. | Sets the layout to null. |
| 38-42. | Adds a listener to the text field that listens for a key release. It will call the method `fireChangeEvent` if a key is pressed and released. |
| 57. | Adds a listener for changes to the listener list. This is for telling the main class that the password has been entered. |
| 65. | Return default help context because we do not need any help for this. We already have the html help file. |
| 68-75. | Returns true if the correct password is entered in the text field. |
| 77-84. | These methods implemented from the `WizardDescriptor.Panel` are not used here. |
| 86-96. | Fires a change event to tell all listeners that something has happened. |
| 101. | The secret password. |

## B.4 HelloWorldPanel2.java

```
1.  package com.incomit.iwarf.modules.wizards.helloworld;
```

```
2.
3.  import java.awt.Component;
4.  import javax.swing.JLabel;
5.  import javax.swing.JPanel;
6.  import javax.swing.event.ChangeListener;
7.  import java.net.URL;
8.  import java.net.MalformedURLException;
9.  import org.openide.util.HelpCtx;
10. import org.openide.WizardDescriptor;
11.
12. public class HelloWorldPanel2 extends JPanel implements
    WizardDescriptor.Panel {
13.
14.      public HelloWorldPanel2(){
15.                  URL url = null;
16.                  try {
17.                              url = new
    URL("nbresloc:/com/incomit/iwarf/
    modules/wizards/resources/helloworld_panel2.html");
18.                  }
19.                  catch (MalformedURLException e) {
20.                  }
21.                  putClientProperty("WizardPanel_helpURL", url);
22.                  putClientProperty("WizardPanel_contentSelectedIndex",
    new Integer(1));
23.
24.                  setLayout(null);
25.
26.                  addComponents();
27.      }
28.
29.      private void addComponents() {
30.                  JLabel topInfo = new JLabel("Congratulations!!! You
    have entered the secret password!!!");
31.
32.                  add(topInfo);
33.                  topInfo.setBounds(0,0,388,20);
34.      }
35.
36.      public void addChangeListener(ChangeListener l) {
37.      }
```

```
38.
39.       public Component getComponent() {
40.                 return this;
41.       }
42.
43.       public HelpCtx getHelp() {
44.                 return HelpCtx.DEFAULT_HELP;
45.       }
46.
47.       public boolean isValid() {
48.                 return true;
49.       }
50.
51.       public void readSettings(Object settings) {
52.       }
53.
54.       public void removeChangeListener(ChangeListener l) {
55.       }
56.
57.       public void storeSettings(Object settings) {
58.       }
59. }
```

## B.5 HelloWorld.java

```
1.  package com.incomit.iwarf.modules.wizards.helloworld;
2.
3.  import javax.swing.event.ChangeListener;
4.  import java.beans.PropertyChangeListener;
5.  import java.beans.PropertyChangeEvent;
6.  import java.text.MessageFormat;
7.  import org.openide.DialogDescriptor;
8.  import org.openide.NotifyDescriptor;
9.  import org.openide.TopManager;
10. import org.openide.WizardDescriptor;
11.
12. public class HelloWorld implements WizardDescriptor.Iterator {
13.       public HelloWorld(){
14.                 panel1 = new HelloWorldPanel1();
15.                 panel2 = new HelloWorldPanel2();
16.                 panel1.setName(PANEL_NAMES[0]);
```

```
17.                 panel2.setName(PANEL_NAMES[1]);
18.                 panels = new WizardDescriptor.Panel[] {panel1,
    panel2};
19.
20.                 wd = new WizardDescriptor(this);
21.                 wd.putProperty("WizardPanel_autoWizardStyle", new
    Boolean(true));
22.                 wd.putProperty("WizardPanel_contentDisplayed", new
    Boolean(true));
23.                 wd.putProperty("WizardPanel_helpDisplayed", new
    Boolean(true));
24.                 wd.putProperty("WizardPanel_contentNumbered", new
    Boolean(true));
25.                 wd.putProperty("WizardPanel_contentData", new
    String[]{"Enter the first secret password.", "Result"});
26.                 wd.setTitleFormat(new MessageFormat("{0}"));
27.                 wd.setTitle("Hello World Wizard");
28.
29.                 PropertyChangeListener listener = new
    PropertyChangeListener() {
30.                             public void
    propertyChange(PropertyChangeEvent event) {
31.                                     if (event.getPropertyName().
    equals(DialogDescriptor.PROP_VALUE)) {
32.                                             Object option =
    event.getNewValue();
33.                                             if (option ==
    WizardDescriptor.FINISH_OPTION){
34.                                             }
35.                                     }
36.                             }
37.                 };
38.
39.                 wd.addPropertyChangeListener(listener);
40.                 wd.setOptions (new Object[] {
    WizardDescriptor.PREVIOUS_OPTION, WizardDescriptor.NEXT_OPTION,
    WizardDescriptor.FINISH_OPTION, NotifyDescriptor.CANCEL_OPTION} );
41.                 TopManager.getDefault().createDialog(wd).
    setVisible(true);
42.     }
43.
```

```
44.       public WizardDescriptor.Panel current() {
45.                 return panels[current_panel];
46.       }
47.
48.       public boolean hasNext() {
49.                 if (current_panel == 0) {
50.                           return panel1.isValid();
51.                 }
52.                 else {
53.                           return false;
54.                 }
55.       }
56.
57.       public boolean hasPrevious() {
58.                 if (current_panel > 0) {
59.                           return true;
60.                 }
61.                 else {
62.                           return false;
63.                 }
64.       }
65.
66.       public String name() {
67.                 return PANEL_NAMES[current_panel];
68.       }
69.
70.       public void nextPanel() {
71.                 current_panel++;
72.       }
73.
74.       public void previousPanel() {
75.                 current_panel--;
76.       }
77.
78.       public void addChangeListener(ChangeListener l) {
79.       }
80.
81.       public void removeChangeListener(ChangeListener l) {
82.       }
83.
84.       static final int NR_OF_PANELS = 2;
```

```
85.       static final String[] PANEL_NAMES = {"Password step", "Final
      step"};
86.
87.       private WizardDescriptor.Panel[] panels;
88.       private WizardDescriptor wd;
89.       private HelloWorldPanel1 panel1;
90.       private HelloWorldPanel2 panel2;
91.
92.       private int current_panel = 0;
93.  }
```

| Line | Description |
|---|---|
| 12. | The main class will extend the `WizardDescriptor.Iterator` interface that includes some useful methods for making this class a wizard. |
| 14-15. | Creates the two panels. |
| 16-17. | Set the names of the panels. |
| 18. | Adds the panels to an array of `WizardDescriptor.Panels`. This is useful when having a lot of panels. |
| 20. | Creates the `WizardDescriptor`. |
| 21. | Turns on subtitle creation (the text above the panel). |
| 22. | Turns on the displaying of the steps pane. |
| 23. | Turns on the displaying of the help pane. |
| 24. | Turns on the numbering of steps. |
| 25. | Sets the steps name, which will be displayed, in the content pane. |
| 26. | Sets the subtitle format. {0} is the name of the panel. |
| 27. | Sets the title of the dialog. |
| 29-39. | Adds a listener for the buttons. In this case we listen for the `FINISH` button as an example of how you can do. |
| 40. | Sets the buttons that will appear in the wizard. |
| 41. | Shows the wizard dialog. |
| 45. | Returns current panel. |
| 48- | Return true here if the next button should be enabled. |

| | |
|---|---|
| 55. | |
| 57-<br>64. | Return true here if the previous button should be enabled. |
| 67. | Returns the name of the current panel. |
| 70. | When the next button has been hit this method will be called. |
| 75. | When the previous button has been hit this method will be called. |

## B.6 HelloWorldWizardAction.java

```
1.  package com.incomit.iwarf.modules.menus.incomit;
2.
3.  import org.openide.util.HelpCtx;
4.  import org.openide.util.NbBundle;
5.  import org.openide.util.actions.CallableSystemAction;
6.  import com.incomit.iwarf.modules.wizards.helloworld.HelloWorld;
7.
8.  public class HelloWorldWizardAction extends CallableSystemAction {
9.       public String getName () {
10.                return
    NbBundle.getBundle(HelloWorldWizardAction.class).
    getString("LBL_HelloWorldWizardAction");
11.      }
12.
13.      public HelpCtx getHelpCtx () {
14.                return HelpCtx.DEFAULT_HELP;
15.      }
16.
17.      protected String iconResource () {
18.                return "/com/incomit/iwarf/modules/menus/resources/
    HelloWorldWizardAction.gif";
19.      }
20.
21.      public void performAction () {
22.                HelloWorld wizard = new HelloWorld();
23.      }
24. }
```

| Line | Description |
|---|---|

| | |
|---|---|
| 8. | Extends the `CallableSystemAction` class. |
| 10. | Returns the name taken from `LBL_HelloWorldWizardAction` property in the bundle file (`Bundle.properties`) in the `incomit` directory. |
| 14. | Returns the help context. If you do not have any help to return, use the `HelpCtx.DEFAULT_HELP`. |
| 18. | Here is the path to the icon that is shown in the menu. It is a 16x16 gif icon. |
| 21. | This method is called when the user clicks on the menu item. |
| 22. | Create and show a new HelloWorld wizard. |

## B.7 Example.java

```
 1.  package com.incomit.iwarf.modules.fileapiexample;
 2.  import org.openide.filesystems.FileObject;
 3.  import org.openide.filesystems.Repository;
 4.  import org.openide.TopManager;
 5.  import java.net.URL;
 6.  import java.lang.Exception;
 7.  import java.io.File;
 8.  import org.openide.filesystems.LocalFileSystem;
 9.  import org.openide.filesystems.FileLock;
10.  import org.openide.filesystems.FileAlreadyLockedException;
11.  import org.openide.filesystems.FileLock;
12.  import java.io.OutputStream;
13.  import java.io.IOException;
14.  import java.beans.PropertyVetoException;
15.  public class Example {
16.       public Example(){
17.                 this.fileDir = fileDir;
18.                 if ( (new File(fileDir)).isDirectory() == false)
19.                           (new File(fileDir)).mkdirs();
20.                 try {
21.                           fs.setRootDirectory(new File(fileDir));

22.                 } catch (PropertyVetoException e) {
23.                           System.out.println("Vetoed by someone
     else");
24.                           e.printStackTrace();
25.                 } catch (IOException e) {
```

```
26.                             System.out.println("Could not set Root
    Directory to fs");
27.                             e.printStackTrace();
28.                 }
29.                 file=fs.findResource(fileName);
30.                 if (file == null) {
31.                             try {
32.                                         file=fs.getRoot().createData(
    fileNameNoExtension, extension);
33.                             } catch (IOException e) {
34.                                         System.out.println("Could not
    create file");
35.                                         e.printStackTrace();
36.                             }
37.                 }
38.                 if (repo.findFileSystem(fs.getSystemName()) == null)
    {
39.                             repo.addFileSystem(fs);
40.                 }
41.       }
42.       //Variables
43.       private FileObject file;
44.       private OutputStream outFile;
45.       private Repository repo =
    TopManager.getDefault().getRepository();
46.       private LocalFileSystem fs=new LocalFileSystem();
47.       private String fileURL;
48.       private String extension = new String("txt");
49.       private String fileNameNoExtension = new String("myfile");
50.       private String fileName = new String("myfile.txt");
51.       private String fileDir = new String("c:\\myfolder\\");
52.   }
```

| Line | Description |
|---|---|
| 17. | Sets the `fileDir` to `c:\myfolder` |
| 18. | Check if `fileDir` is a valid folder (if it exist). |
| 19. | Create `fileDir` if not exist. (the method `mkdirs()` can create folders recursively, for example, if you want to create the subfolder `c:\myfolder\sub`, and none of these exists, they will both be created) |

| | |
|---|---|
| 21. | We set the `fs` root directory to `c:\myfolder`. |
| 29. | Search for the file, `myfile.txt` in `fs`. On success, store the returned `FileObject` to `file`, else `file` will be null. |
| 30-37. | If `file` is null, create `myfile.txt` in `fs` root directory (`c:\myfolder`). |
| 38-40. | Check if `fs` is mounted in the repository, if not add `fs` to the repository. This will make the filesystem, `fs`, visible in the NetBeans Explorer window. |

## B.8 FileExampleAction.java

```
 1.  package com.incomit.iwarf.modules.menus.incomit;
 2.
 3.  import java.awt.BorderLayout;
 4.  import java.awt.Dimension;
 5.  import java.awt.Color;
 6.  import javax.swing.JFrame;
 7.  import java.awt.Font;
 8.  import javax.swing.JLabel;
 9.  import javax.swing.SwingConstants;
10.  import org.openide.util.HelpCtx;
11.  import org.openide.util.NbBundle;
12.  import org.openide.util.actions.CallableSystemAction;
13.  import org.openide.windows.TopComponent;
14.
15.  public class FileExampleAction extends CallableSystemAction {
16.
17.       public String getName () {
18.                 return NbBundle.getBundle
     (FileExampleAction.class).getString ("LBL_FileExampleAction");
19.       }
20.
21.       public HelpCtx getHelpCtx () {
22.                 return HelpCtx.DEFAULT_HELP;
23.       }
24.
25.       protected String iconResource () {
26.                 return "/com/incomit/modules/menus/
27.  resources/FileExampleAction.gif";
28.       }
```

```
29.
30.         public void performAction () {
31.                     Example iExample = new Example();
32.         }
33.     }
```

| Line | Description |
|---|---|
| 17-19. | Gets the name (`LBL_FileExampleAction`) from the bundle file. |
| 21-23. | About the class `HelpCtx`, taken quoted from reference [3]:<br><br>`"Provides help for any window or other feature in the system. It is designed to be JavaHelp-compatible and to use the same tactics when assigning help to JComponent instances."`<br><br>Look it up on the web if you need to have one. If you do not have a `HelpCtx` to return, just return the `DEFAULT_HELP`, as in this code. |
| 25-27. | Gets the URL to the icon used in the menu. This could actually also be fetched from the bundle file, as we did with the name. In this example both alternatives are shown. |

# C Descriptor wizard steps

This appendix explains the four steps for creating a descriptor file, using the descriptor wizard.

1. **Basic service definition**

   The user will fill out all, or some of these fields.

   - **Name (required)**

     The name the service will have in the SLEE.
   - **Version**

     The service version number.
   - **Company**

     The company that developed the service.
   - **Parent C.L.S.**

     Parent Class Loader Service. Sets the parent class loader of this service.
   - **Max alarms**

     Max number of alarms before the service will be shutdown.
   - **Trace**

     Turns logging to file on/off.

2. **Service classes definition**

   The user will fill out all, or some of these fields:

   - **Deployable (required)**

     Class that implements the deployable interface, needed for installation into iSea.
   - **Accessible**

     Class that implements the accessible interface, needed for outside access from a web page or servlet.
   - **Manageable**

     Class that implements the manageable interface, needed if the service should be managed from another application.
   - **Manageable_idl**

     Class that implements the manageable_idl interface. Describes the interface for the manageable object.

3. **Choose descriptor path**

   The user will browse to, or type in the path where the descriptor file will be stored. An exception will be thrown if the file is read-only, or locked by another program. The user will not get pass this step, until a legal path is entered.

4. **Manual edit**

   The generated descriptor file is shown for the user in a text-field. The user can edit the generated text, or choose finish to create the descriptor file. When the descriptor file is created, it will be added to the NetBeans Explorer window, so it can be accessed from the NetBeans IDE.