

BYPS – Communication Layer for Distributed Software

Wolfgang Imig, 2014-03-23
“under construction”

Abstract: [BYPS] (Binary Portable Serialization) connects Java, C#, C++ and JavaScript client programs via HTTP to Java EE servers using an RPC protocol. By means of an integrated push mechanism, a server can also call interfaces implemented on clients and clients can call interfaces among each other (over one or more servers). This is possible even if clients are connected to different servers. The underlying protocol either uses an optimized binary data serialization or JSON. Service interfaces and data structures are described in pure Java and some particular javadoc tags. In addition to primitive data types, the protocol supports lists, sets, maps and streams. A versioning mechanism allows to connect older clients to newer servers and vice versa.

Inhaltsverzeichnis

1 Overview.....	1
1.1 Define the API.....	2
1.2 Generate the Serialization Layers.....	3
2 Supported Platforms.....	4
3 Development.....	4
3.1 Requirements.....	4
3.2 Setup Java Development Workspace.....	4
3.3 Define “taskapp” API and Create Serialization Code.....	17
3.4 Implement the Web-Service “taskapp-srv”.....	22
3.5 Implement the Java Client Application taskapp-client-j.....	31
3.6 Using HTTP Authentication.....	33
3.7 Versioning.....	36
3.8 Streams.....	37
3.9 Client-side Interface, Invoke Client from Server.....	39
3.10 JavaScript Client.....	41
3.11 Client-to-Client Communication.....	49
3.12 Develop C++ Client Applications.....	53
4 Eclipse Tips.....	54
4.1 Client Cannot Connect to Server, HTTP 404.....	54
5 Bibliography.....	54

1 Overview

The project workflow with BYPS has typically the following tasks:

- Define interfaces and data structures
- Generate the serialization layer
- Compile the generated code and the target projects

1.1 Define the API

The interfaces and data structures can be seen as an API. Unlike object oriented APIs, BYPS does not support classes to have any methods (except trivial getters and setters). Methods can only be

defined in interfaces and classes are constrained to hold data members (classes are DTOs, Data Transfer Objects).

Example of a server-side interface:

```
package my.example.byps;

import byps.RemoteException;
import byps.Remote;

public interface TaskService extends Remote {

    void addTask(TaskInfo task) throws RemoteException;

    List<TaskInfo> getTasks() throws RemoteException;

}
```

Example of a class (DTO):

```
package my.example.byps;

import java.io.Serializable;

public class TaskInfo implements Serializable {

    private final static long serialVersionUID = 123456789L;

    protected int id;

    protected String title;

    protected List<InputStream> attachments;

    public int getId() { ... }
    public void setId(int id) { ... }
    ...
}
```

Client-side interfaces (called from the server or from other clients) have to be tagged by **@BClientRemote**. This advises the BYPS generator to create skeleton classes for each target platform and not only for Java.

```
package my.example.byps;

import byps.RemoteException;
import byps.Remote;

/**
 * @BClientRemote
 */
public interface TaskNotify extends Remote {

    void updateTask(Task task) throws RemoteException;

}
```

An API is described by the mandatory class BApi, that defines a name and a version:

```
package my.example.byps;

public class BApi {
    public final static String VERSION = "1.0";
    public final static String NAME = "TaskApplication";
}
```

Usually, APIs evolve over the time. If a new member is added to a class, the API version has to be incremented and the new member has to be tagged by a **@since <version>**:

...

```

public class Task implements Serializable {

    private final static long serialVersionUID = 123456789L;

    protected int id;

    protected String title;

    protected List<InputStream> attachments;

    /**
     * @since 1.1
     */
    protected int priority;

    ...
}

```

Interface methods cannot be modified in newer API versions since this would break compatibility. Thus, it is a good idea to always have a class type parameter in methods (here FindTaskOptions), which could be extended if necessary.

```

...
public interface TaskService extends Remote {

    List<Task> findTasks(String filter, FindTaskOptions opts) throws RemoteException;

    ...
}

```

Class members or interface methods must not be removed, otherwise compatibility is lost. If members are no more needed, they can be set as `@deprecated`. But this is only a documentation tag, deprecated members are still transferred.

1.2 Generate the Serialization Layers

In order to generate the serialization layers for the target platforms, the BYPS generator tool „bypsgen.jar“ has to be invoked. The best way to achieve this is by using an ANT script.

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<project name="TaskApplication" default="build">

    <property name="byps.home" value="${workdir}/byps-lib/java"/>

    <target name="build">

        <java jar="${byps.home}/bypsgen.jar" fork="true" >

            <!-- API definition from directoy ... -->
            <arg value="--sourcepath"/><arg value="./src"/>

            <!-- Generate Java code into directory ... -->
            <arg value="-genj.dir-ser"/>          <arg value="..\taskapp-ser\src-ser"/>
            <arg value="-genj.dir-ser-json"/>    <arg value="..\taskapp-srv\src-ser-json"/>

            <!-- Generate C# code into directory ... -->
            <arg value="-gencs.dir-ser"/>        <arg value="..\taskapp-ser\src-cs"/>

            <!-- Generate C++ code into directory ... -->
            <arg value="-gencpp.dir-api"/>        <arg value="..\taskapp-ser\src-cpp/api"/>
            <arg value="-gencpp.dir-impl"/>      <arg value="..\taskapp-ser\src-cpp/impl"/>

            <!-- Generate JavaScript code into file ... -->
            <arg value="-genjs.dest"/>            <arg value="..\taskapp-srv\WebContent\taskapp.js"/>

        </java>
    </target>

```

2 Supported Platforms

The server side is based on the Java EE Servlet 3.0 specification. Currently, only Tomcat 7.0 has been used as development and run-time server.

The supported client side platforms are listed below:

Programming Platform	Runtime Platform
Java 7.0 or newer	Every operation system with a Java SE 7.0 run-time. Android.
C# and .NET 4.0 or newer	Windows XP SP 3 or newer. For a detailed list see system requirements for .NET 4.0.
MSVC 2012, C++11 Standard	Windows XP or newer. For a detailed list see system requirements of MSVC run-time components.
QT 5.0 (currently under development)	See system requirements for QT 5.0
JavaScript	Tested with Chrome Version 33, Firefox Version 26 and IE 11. Might also run on other browsers.

Tabelle 1: Supported Platforms

3 Development

This chapter shows on the basis of an example how to develop programs using BYPS. The application is called “taskapp” and allows to create and list task objects.

You must have basic Java, JavaScript and HTML knowledge to follow the example.

3.1 Requirements

Since Java is the leading development platform for BYPS, a Java Software Development Kit version 7.0 [JDK] or newer and a Java IDE is required.

This documentation uses the “Eclipse IDE for **Java EE Developers**” [ECLIPSE].

In order to develop and run the server-side application, the [TOMCAT] application server has to be installed. For using on Windows, download the 32bit or 64bit Windows ZIP file. Unpack the ZIP archive into an arbitrary directory.

3.2 Setup Java Development Workspace

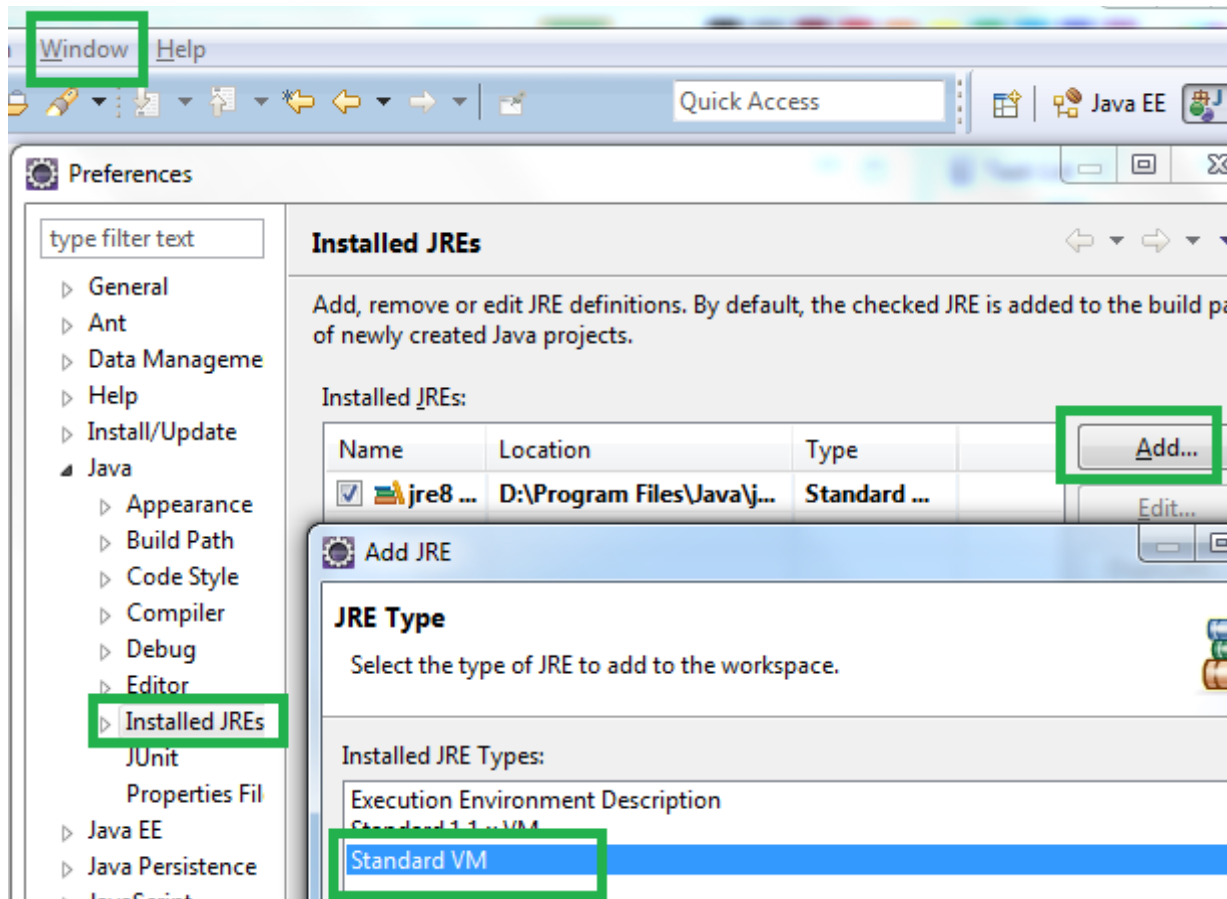
This section creates a workspace with the Java projects for API definition, serialization, web-service, and client application.

- Start the Eclipse IDE and enter a new workspace directory. This directory is referred to as “workdir” in the following.
- Click on the workbench symbol in the upper left corner in order to start working.

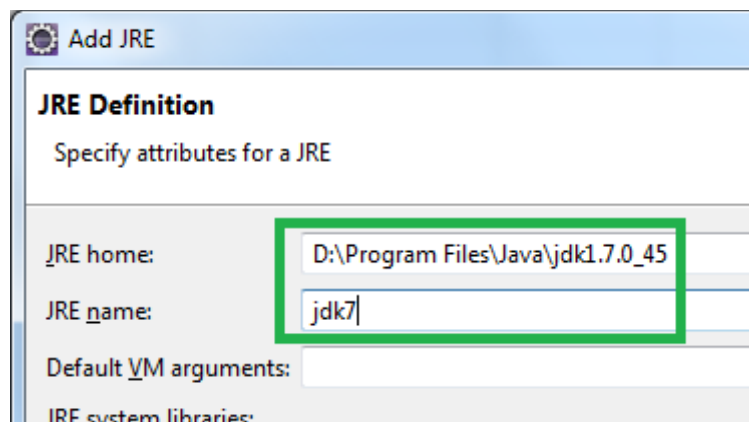


Configure Eclipse for using the [JDK]:

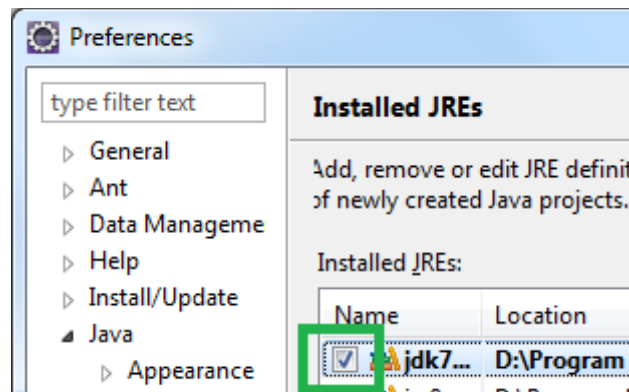
- Click menu “Window – Preferences”, “Java – Installed JREs”, “Add...”, select “Standard VM”:



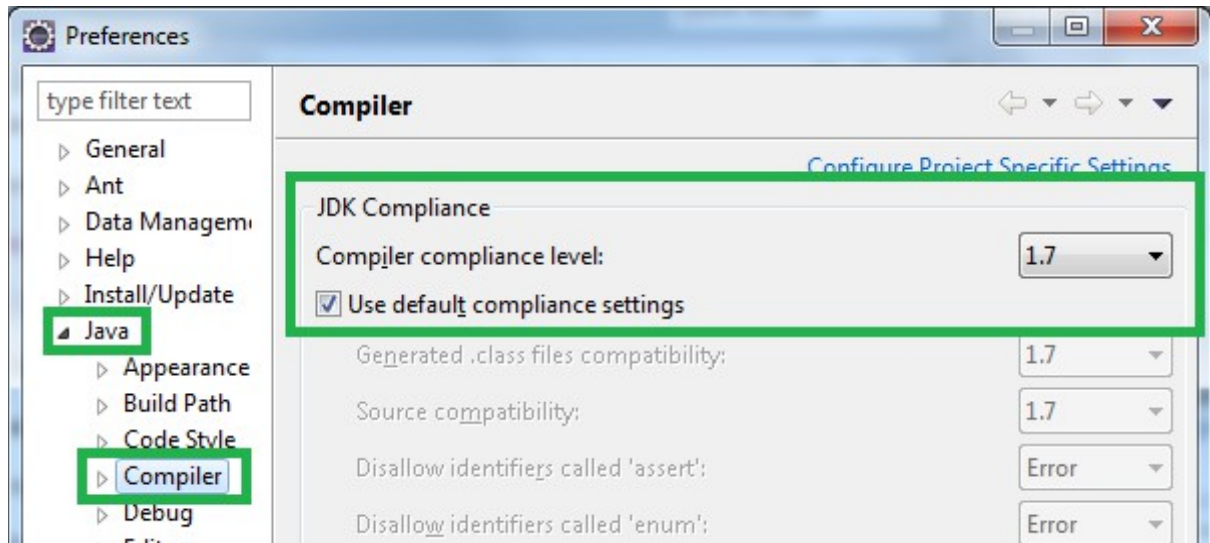
- “Next >”, select JDK installation directory and name it “jdk7”:



- “Finish”, set this JDK as the default Java runtime to be used:



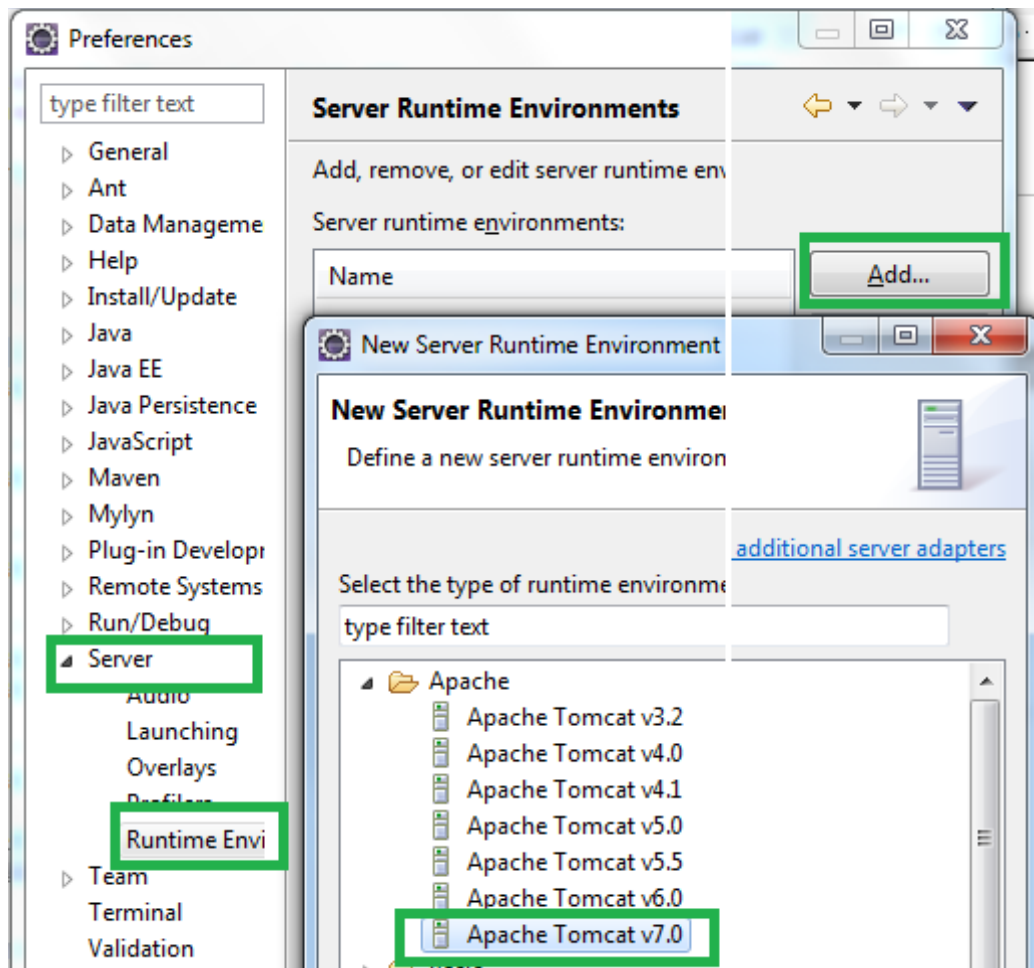
- Select in the tree on the left “Java – Compiler”, set JDK Compliance to 1.7 (at least 1.5):



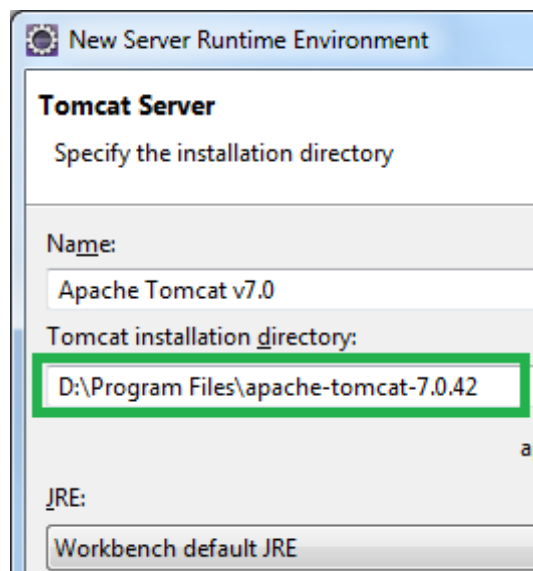
- “OK”.

Install [TOMCAT] in Eclipse IDE:

- Click menu “Window – Preferences”, “Server – Runtime Environments”, “Add...”



- “Next >”, enter Tomcat installation directory:



- “Finish”

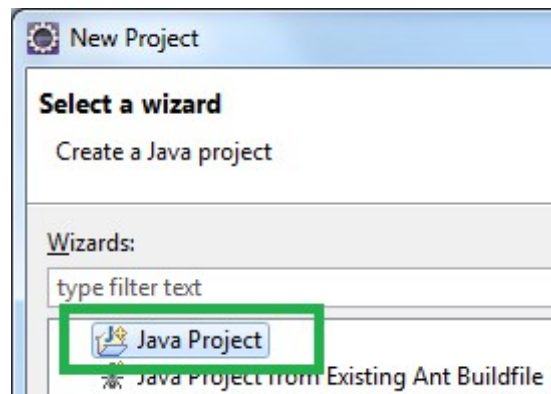
Install [LIBDIR]:

- Download “byps-lib.zip” from [LIBDIR] (click “View Raw”)
- Unpack “byps-lib.zip” into “workdir/byps-lib”
- Copy file “tools.jar” from JDK installation directory, sub-directory “lib”, into “workdir/byps-lib”.

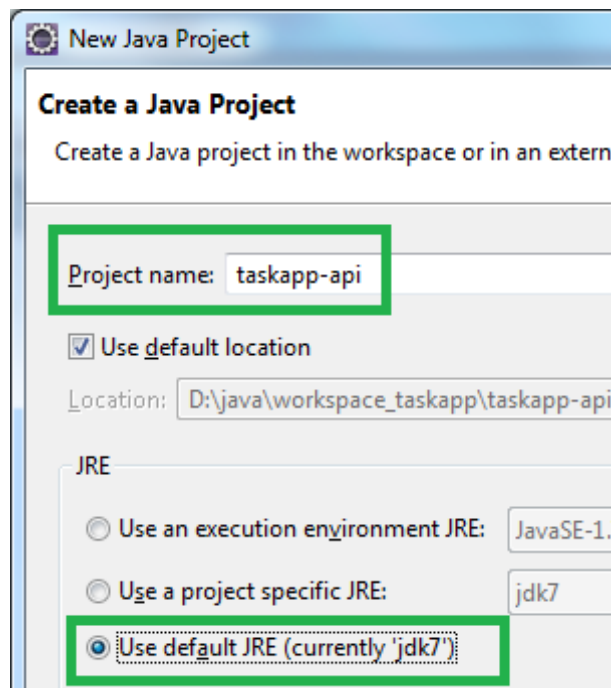
3.2.1 Create taskapp-api Project for the API Definition

This project contains the API definition and the ANT script for generating the serialization layer. It is referenced by all other projects.

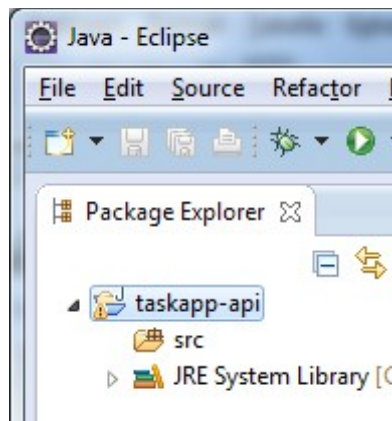
- Click “File – New – Project...”, select “Java Project”



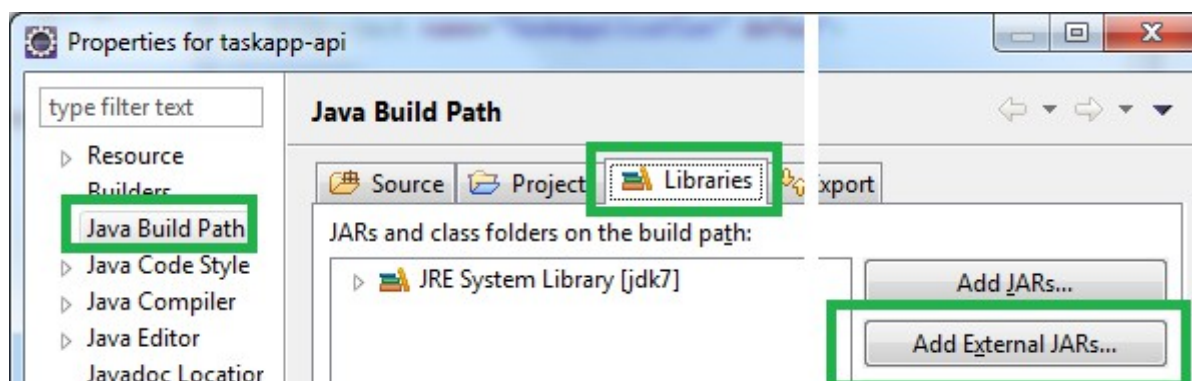
- Click “Next >”, enter “taskapp-api” as project name, ensure using the previously defined “jdk7”.



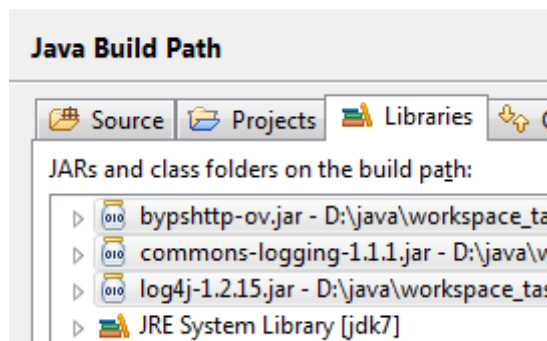
- “Finish”. Choose “Yes” to open the Java Perspective. The new project is shown in “Project Explorer”:



- In “Project Explorer” right-click on “taskapp-api”, “Properties”, select “Java Build Path”, select tab “Libraries”



- Click “Add External JARs”, from directory “workdir/byps-lib/java” select the JARs “bypshttp-ov.jar”, “commons-logging-1.1.1.jar”, “log4j-1.2.15.jar”:

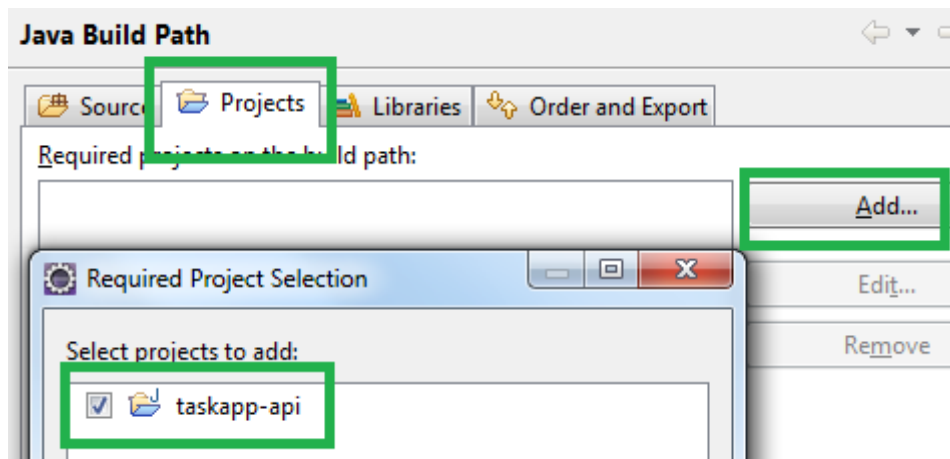


- “OK”.

3.2.2 Create taskapp-ser Project for Serialization Code

This project will receive the generated serialization code.

- Click “File – New – Other...”, select “Java Project”, “Next >”
- Enter “taskapp-ser” as project name, ensure using the default JRE “jdk7”.
- “Finish”
- In “Project Explorer” right-click on “taskapp-ser”, “Properties”, select “Java Build Path”, select tab “Projects”, “Add...”, select project “taskapp-api”:

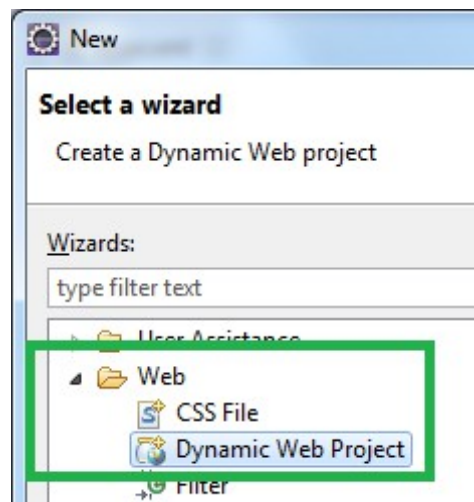


- Add the same JARs from “workdir/byps-lib/java” to the Java build path as done in 3.2.1 for the “taskapp-api” project.
- “OK”

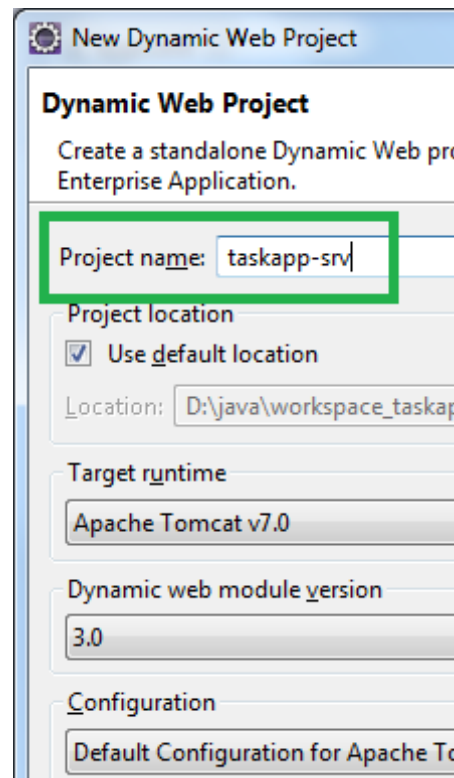
3.2.3 Create “taskapp-srv” Web Project

This project compiles the JEE web-service that serves the requests for the “taskapp” clients.

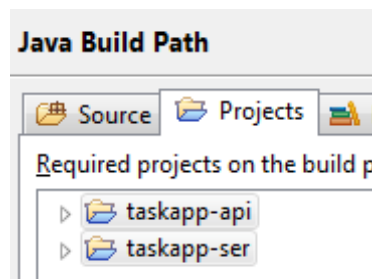
- Click “File – New – Other...”, select “Web – Dynamic Web Project”:



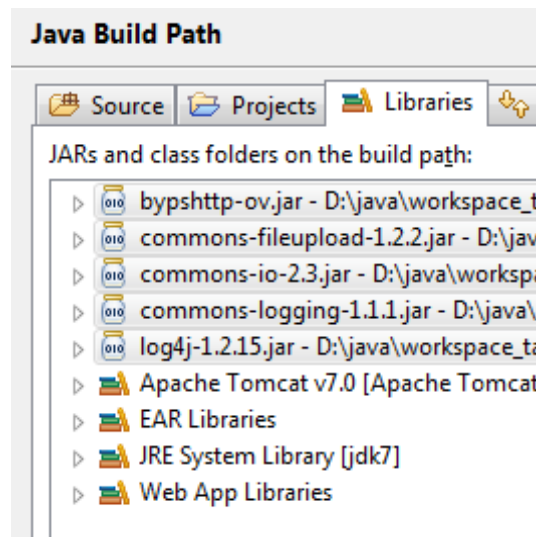
- “Next >”, enter project name “taskapp-srv”:



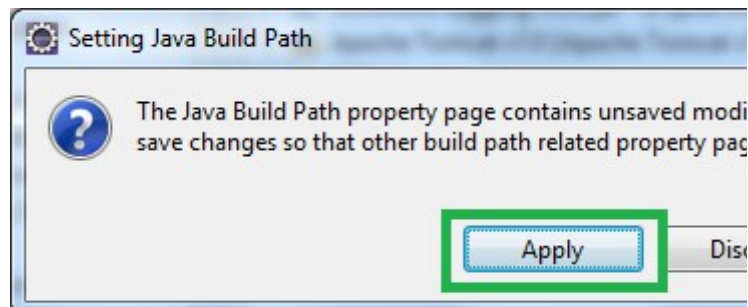
- “Finish”. When asked for changing the perspective to “Java EE” you may choose “No” if you are not familiar with Eclipse.
- In “Project Explorer” right-click “taskapp-srv” and choose “Properties”, select “Java Build Path”, select tab “Projects”, “Add...”, select project “taskapp-api” and “taskapp-ser”:



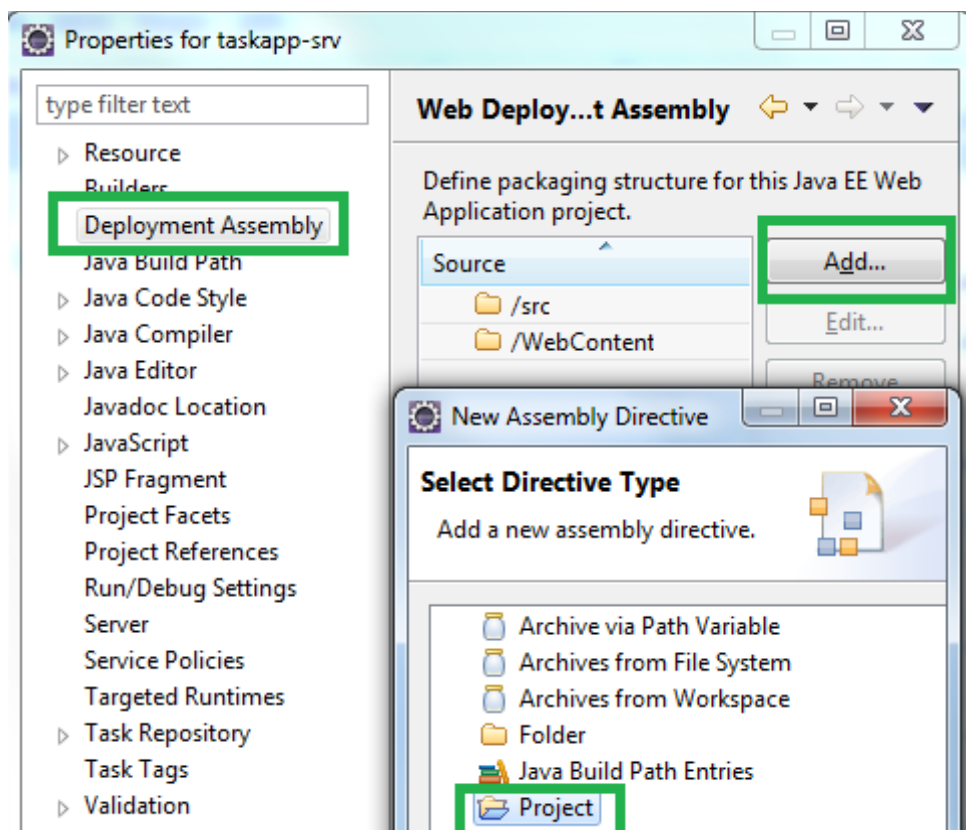
- Click tab “Libraries”, “Add External JARs...”, from “workdir/byps-lib/java” select the JARs: "bypshttp-ov.jar" "commons-fileupload-1.2.2.jar" "commons-io-2.3.jar" "commons-logging-1.1.1.jar" "log4j-1.2.15.jar":



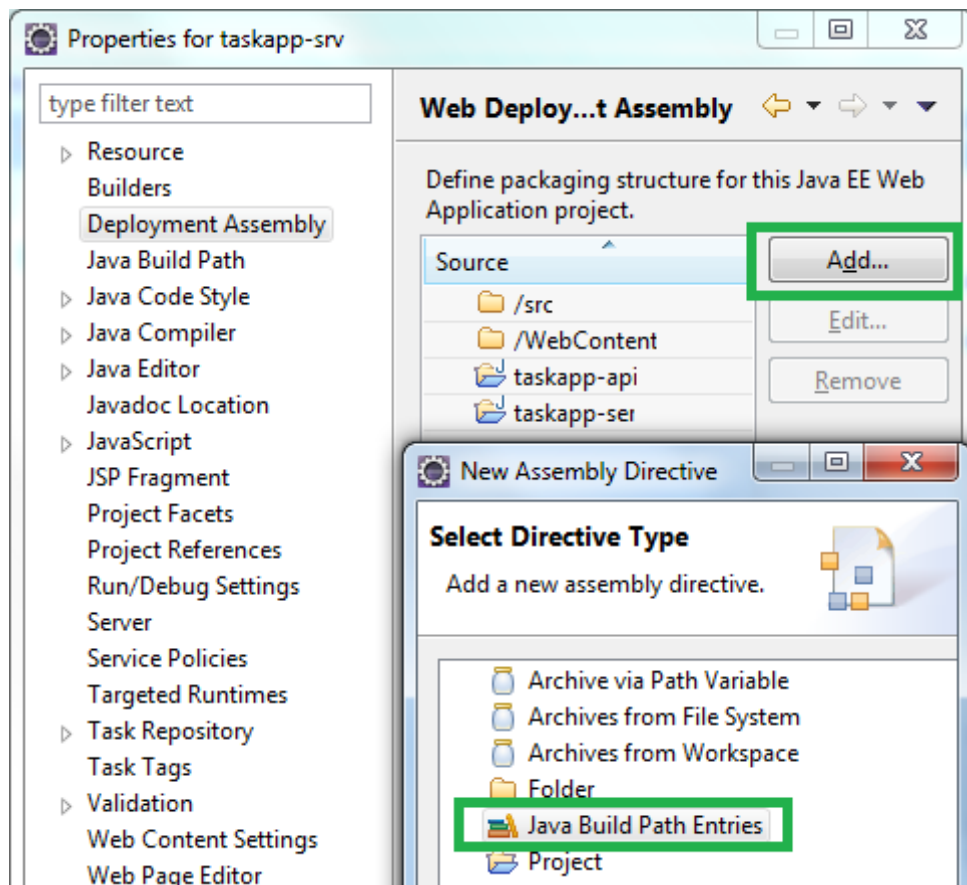
- Select inside the left tree “Deployment Assembly”, a message box is opened:



- Click “Apply” then “Add...” and select directive type “Project”:



- “Next >”, select “taskapp-api” and “taskapp-ser”, click “Finish” then “Add...”, select directive type “Java Build Path Entries”:



- “Next >”, select all entries, click “Finish”. Now, the listbox has the following entries:

Web Deployment Assembly	
Define packaging structure for this Java EE Web Application project.	
Source	Deploy Path
/src	WEB-INF/classes
/WebContent	/
bypshhttp-ov.jar - D:\java\w	WEB-INF/lib
commons-fileupload-1.2.2	WEB-INF/lib
commons-io-2.3.jar - D:\ja	WEB-INF/lib
commons-logging-1.1.1.ja	WEB-INF/lib
log4j-1.2.15.jar - D:\java\w	WEB-INF/lib
taskapp-api	WEB-INF/lib/taskapp-api.jar
taskapp-ser	WEB-INF/lib/taskapp-ser.jar

- “OK”.
- Create a source folder for the JSON serialization code that is generated in a late task. Therefore, in “Project Explorer” right-click on “taskapp-srv”, “New – Source Folder”.

3.2.4 Create “taskapp-client-j” Project, Java Client Application

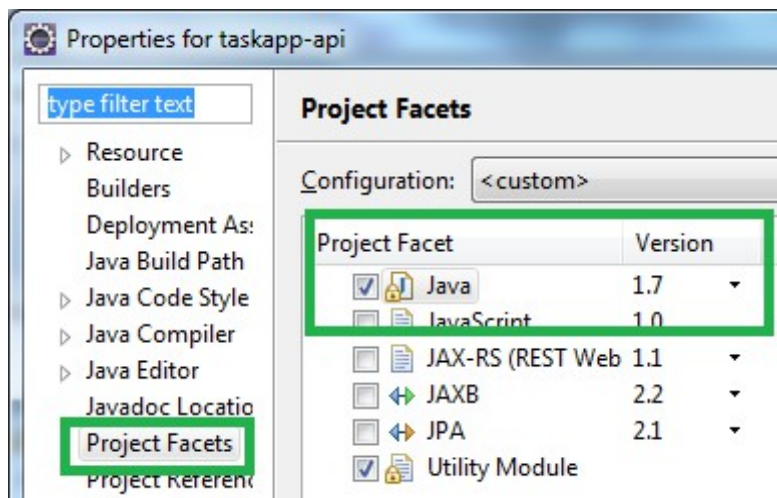
This project contains the code for a Java client application.

- Create a new project as shown in 3.2.1 . Use “taskapp-client-j” as project name.
- In the project properties under “Java Build Path”, add projects “taskapp-api” and “taskapp-ser” and external JARs: “bypshhttp-ov.jar”, “commons-logging-1.1.1.jar”, “log4j-1.2.15.jar”.

3.2.5 Check Eclipse Project Facets

Eclipse loves to make developers life more interesting. In order to prevent suspicious errors, check the “facets” configuration of all projects. Some projects might be configured to use the Java 1.4 facet.

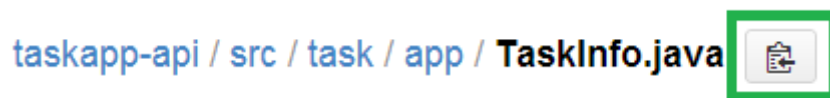
- In “Project Explorer” right-click on the project, select “Properties...”, select “Project Facets”, ensure the Java facet is set to a value greater than 1.4, e.g. choose 1.7:



3.3 Define “taskapp” API and Create Serialization Code

In this section, the interfaces and classes of the example API are created. Furthermore the serialization code is generated into the project “taskapp-ser”.

The source code files can be found at [TASKAPP]. From there, you may copy the file contents using the copy-to-clipboard button:



3.3.1 Interface “TaskService”

The main service interface is the “TaskService” interface.

- In “Project Explorer” expand “taskapp-api/src”, right-click “task.app”, “New – Interface”.
- Enter “TaskService” as name.
- “Finish”.

Replace the file contents with the following definition (which can be copied from

[BYP5]/doc/examples/workspace_taskapp/taskapp-api/src/task/app/TaskService.java”).

```
package task.app;

import java.util.List;

import byps.Remote;
import byps.RemoteException;

public interface TaskService extends Remote {

    void addTask(TaskInfo task) throws RemoteException;

    List<TaskInfo> getTasks() throws RemoteException;

}
```

Basically, all interfaces to be used with BYPS have to extend the “byps.Remote” interface and their methods have to throw a “byps.RemoteException”.

The “login” function authenticates a user and returns a “Token” object with a session identifier. The “Token” has to be passed in each of the other functions. In order to terminate a session, the client application calls “logout”.

A task is represented by an object of class “TaskInfo”. It and can be added to the list of tasks by calling the function “addTask”. The task list for the user can be obtained from “getTasks”.

3.3.2 Create class TaskInfo

Tasks are stored in objects of type “TaskInfo”. Create the class as follows.

- In “Project Explorer” expand “taskapp-api/src”, right-click “task.app”, “New – Class”.
- Enter “TaskInfo” as name.
- “Finish”.

Replace the file with this code (which can be copied from [BYP5]/doc/examples/workspace_taskapp/taskapp-api/src/task/app/TaskInfo.java)

```
package task.app;

import java.io.Serializable;
import java.util.Date;

public class TaskInfo implements Serializable {

    private static final long serialVersionUID = 3264907623228026167L;

    protected int id;

    protected String userName;

    protected Date dueDate;

    protected String todo;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUserName() {
        return userName;
    }

}
```

```

    public void setUsername(String userName) {
        this.userName = userName;
    }

    public Date getDueDate() {
        return dueDate;
    }

    public void setDueDate(Date dueDate) {
        this.dueDate = dueDate;
    }

    public String getTodo() {
        return todo;
    }

    public void setTodo(String todo) {
        this.todo = todo;
    }
}

```

Classes must implement the interface “`java.io.Serializable`” and define a static member named “`serialVersionUID`” with a unique value. This value is internally used as a type identifier for the class, whereby only the lowest 32bit are taken into account.

As already mentioned, **class members can only be variables** or trivial getter and setter functions (classes are DTOs, Data Transfer Objects).

If a data member is declared “private”, a getter and setter function must be declared. Otherwise, access functions are optional. It is recommended to use “protected” over “private” because then, the serialization functions can bypass the calls to the access functions.

3.3.3 Create the API Descriptor Class “BApi”

Each BYPS API has to include one class named “BApi” in the source tree that defines a name and a version.

- In “Project Explorer” expand “taskapp-api/src”, right-click “task.app”, “New – Class”.
- Enter “BApi” as name.
- “Finish”.

Replace the code with the following definition:

```

package task.app;

public class BApi {

    public static final String NAME = "Taskapp";

    public static final String VERSION = "1.0.0.0";
}

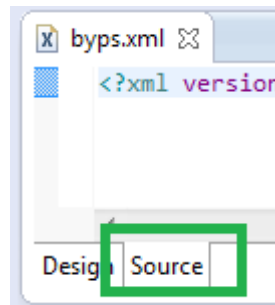
```

3.3.4 Create ANT Script “byps.xml” for Generator

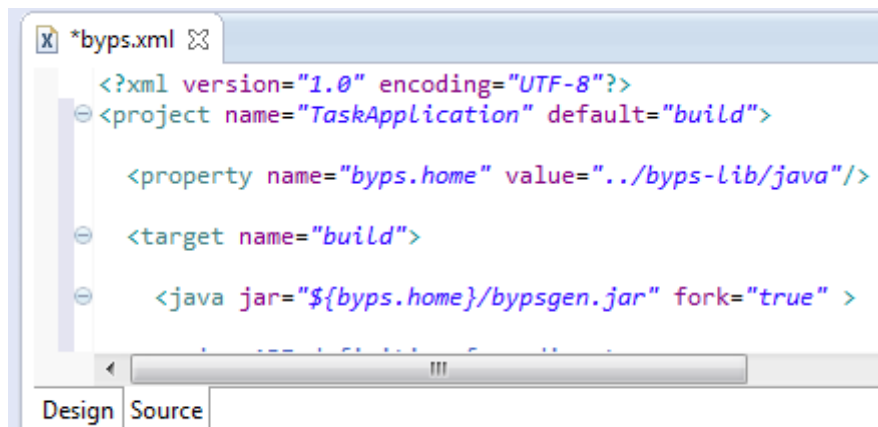
This section creates the ANT script that invokes the BYPS code generator.

- In “Project Explorer” right-click on “taskapp-api”, “New – Other...”, select “XML File”:
- Enter “byps.xml” as file name

- “Finish”
- Switch editor mode to “Source”



- Replace the XML code in the generated file with the ANT script from section 1.2 (or copy from [BYPs]/doc/examples/workspace_taskapp/taskapp-api/byps.xml).

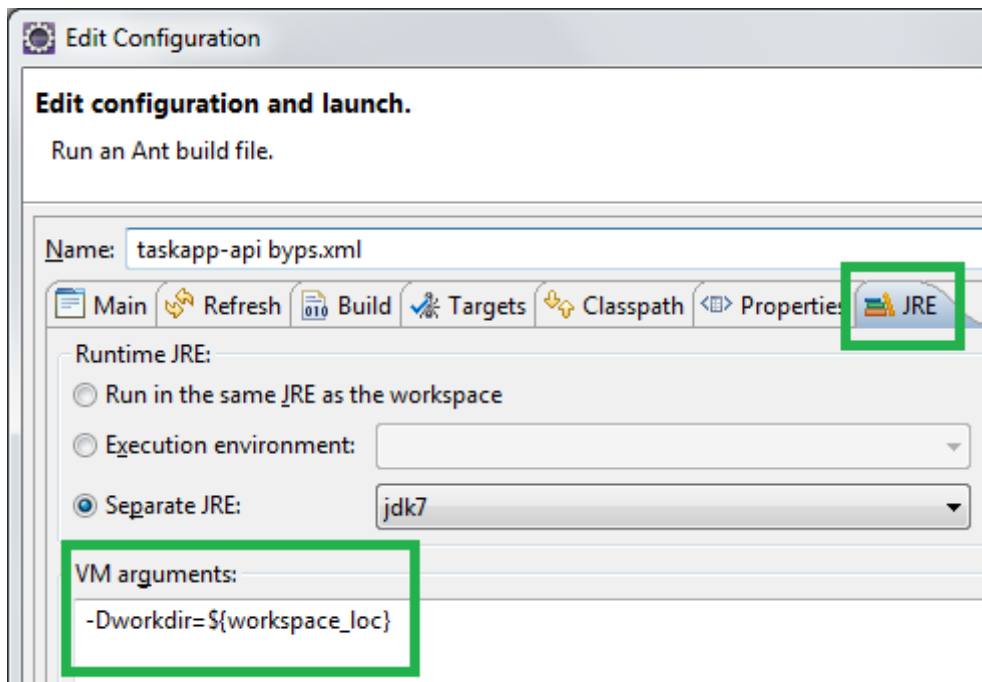


- Save the file, CTRL-S.

3.3.5 Run the BYPS Generator

At this step, the BYPS generator can be executed. But before this can be done, a configuration option must be set to pass the workspace directory to the generator.

- In “Project Explorer” expand “taskapp-api” and right-click on “byps.xml”.
- Select “Run As – Ant Build...”
- Select tab “JRE”
- Add as “VM arguments”: -Dworkdir=\${workspace_loc}



- “Apply”
- “Close”

Now, the Eclipse is prepared to pass the workspace directory to the generator.

- Execute the generator: Right-click “byps.xml”, select “Run As – Ant Build” (without dots).
- The “Console” view should look like this:

```

<terminated> taskapp-api byps.xml [Ant Build] D:\Program Files\Java\jdk1.7.0
Buildfile: D:\git\byps\doc\examples\workspace taskapp\task
build:
    [java] Loading source files for package task.app...
    [java] Constructing Javadoc information...
    [java] Finished
BUILD SUCCESSFUL
Total time: 5 seconds

```

- The generated source files have been written into the “taskapp-ser” project directory. Refresh the project: Right-click “taskapp-ser”, select “Refresh”.

The generator has created Java source files for binary serialization in directory “taskapp-ser/src”.

Sources for C++ and C# have been written into “taskapp-ser/src-cpp” respectively “taskapp-ser/src-cs”.

The JavaScript serialization code can be found in the web application project in file “taskapp-srv/WebContent/taskapp.js”. The corresponding JSON serialization code in Java has been written into “taskapp-srv/src-ser-json”.

3.4 Implement the Web-Service “taskapp-srv”

There are two key classes on the server-side for BYPS applications. The first one is the JEE servlet class which receives the HTTP requests and creates objects of the second class: the session class. This class holds the implementations of the server-side interfaces. Both classes are very similar for all applications.

For this example, one server-side interface has to be implemented: the “TaskService” interface. In general, an implementation must inherit from the generated skeleton class of this interface. The skeleton class for “TaskService” is the class “BSkeleton_TaskService”.

The next sections create this three classes ordered by their dependency: 1. “TaskService” implementation, 2. Session class “TaskappSession”, and 3. Servlet class “TaskappServlet”.

3.4.1 Service implementation class TaskServiceImpl

The “TaskServiceImpl” class provides the server-side functionality for the “TaskService” interface. Implementation classes have to be inherited from the generated skeleton class associated to the interface, which is by convention named “BSkeleton” + underscore + interface name.

- In “Project Explorer” expand “taskapp-srv/Java Resources/src”, right-click “task.app”, “New – Class...”
- Enter class name “TaskServiceImpl”
- Click “Finish”.

Replace the generated code with the following lines:

```
package task.app.srv;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import task.app.BSkeleton_TaskService;
import task.app.TaskInfo;
import byps.RemoteException;

public class TaskServiceImpl extends BSkeleton_TaskService {

    private String userName;
    private static HashMap<String, ArrayList<TaskInfo>> tasksOfAllUsers =
        new HashMap<String, ArrayList<TaskInfo>>();

    public TaskServiceImpl() {
        this.userName = "Fritz";
    }

    @Override
    public void addTask(TaskInfo task) throws RemoteException {
        synchronized(tasksOfAllUsers) {
            ArrayList<TaskInfo> tasksOfUser = tasksOfAllUsers.get(task.getUserName());
            if (tasksOfUser == null) {
                tasksOfAllUsers.put(task.getUserName(),
                    tasksOfUser = new ArrayList<TaskInfo>());
            }
            tasksOfUser.add(task);
        }
    }

    @Override
    public List<TaskInfo> getTasks() throws RemoteException {
        synchronized(tasksOfAllUsers) {
            ArrayList<TaskInfo> tasksOfUser = tasksOfAllUsers.get(userName);
            if (tasksOfUser == null) {
                tasksOfAllUsers.put(this.userName,
                    tasksOfUser = new ArrayList<TaskInfo>());
            }
            return tasksOfUser;
        }
    }
}
```

```
}  
}
```

3.4.2 Session class “TaskappSession”

The following steps create the session class. Objects of this class are tied to a user session and are maintained by the application servers session map.

Session classes have to be derived from `byps.http.HSession`.

- In “Project Explorer” expand “taskapp-srv/Java Resources/src”, right-click “task.app”, “New – Class...”
- Enter class name “TaskappSession”, enter superclass name “byps.http.HSession”.
- Click “Finish”.
- Right-click somewhere in the editor window, “Source – Generate Constructors from Superclass”, check only “HSession(HttpSession, String, File, BServerRegistry)”, choose “First member” as insertion point.
- Click “OK”. This lines are generated:

```
public TaskappSession(HttpSession hsess, String remoteUser, HServerContext serverContext) {  
    super(hsess, remoteUser, serverContext);  
    // TODO Auto-generated constructor stub  
}
```

- The server-side BYPS framework will ask the session class for a BServer object that provides the implementations of the server-side interfaces. The generator has already created a class `BServer_Testser` class, inherited from `BServer`, that is to be used in this case. Create a member variable in the session class for an object of type `BServer_Testser`. Since this member will be initialized in the constructor and does never change it can be declared as final. Press CTRL-SHIFT-O in order to let Eclipse add the import directive for the `BServer_Testser` class.

```
private final BServer_Taskapp bserver;
```

- Let the getter “getServer()” return the member “bserver”.

```
@Override  
public BServer getServer() {  
    return bserver;  
}
```

- The next steps initialize the server-side serialization framework. At first, an extended API descriptor object has to be created that allows [JSON] serialization. The generated object does only provide binary serialization. Since the Servlet class also needs this object and it is equal for all sessions, the object is declared as a final static member:

```
// Append the JSON serializer registry to the API descriptor  
public final static BApiDescriptor apiDesc =  
    BApiDescriptor_Taskapp.instance()  
        .addRegistry(new JRegistry_Taskapp());
```

- Inside the body of the constructor delete the TODO line and insert the following lines.

```
// Create the BServer object  
BTransportFactory transportFactory = getTransportFactory(apiDesc);  
bserver = BServer_Taskapp.createServer(transportFactory);
```

```
// Add the interfaces you want to implement
bserver.addRemote(new TaskServiceImpl(remoteUser));

// Mark session valid
setSessionAuthenticated();
```

The first lines create the BServer object and initialize it with a BTransportFactory which is responsible for receiving and de-serializing method calls.

The second step adds the interface implementations provided by the server. Here, an object of the TaskServiceImpl class is passed, which was created in 3.4.1. As a constructor parameter, the current user name is supplied. This value is the result of a successful HTTP authentication and is obtained from the getRemoteUser function of the HttpServletRequest object.

The last block declares the session as valid. Before this, the underlying HTTP session is only valid for 10 seconds.

The entire class definition looks like this and can also be found at [TASKAPP].

```
package task.app.srv;

import java.io.File;

import javax.servlet.http.HttpSession;

import task.app.BApiDescriptor_Taskapp;
import task.app.BServer_Taskapp;
import task.app.JRegistry_Taskapp;
import byps.BApiDescriptor;
import byps.BServer;
import byps.BServerRegistry;
import byps.BTransportFactory;
import byps.http.HSession;

public class TaskappSession extends HSession {

    private final BServer_Taskapp bserver;

    // API descriptor with JSON serializers
    public final static BApiDescriptor apiDesc =
        BApiDescriptor_Taskapp.instance()
            .addRegistry(new JRegistry_Taskapp());

    public TaskappSession(HttpSession hsess, String remoteUser, HserverContext serverContext) {
        super(hsess, remoteUser, serverContext);

        // Create the BServer object
        BTransportFactory transportFactory = getTransportFactory(apiDesc);
        bserver = BServer_Taskapp.createServer(transportFactory);

        // Add the interfaces you want to implement
        bserver.addRemote(new TaskServiceImpl(this));

        // Mark session valid
        setSessionAuthenticated();
    }

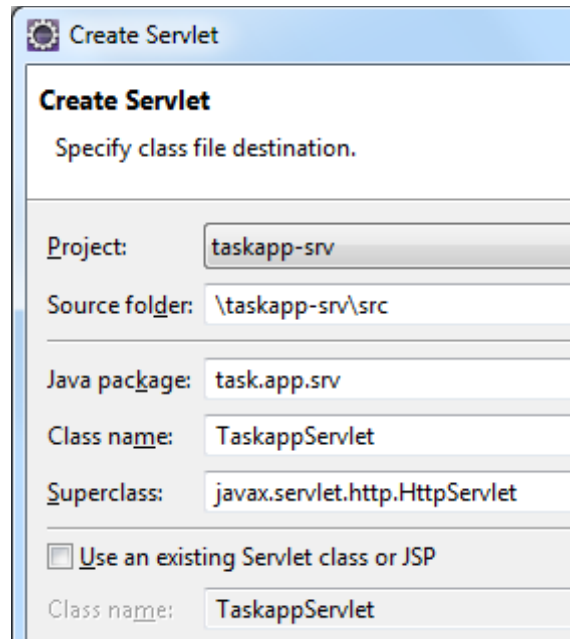
    @Override
    public BServer getServer() {
        return bserver;
    }
}
```

3.4.3 Servlet class TaskappServlet

The main purpose of this Servlet class is to create session objects. Furthermore it has to provide the configuration parameters required by the server-side framework.

- In “Project Explorer” right-click project “taskapp-srv”, select “New – Other...”

- Select wizard “Web/Servlet”, click “Next >”
- Enter “task.app.srv” as “Java package”
- Enter “TaskappServlet” as “Class name”

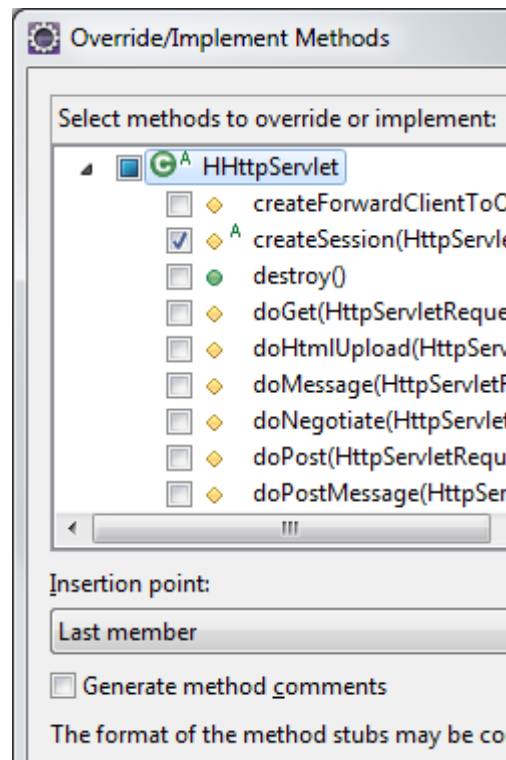


- Click “Finish”
- Eclipse opens an editor window for the new class. Delete functions “doGet” and “doPost” in the class.
- Remove the annotation `@WebServlet("/TaskappServlet")` and replace it by the following lines:

```
@WebServlet(
    // mandatory: must be true
    asyncSupported = true,

    // mandatory: server URL patterns
    urlPatterns = { "/taskapp", "/taskapptauth/auth" }
)
```

- Replace the base class “HttpServlet” by “HHttpServlet”
- Press “CTRL-SHIFT-O” (or context menu “Source – Organize Imports”) to let Eclipse add the missing import statement.
- Right-click somewhere in the editor window and select “Source – Override/Implement Methods...”
- The methods to be implemented are already selected since they are abstract methods in the base class “HHttpServlet”.



- Click “OK”. Eclipse generates method stubs for the functions “createSession”, “getApiDescriptor”, and “getConfig”.
- For providing configuration parameters for the BYPS server-side framework, the helper class “HConfigImpl” can be used. Add an instance of this class as a member variable to the Servlet. Press CTRL-SHIFT-O to add the required import statement. Let the function “getConfig” return this object.

```
private final HConfigImpl config = new HconfigImpl();
...
@Override
public HConfig getConfig() {
    return config;
}
```

- Add the code for creating session objects:

```
@Override
protected HttpSession createSession(HttpServletRequest request,
    HttpServletResponse response, HttpSession hsess) throws BException {
    String remoteUser = request.getRemoteUser();
    return new TaskappSession(hsess, remoteUser, this);
}
```

- Implement the “getApiDescriptor” function:

```
@Override
protected BApiDescriptor getApiDescriptor() {
    return TaskappSession.apiDesc;
}
```

3.4.4 Add a Start Page to the Web Application

Usually, a web application has a default page that is displayed, if no URI path into the web application is specified. The next steps add such a default page to “taskapp-srv”.

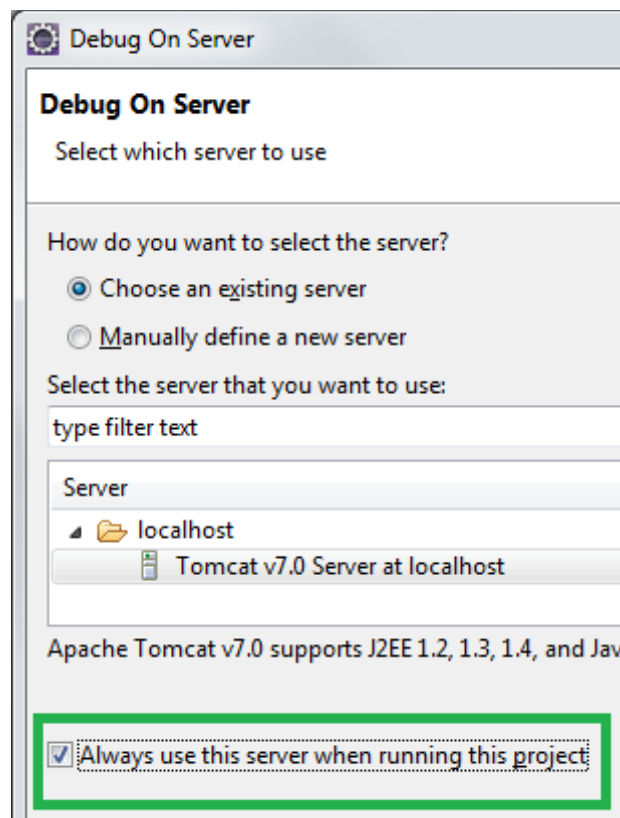
- In “Project Explorer” expand “taskapp-srv” and right-click “WebContent”.
- Select “New – HTML File”
- Enter file name “index.html”
- Click “Finish”
- Modify as follows and save.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Taskapp</title>
</head>
<body>
<a href="taskapp">Status</a>
</body>
</html>
```

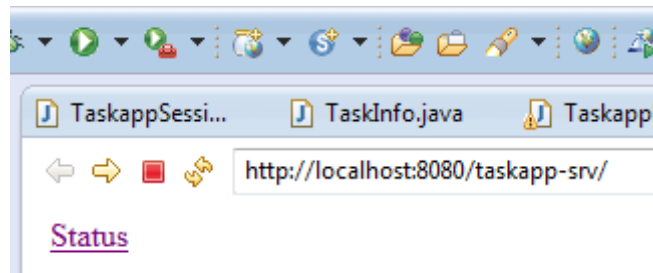
3.4.5 Start Server

The server should be started now to make a simple check.

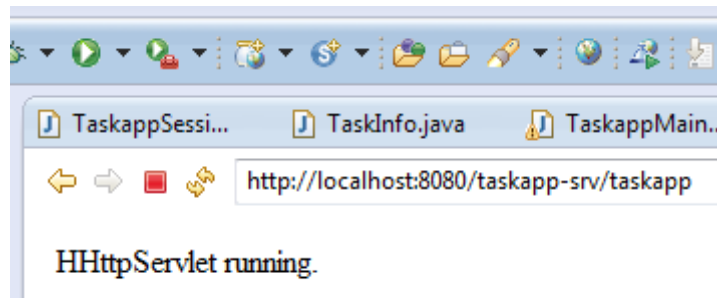
- In “Project Explorer” right-click “taskapp-srv”, “Debug As – Debug On Server”
- The “Debug On Server” Dialog is opened, check “Always use this server...”



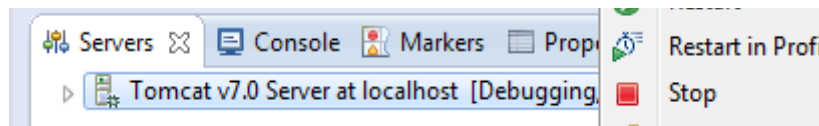
- Click “Finish”
- Eclipse should display the index.html inside an integrated browser window:



- Click “Status”, the browser navigates to our Servlet and displays a simple status text:



- In order to stop the server, select the view “Servers”, right-click the “Tomcat ...” entry and select stop.



3.5 Implement the Java Client Application *taskapp-client-j*

This section creates a very simple client application without a user interface.

- In project “taskapp-client-j” create class “TestappMain” in package “task.app.client”
- Create the entry point function for the java application: the “main” function:

```
public final static void main(String[] args) {

    String url = "http://localhost:8080/taskapp-srv/taskapp";
    BClient_Taskapp bclient = createClient(url);

    try {
        bclient.start();
        System.out.println("connected, targetId=" +
            bclient.getTransport().getTargetId());

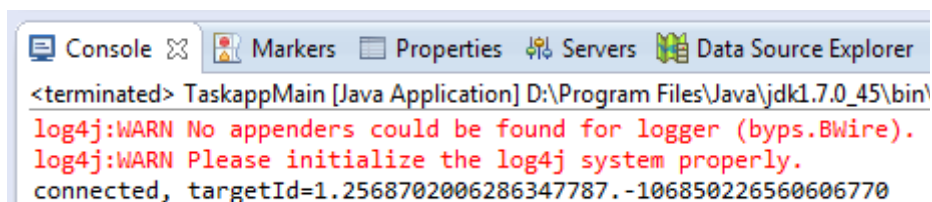
        String command = args.length > 0 ? args[0] : "";
        if (command.equals("-add")) {
            addTask(bclient, args);
        }
        else if (command.equals("-list")) {
            listTasks(bclient);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    finally {
        if (bclient != null) {
            bclient.done();
        }
    }
}
```

```
}
```

- Press CTRL-SHIFT-O to let Eclipse add the required import directives.
- Create function “createClient”. It initializes a BClient_Taskapp object.

```
private static BClient_Taskapp createClient(String url) throws RemoteException {  
    BWire wire = new HWireClient(url, 0, 120, null);  
    BTransportFactory tfact =  
        new HTransportFactoryClient(BApiDescriptor_Taskapp.instance(), wire, 1);  
    BClient_Taskapp bclient = BClient_Taskapp.createClient(tfact);  
    return bclient;  
}
```

- This object holds the servers interfaces. The HWireClient object is responsible for the HTTP connections and sends and receives the request. BTransportFactory is used to serialize and de-serialize the messages with the negotiated protocol. The “bclient.start()” call (see function “main”) performs the protocol negotiation and the authentication. Furthermore it starts a long-poll request to listen for server-push requests.
- Make sure that the server is running and start the client application: in “Project Explorer” expand “taskapp-client-j/src/task.app.client” and right-click on “TaskappMain.java”. Select “Debug As – Java Application”.
- The console window should print something like this:



```
<terminated> TaskappMain [Java Application] D:\Program Files\Java\jdk1.7.0_45\bin\  
log4j:WARN No appenders could be found for logger (byps.BWire).  
log4j:WARN Please initialize the log4j system properly.  
connected, targetId=1.2568702006286347787.-106850226560606770
```

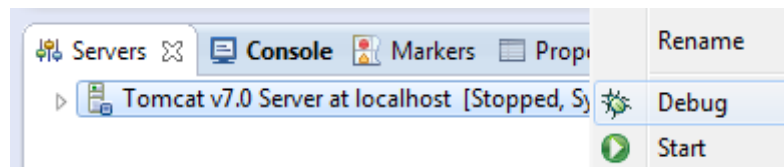
- Add the functions for adding, removing and listing tasks:

```
private static void addTask(BClient_Taskapp bclient, String[] args) throws RemoteException {  
    TaskInfo t = new TaskInfo();  
    t.setId((int) (System.currentTimeMillis() % 1000));  
    t.setUsername(args[1]);  
    t.setTodo(args[2]);  
    t.setDueDate(new java.util.Date(System.currentTimeMillis() + 24*60*60*1000));  
    bclient.getTaskService().addTask(t);  
}  
  
private static void listTasks(BClient_Taskapp bclient) throws RemoteException {  
    List<TaskInfo> tasks = bclient.getTaskService().getTasks();  
    for (TaskInfo t : tasks) {  
        System.out.println("id=" + t.getId() +  
            "\tuserName=" + t.getUsername() +  
            "\ttodo=" + t.getTodo());  
    }  
}
```

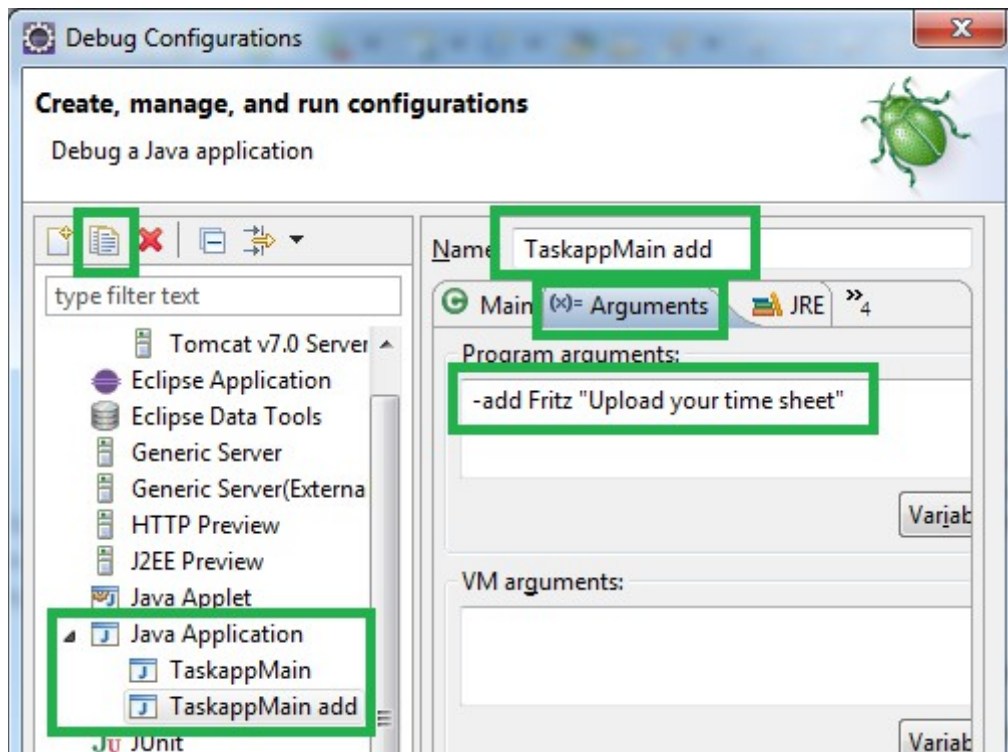
- Remove the comment slashes of the lines after “connect()”. Therefore select the lines and press CTRL-SHIFT-7

3.5.1 Add a Debug Configuration for Each Action

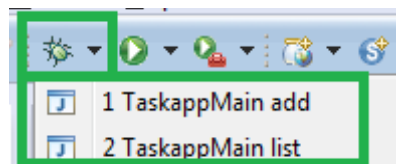
- Start the Tomcat server in the “Servers” view: right-click “Tomcat...” entry and select “Debug”:



- In order to add a debug configuration to test each of the functions, right-click project “taskapp-client-j”, select “Debug As – Debug Configurations...”.
- In the left tree select “Java Application / TaskappMain”, click the copy button in the toolbar, rename the new configuration to “TaskappMain add”, select tab “Arguments”, enter as “Program arguments”: -add Fritz... as shown in the screenshot:



- Press “Apply” then “Debug”
- Create also debug configurations for the action “list”. It has to be started with the arguments “-list”.
- When a debug configuration has been executed once, it is inserted into the drop down menu of the debug button in the toolbar. From there it could be started easier:



- Now, you are well prepared to step through the code and see how the BYPS framework works.

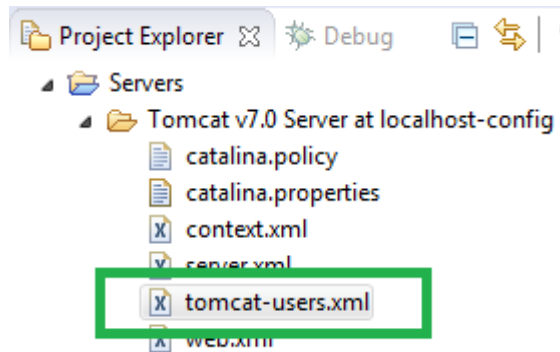
3.6 Using HTTP Authentication

JEE servlet engines (like Tomcat) usually implement authentication by the use of servlet filters. Since the entry point for the server-side of the BYPS framework is a JEE servlet, servlet filters can also be used here in order to constrain access to the application.

Although servlet filters are the standard way to implement authentication, the next example uses the Tomcat owned “Basic” authentication, since it does not require any additional library.

In the first step, a “users database” has to be set up in the server's “tomcat-users.xml” file.

- In “Project Explorer” expand “Servers – Tomcat ...”, open “tomcat-users.xml”

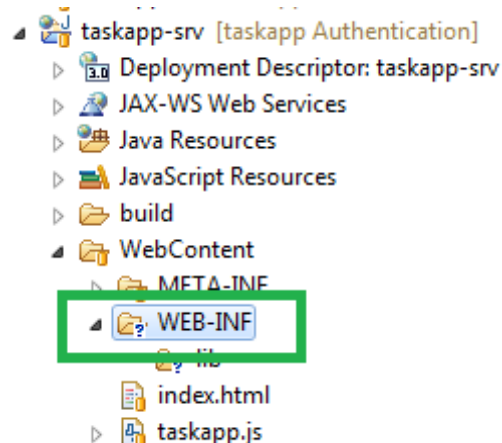


- Inside the “<tomcat-users>” element add a role and a user:

```
<role rolename="Taskapp Role" />
<user username="Fritz" password="abc" roles="Taskapp Role"/>
```

Access to the “TaskappServlet” must be constrained to users of role “Taskapp Role”:

- In “Project Explorer” expand “taskapp-srv – WebContent”, right-click “WEB-INF”



- Select “New – Other...”, expand “XML”, select “XML File”, click “Next >”
- Enter file name “web.xml”, click “Finish”
- Replace the contents of “web.xml” with this lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://xmlns.jcp.org/xml/ns/javaee">
  <display-name>Taskapp Application</display-name>
  <servlet>
    <servlet-name>Taskapp Servlet</servlet-name>
    <servlet-class>task.app.srv.TaskappServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
```

```

    <servlet-name>Taskapp Servlet</servlet-name>
    <url-pattern>/taskappauth/auth</url-pattern>
</servlet-mapping>
<security-constraint>
    <display-name>Taskapp Security Constraint</display-name>
    <web-resource-collection>
        <web-resource-name>Taskapp Session Resource</web-resource-name>
        <url-pattern>/taskappauth/auth</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Taskapp Role</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Taskapp Realm</realm-name>
</login-config>
</web-app>

```

- Save “web.xml” and start the server. If any errors occur in the console window, see 4 .
- Check that authentication is applied: start a browser and navigate to <http://localhost:8080/taskapp-srv/taskappauth/auth>
- A login dialog should be displayed, check that you can login with the user defined in the “tomcat-users.xml” file.

The servlet class must refuse an authentication request without a “remoteUser”:

- Edit “TaskappServlet”, modify function “createSession”, throw an exception if the “remoteUser” is not set:

```

@Override
protected HttpSession createSession(HttpServletRequest request,
    HttpServletResponse response, HttpSession hsession,
    BServerRegistry serverRegistry) throws BException {

    String remoteUser = request.getRemoteUser();
    if (remoteUser == null) throw new BException(BExceptionC.FORBIDDEN,
        "Missing remote user attribute.");
    return new TaskappSession(hsession, remoteUser, null, serverRegistry);
}

```

- Add a parameter to the constructor of “TaskappServiceImpl” for the session object and obtain the user name from the session:

```

public TaskServiceImpl(TaskappSession sess) {
    this.userName = sess.getRemoteUser();
}

```

- Pass the session object in the constructor of “TaskappSession”:

```

// Add the interfaces you want to implement
bserver.addRemote(new TaskServiceImpl(this));

```

The client application has to be prepared for handling HTTP authentication:

- Create a function to the “TaskappMain” class that sets a “java.net.Authenticator”:

```

private static void setCredentials(final String userName, final String userPwd) {
    class MyAuthenticator extends Authenticator {
        public PasswordAuthentication getPasswordAuthentication () {
            return new PasswordAuthentication (userName, userPwd.toCharArray());
        }
    }
    Authenticator.setDefault(new MyAuthenticator());
}

```

- Invoke this function before the BClient object is started and pass the user credentials defined in the “tomcat-users.xml” file:

```
public final static void main(String[] args) {

    String url = "http://localhost:8080/taskapp-srv/taskapp";
    BClient_Taskapp bclient = createClient(url);

    try {
        setCredentials("Fritz", "abc");

        bclient.start();
    }
}
```

- In function “createClient”, supply the servlet path for HTTP authentication. The “<url-pattern>” defined in the “web.xml” file above has to be used here.

```
private static BClient_Taskapp createClient(String url) {

    BWire wire = new HWireClient(url, 0, 120, null) {
        @Override
        public String getServletPathForNegotiationAndAuthentication() {
            return "/taskappauth/auth";
        }
    };

    BTransportFactory tfact = new HTransportFactoryClient(
        BApiDescriptor_Taskapp.instance(), wire, 1);
    BClient_Taskapp bclient = BClient_Taskapp.createClient(tfact);
    return bclient;
}
```

3.7 Versioning

Now imagine, that the “taskapp-client-j” has been distributed on many client computers and the API has been integrated into other applications as well. A new feature request requires to add a list of properties (as key-value-pairs) to the “TaskInfo” class. Just adding new members to the class would break compatibility to existing client applications and you would have to update all this applications before they could connect to the server again.

In order to handle this scenario by supporting compatibility with older (or even newer) client applications, BYPS supports a versioning mechanism. When client and sever start communicating, a protocol version is negotiated and only data members of this protocol version are transferred between the applications. Therefore, the data members have to be tagged with the version since when they are available. The “Javadoc” tool already defines a tag for such purposes: the “@since” tag.

Hence, adding new data members require the following steps:

- Increment the version number in BAPI class.
- Add new member(s) and tag them with “@since <new version number>”

Experience has shown, that programmers sometimes forget to add the “@since” tag to new members or incrementing the BAPI version number. In order to prevent creating incompatible interfaces, the BYPS generator can check the modifications against the previous version of the API.

The generator looks for the previous definitions in a file named “bapi.xml”. Each time the generator runs it generates a “bapi_new.xml”. If your API is released, the file should be renamed to “bapi.xml” and checked in to your source code control system.

Below it is shown how to add a new member to the “TaskInfo” class. The code can be found on [TASKAPP] in branch “Versioning”.

- In class “BApi” change the version number:

```
public static final String VERSION = "1.0.0.1";
```

- In class “TaskInfo” add a new member as follows:

```
/**
 * @since 1.0.0.1
 */
protected HashMap<String, String> properties;
```

- Press CTRL-SHIFT-O to add required imports
- Add getter and setter methods: right-click in editor window, “Source – Generate Getters and Setters”, select member “properties”, click “OK”.
- Save all files: CTRL-SHIFT-S
- Run the generator: right-click “byps.xml”, “Run As – Ant Build”
- Refresh projects “taskapp-ser” and “taskapp-srv”: right-click project, “Refresh”
- Add some code to set the new member in “TaskappMain” class of project “taskapp-client-j”, e.g.:

```
private static void addTask(BClient_Taskapp bclient, String[] args) throws RemoteException {
    TaskInfo t = new TaskInfo();
    t.setId((int) (System.currentTimeMillis() % 1000));
    t.setUsername(args[1]);
    t.setTodo(args[2]);
    t.setDueDate(new java.util.Date(System.currentTimeMillis() +
        24*60*60*1000));

    HashMap<String,String> properties = new HashMap<String,String>();
    properties.put("INVOICE", "12345");
    properties.put("CUSTOMER", "67890");
    t.setProperties(properties);

    bclient.getTaskService().addTask(t);
}
```

- Modify function “getTasks” to print the member “properties”, e.g:

```
private static void getTasks(BClient_Taskapp bclient) throws RemoteException {
    List<TaskInfo> tasks = bclient.getTaskService().getTasks();
    System.out.println("id\tuser\ttodo");
    for (TaskInfo t : tasks) {
        System.out.println("id=" + t.getId() +
            "\tuserName=" + t.getUsername() +
            "\ttoDo=" + t.getTodo() +
            "\tproperties=" + t.getProperties());
    }
}
```

3.8 Streams

The BYPS communication layer allows to use streams as data members, function parameters and return values. A stream can be up to $2^{63}-1$ bytes and is transferred in a separate request. When sending a stream from the client to the server, it requires its own connection to the server. Thus in order to invoke a method with a stream parameter, two connections are opened since the method

data and stream data is sent in parallel.

If you receive a stream by a method, a connection is required at the moment you start reading from the stream. A stream can only be read once and the server closes the stream if the client application does not read it for some minutes – defined in project “bypshttp”, “byps.http.HConstants.REQUEST_TIMEOUT_MILLIS”.

The next example shows how to work with streams. The code can be found at [TASKAPP] in branch “Streams”.

- Add a new member to the “TaskInfo” class as follows:

```
/**
 * @since 1.0.0.1
 */
protected List<InputStream> attachments;
```

- Add imports, add a getter and setter function (java.io.InputStream, java.util.List).
- Run the generator: right-click on “byps.xml”
- Refresh projects “taskapp-ser” and “taskapp-srv”: right-click project, “Refresh”
- In the client application, add a function “addAttachments” to the “TaskappMain” class as shown below. The supplied streams are automatically closed after they have been sent to the server. Call this function inside “addTask”.

```
private static ArrayList<InputStream> addAttachments(TaskInfo t) {
    ArrayList<InputStream> attachments = new ArrayList<InputStream>();
    try {
        BContentStream strml = new BContentStreamWrapper(new File(".project"));
        strml.setContentType("text/plain");
        BContentStream strml2 = new BContentStreamWrapper(
            new ByteArrayInputStream("Text as stream".getBytes()));
        strml2.setContentType("text/plain");
        attachments.add(strml);
        attachments.add(strml2);
        t.setAttachments(attachments);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return attachments;
}

private static void addTask(BClient_Taskapp bclient, String[] args) throws RemoteException {
    TaskInfo t = new TaskInfo();
    ...
    addAttachments(t);
    ...
    bclient.getTaskService().addTask(t);
}
```

- Add this lines in the for-loop of “listTasks” in order to print some information about the received streams:

```
try {
    System.out.print("--- attachments: ");
    for (InputStream s : t.getAttachments()) {
        BContentStream bs = (BContentStream)s;
        System.out.print("{ " + bs.getContentType() +
            ", #" + bs.getContentLength() + ", [");
        for (int i = 0; i < 10; i++) {
            System.out.print((char)bs.read());
        }
        System.out.print("}], ");
    }
    System.out.println();
}
```



```

        catch (IOException e) {
            e.printStackTrace();
        }
    }

```

- On the server-side, the function “addTask” of class “TaskServiceImpl” has to be modified. The streams inside the “Task” object passed to the function are only valid during the function call. They have to be duplicated in order to store them. For this reason, a copy function is introduced that clones a “Task” object.

```

private TaskInfo cloneTask(TaskInfo t) throws RemoteException {
    TaskInfo n = new TaskInfo();
    n.setId(t.getId());
    n.setUserName(t.getUserName());
    n.setDueDate(t.getDueDate());
    n.setTodo(t.getTodo());

    List<InputStream> istreams = t.getAttachments();
    if (istreams != null) {
        List<InputStream> nstreams = new ArrayList<InputStream>(istreams.size());
        n.setAttachments(nstreams);
        for (InputStream is : istreams) {
            BContentStream ns;
            try {
                ns = ((BContentStream)is).materialize();
                nstreams.add(ns);
            } catch (IOException e) {
                throw new RemoteException("Failed to clone stream.", e);
            }
        }
    }

    return n;
}

```

- Call this function in “addTask” before the “synchronized” block:

```

task = cloneTask(task);

```

- The function “getTasks” has to be modified too. The returned streams will be closed by the framework after the client has downloaded them. Since the server object should be able to list the tasks many times, the stream objects have to be duplicated before they are returned. We use the “cloneTask” function for this purpose again and add a new function to duplicate a list of tasks:

```

private List<TaskInfo> cloneTaskList(List<TaskInfo> tasks) throws RemoteException {
    if (tasks == null) return null;
    ArrayList<TaskInfo> ntasks = new ArrayList<TaskInfo>(tasks.size());
    for (TaskInfo t : tasks) {
        ntasks.add(cloneTask(t));
    }
    return ntasks;
}

```

- Now, let function “getTasks” return a copy of tasks:

```

return cloneTaskList(tasksOfUser);

```

3.9 Client-side Interface, Invoke Client from Server

It could be useful in client-server applications to be able to initiate the data exchange by the server.

This is often called as server-push technology. BYPS supports a server-push functionality in a very convenient way, whereby the server can call interface methods implemented by the client. This methods can return values to the server and they even can throw exceptions which can be caught by the server.

The next example shows how to work with client-side interfaces. The code can be found at [TASKAPP] in branch “Notify”.

- In project “taskapp-api”, create a new interface “TaskNotify” in package “task.app” as follows, tag the interface with “@BClientRemote”. This advises the generator to create a skeleton class for all supported client-side platforms.

```
/**
 * @BClientRemote
 * @since 1.0.0.1
 */
public interface TaskNotify extends Remote {

    public int receiveTask(TaskInfo tasks) throws RemoteException;

}
```

- Run the generator: right-click “byps.xml”, “Run As – Ant Build”
- Refresh projects “taskapp-ser” and “taskapp-srv”: right-click project, “Refresh”
- In project “taskapp-client-j”, create a new class “TaskNotifyImpl” in package “task.app” inherited from “BSkeleton_TaskNotify”. Implement the “receiveTasks” function:

```
public class TaskNotifyImpl extends BSkeleton_TaskNotify {

    @Override
    public int receiveTask(TaskInfo task) throws RemoteException {
        System.out.println("received task: " + task);
        return 123;
    }

}
```

- In class “TaskappMain” add an instance of this class to the client object:

```
...
    BClient_Taskapp bclient = createClient(url);

    try {

        setCredentials("Fritz", "abc");
        bclient.addRemote(new TaskNotifyImpl());

        bclient.start();
    }
    ...
```

- In project “taskapp-srv”, class “TaskServiceImpl”, add a member variable “TaskappSession”:

```
private TaskappSession session;

public TaskServiceImpl( TaskappSession sess) {
    this.session = sess;
    this.userName = sess.getUserName();
}
```

- At the end of function “addTask”, add a call to the “receiveTask” function of the client.

Note, that the task object has to be copied again in order to return cloned streams. This is for the same reason for which “getTasks” returns cloned streams (see 3.8).

```
// Notify client
BClient_Taskapp bclient = (BClient_Taskapp)session.getClientR();
bclient.getTaskNotify().receiveTask(cloneTask(task));
```

- Save all sources, start the server and execute “TaskappMain add” from the Eclipse toolbar. The line “received task: ...” should be displayed in the console window.
- If you want to verify, that the streams are available too, add the following lines to your “receiveTask” implementation:

```
try {
    System.out.print("--- attachments: ");
    for (InputStream s : task.getAttachments()) {
        BContentStream bs = (BContentStream)s;
        System.out.print("{ " + bs.getContentType() +
            ", #" + bs.getContenLength() + ", [" );
        for (int i = 0; i < 10; i++) {
            System.out.print((char)bs.read());
        }
        System.out.print("]}, ");
    }
    System.out.println();
}
catch (IOException e) {
    e.printStackTrace();
}
```

3.10 JavaScript Client

This section creates a simple HTML page with JavaScript access to the “taskapp” server. It is a precondition for the example in section 3.11 where two client applications communicate with each other.

The code of this section can be found at [TASKAPP] in branch “JavaScript-Client”.

- Copy all files from [LIBDIR], sub-directory “js”, to “taskapp-srv/WebContent”.
- Open “taskapp-srv/WebContent/index.html”.

Define HTML Skeleton

- Insert the following style definitions somewhere in the HTML head element:

```
<head>
...
<style type="text/css">
table {border-spacing: 5px;}
td { background-color:#EEEEEE; }
</style>
...
</head>
```

- Replace the HTML body by the following block. The first table will display the task list, the second one will be used to add a task.

```
<body>
  <table>
    <tr>
      <td>Tasklist <input type="button" value="Update" onclick="updateTaskList()" /> </td>
    </tr>
    <tr>
```

```

        <td>
            <table id="taskList" >
                <tr>
                    <td>ID</td>
                    <td>User Name</td>
                    <td>Due Date</td>
                    <td>TODO</td>
                    <td>Properties</td>
                    <td>Attachments</td>
                </tr>
            </table>
        </td>
    </tr>
</table>
<br>
<table >
    <tr>
        <td>Add Task</td>
    </tr>
    <tr>
        <td>
            <table id="taskList" >
                <tr>
                    <td>ID</td>
                    <td>User Name</td>
                    <td>Due Date</td>
                    <td>TODO</td>
                </tr>
                <tr>
                    <td><input id="taskId" /> </td>
                    <td><input id="taskUserName" /></td>
                    <td><input id="taskDueDate" /></td>
                    <td><input id="taskTodo" /></td>
                </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td> <input type="button" value="Add"          onclick="addTask()" /> </td>
    </tr>
</table>
</body>

```

- Add script includes after the title definition and start a new script block. All subsequent functions will be defined in this block

```

<script type="text/javascript" src="json2.js"></script>
<script type="text/javascript" src="byps.js"></script>
<script type="text/javascript" src="taskapp.js"></script>

<script type="text/javascript">
    <!-- taskapp functions -->
</script>

```

Connect/Disconnect

- At first, define a variable for the URL to the “taskapp” servlet. Since the browser enforces the Same-Origin-Policy, this URL can be relative to the HTML page.

```

var url = "/taskapp-srv/taskapp"

```

- At next, the client object is initialized and stored in the global variable “bclient”. The initialization code is very similar to the Java code.

```

var bclient = null;
function init() {

    var wire = new byps.BWireClient(url, 0, 120);
    wire.getServletPathForNegotiationAndAuthentication = function() {
        return "/taskappauth/auth";
    }
}

```

```

};

var transportFactory = new byps.BTransportFactory(
    task.app.BApiDescriptor_Taskapp.instance(),
    wire, 1);

this.bclient = task.app.createClient_Taskapp(transportFactory);

this.bclient.start();
}

```

- The “init” function should be called, when the page is loaded:

```

window.onload = function() {
    init();
}

```

- The client object can be disconnected when the browser navigates to another location. This cancels the long-poll request that is always active in the background.

```

function done() {
    if (bclient) {
        bclient.done();
    }
}

window.onbeforeunload = function() {
    done();
};

```

3.10.1 Display the Task List

This section calls the “getTasks” function of the server and displays the received tasks in the upper table. Here, the server request is processed synchronously which makes it easier to follow the program flow. However, all server interface functions can be called asynchronously too. The section 3.10.2 shows how to achieve asynchronous calls.

- The following function “updateTaskList” reads the tasks from the server and adds a row to the HTML table for each task by calling “insertTaskRow”.

```

function updateTaskList() {

    // get tasks from server
    var tasks = bclient.taskService.getTasks();

    var tab = document.getElementById("taskList");

    // clear all data rows
    while (tab.rows.length > 1) tab.deleteRow(1);

    // insert rows
    for (var i = 0; i < tasks.length; i++) {
        insertTaskRow(tab, tasks[i]);
    }
}

function insertTaskRow(tab, task) {
    var colIdx = 0;
    var row = tab.insertRow(tab.rows.length);
    row.insertCell(colIdx++).innerHTML = task.id;
    row.insertCell(colIdx++).innerHTML = task.userName;
    row.insertCell(colIdx++).innerHTML = task.dueDate;
    row.insertCell(colIdx++).innerHTML = task.todo;
    row.insertCell(colIdx++).innerHTML = getTaskListProperties(task);
    row.insertCell(colIdx++).innerHTML = getTaskListAttachments(task);
}

```

- “insertTaskRow” uses two helper functions. The first one, “getTaskListProperties”, adds a

column with the task property values.

```
function getTaskListProperties(task) {
    var ret = "";
    if (task.properties) {
        for (var k in task.properties) {
            ret += k + "=" + task.properties[k] + "<br/>";
        }
    }
    return ret;
}
```

- The second helper function, “getTaskListAttachments”, renders a HTML anchor for each task attachment. The JavaScript definition of the “byps.BContentStream” class provides the member “url” from where the stream can be downloaded.
Please note that the stream objects can be downloaded only once – like in all other supported programming environments. This means, that the HTML anchor can be clicked only once. Furthermore clicking the anchor should not unload the current page since this would cancel the long-poll request that is always active in background. That's why “target=_blank” is assigned to the anchor.

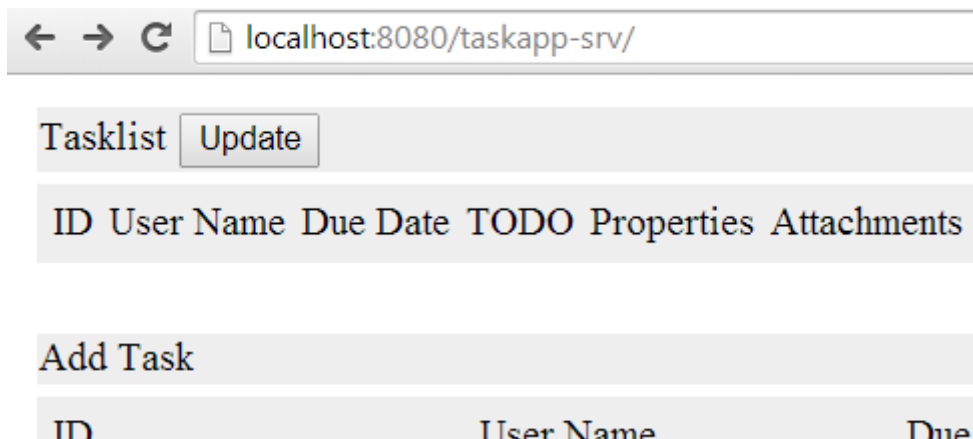
```
function getTaskListAttachments(task) {
    var ret = "";
    if (task.attachments) {
        for (var i = 0; i < task.attachments.length; i++) {
            var stream = task.attachments[i];
            ret += "<a target='_blank' " +
                "href='" + stream.url + "'>attachment" + (i+1) + "</a><br/>";
        }
    }
    return ret;
}
```

- For this example it is more convenient to let the browser download the attachments in order to display them in the browser window. Therefore, edit “TaskServiceImpl” and modify function “cloneTask”. Set property “attachment” for the copied “byps.BContentStream” objects:

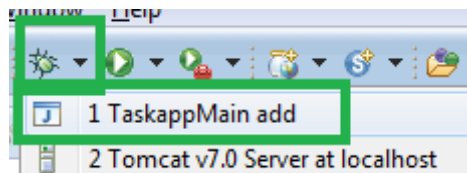
```
...
ns = ((BContentStream)is).cloneInputStream();
ns.setAttachment(true);
nstreams.add(ns);
...
```

Check Task List

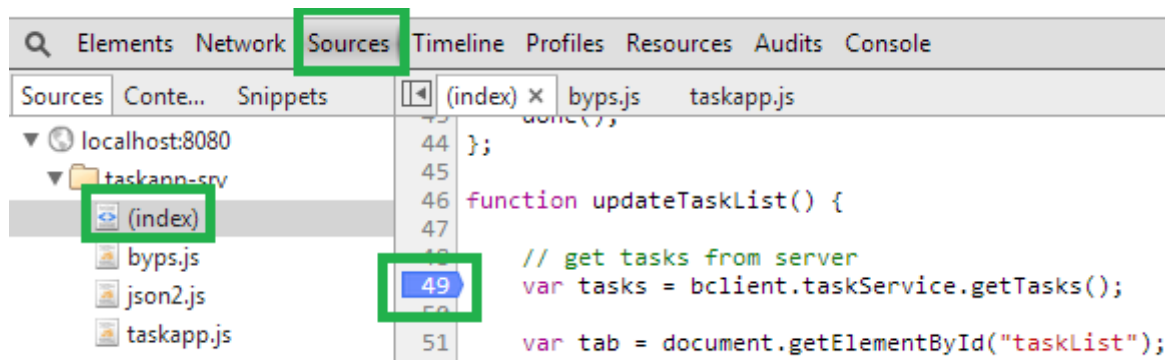
- In order to check the code written so far, start the server and navigate to the URL “<http://localhost:8080/taskapp-srv>”.
- The browser prompts for login credentials. Use “Fritz” and “abc” as defined in the file “Servers/Tomcat v7.0 .../tomcat-users.xml”. The result should look like this:



- Now, execute the “taskapp-client-j” to add a task object:



- Start the browser's debugger (e.g. Chrome: F12) and set a breakpoint in function “updateTaskList”.



- On the web page, click button “Update” and step through the code (e.g. in Chrome press F10 for a single step over).
- After the function is executed, the web page should display the task in the first table.

3.10.2 Add Task

This example adds a task object using an asynchronous request to the server.

- Before this can be done, the client notification code added in section 3.9 have to be commented out, since there is currently no client-side implementation (see 3.10.4). Edit “TaskServiceImpl”, function “addTask”:

```
// Notify client
// BClient_Taskapp bclient = (BClient_Taskapp) session.getClientR();
// bclient.getTaskNotify().receiveTask(cloneTask(task));
```

- Open “taskapp-srv/WebContent/index.html” and add the following lines. This function

assigns the input field values of the table to a “TaskInfo” object and sends the object by calling “addTask” to the server. Each interface function is performed asynchronously if a function object is added to the parameter list. The function object is called when the server responds. It receives the return value and an `byps.BException` object in case of an error.

```
function addTask() {
    var t = new task.app.TaskInfo();
    t.id = document.getElementById("taskId").value;
    t.userName = document.getElementById("taskUserName").value;
    t.dueDate = new Date(document.getElementById("taskDueDate").value);
    t.todo = document.getElementById("taskTodo").value;

    bclient.taskService.addTask(t, function(result, ex) {
        if (ex) alert(ex);
    });
}
```

Check Add Task

- Start the server, navigate to the URL “<http://localhost:8080/taskapp-srv>” and login with “Fritz”, “abc”.
- Enter some values in the “Add Task” table. Enter “Fritz” as user name. For “Due Date”, use a date format that can be parsed by the JavaScript Date object, e.g. an ISO conform format like “2014-02-01 11:12:13”.
- Click “Add” and then “Update” to view the created task.

3.10.3 Upload Files

This example shows how to upload files. The code of this section can be found at [TASKAPP] in branch “JavaScript-Upload”.

For uploading files on a web page, an HTML form has to be defined with an input field of type “file”. This form sends the file contents to the server where they are temporary buffered. The server answers with an array of stream IDs which are loaded into a hidden “iframe” on the web page.

An “onload” handler of the hidden “iframe” takes the stream IDs and assigns them to “byps.BContentStream” objects which in turn are assigned to a new “TaskInfo” object. Then, the handler calls the interface function “addTask” to add the new task object.

- Replace the code of the “Add Task” table by the following lines:

```
<form id="addTaskForm" method="post" enctype="multipart/form-data"
    action="/taskapp-srv/taskapp?uploadHandler=htmlform"
    target="uploadResultFrame">
    <table>
        <tr>
            <td>Add Task</td>
        </tr>
        <tr>
            <td>
                <table id="taskList">
                    <tr>
                        <td>ID</td>
                        <td>User Name</td>
                        <td>Due Date</td>
                        <td>TODO</td>
                        <td>Attachments</td>
                    </tr>
                    <tr>
                        <td><input id="taskId" /></td>
                        <td><input id="taskUserName" /></td>
                        <td><input id="taskDueDate" /></td>
                        <td><input id="taskTodo" /></td>
                        <td><input id="taskAtts" name="files[]"
                            type="file" multiple></td>
                    </tr>
                </table>
            </td>
        </tr>
    </table>
</form>
```



```

        </tr>
    </table>
</td>
</tr>
<tr>
    <td><input type="submit" value="Add" /></td>
</tr>
</table>
<iframe id="uploadResultFrame" name="uploadResultFrame"
    onload="onUploadResult()" style="visibility: hidden; display: none">
</iframe>
</form>

```

- The handler function “onUploadResult” is defined as follows:

```

function onUploadResult() {

    // Get DOM object of iframe body
    var iFrameBody;
    var iFrame = document.getElementById('uploadResultFrame');
    if (iFrame.contentDocument) { // FF
        iFrameBody = iFrame.contentDocument.getElementsByTagName('body')[0];
    } else if (iFrame.contentWindow) { // IE
        iFrameBody = iFrame.contentWindow.document.getElementsByTagName('body')[0];
    }

    // Stream IDs are in the iframe's body
    var ret = iFrameBody.innerHTML;
    if (ret) {

        // Obtain stream IDs in an array
        var streamIds = JSON.parse(ret);

        // Create BContentStream objects
        var attachments = [];
        for (var i = 0; i < streamIds.length; i++) {
            attachments[i] = new byps.BContentStream();
            attachments[i].streamId = streamIds[i];
        }

        // Add new task
        var t = new task.app.TaskInfo();
        t.id = document.getElementById("taskId").value;
        t.userName = document.getElementById("taskUserName").value;
        t.dueDate = new Date(document.getElementById("taskDueDate").value);
        t.todo = document.getElementById("taskTodo").value;
        t.attachments = attachments;

        bclient.taskService.addTask(t, function(result, ex) {
            if (ex) alert(ex);
        });
    }

    return ret;
}

```

3.10.4 Client-side Notification

JavaScript client applications can also implement client-side interfaces in order to receive notifications from the server.

The code of this example can be found at [TASKAPP] in branch “JavaScript-Notify”.

The notification code in the “TaskServiceImpl” has to be reactivated. This code informs the user of the current session of a task that she has just created. In practice it would be more useful to inform the user who is responsible for the task. In order to do this, the session of this user must be made accessible.

Obtain Access to all Sessions in TaskappSession

- In class “TaskappSession” add a static hash map for the sessions:

```
final static ConcurrentHashMap<String, TaskappSession> userSessions =
    new ConcurrentHashMap<String, TaskappSession>();
```

- Inside the class's constructor add the instance to the map:

```
public TaskappSession(HttpSession hsess, String remoteUser, File tempDir,
    BServerRegistry stubRegistry) {
    super(hsess, remoteUser, tempDir, stubRegistry);
    userSessions.put(remoteUser, this);
    ...
}
```

- Override the “done” method to remove the session from the map if it is disconnected:

```
@Override
public void done() {
    userSessions.remove(super.getRemoteUser());
    super.done();
}
```

Notify the Responsible User

- Edit class “TaskServiceImpl”, function “addTask”. Replace the comment lines at “// Notify client” with the following lines.
Since the client is a single-threaded web page, calling the “receiveTask” function synchronously might cause a deadlock. In order to invoke asynchronously, a “BAsyncResult” parameter was added to the function call.

```
// Notify client
TaskappSession taskSession = TaskappSession.userSessions.get(task.getUserName());
if (taskSession != null) {
    BClient_Taskapp bclient = (BClient_Taskapp) taskSession.getClientR();
    bclient.getTaskNotify().receiveTask(cloneTask(task),
        new BAsyncResultIgnored<Integer>());
}
```

Provide Client-Side Implementation

- Providing a client-side interface implementation in JavaScript looks very similar to Java. One most obvious difference is, that prototyping is to be used instead of class inheritance. Insert the following code somewhere inside the script block of the “index.html”:

```
// Implementation of TaskNotify
var TaskNotifyImpl = function() {

    this.receiveTask = function(task) {
        var tab = document.getElementById("taskList");
        insertTaskRow(tab, task);
    };

};

TaskNotifyImpl.prototype = new task.app.BSkeleton_TaskNotify();
```

- An instance of this “TaskNotifyImpl” function has to be added to the “bclient” object in the “init” function:

```
function init() {
    ...

    this.bclient = task.app.createClient_Taskapp(transportFactory);

    this.bclient.addRemote(new TaskNotifyImpl());

    this.bclient.start();
}
```

...

Try it out

- Start two independent browsers – e.g. browsers of different vendors.
- Open the web page under different accounts – e.g. login as “Fritz/abc” in the first browser window and “Maria/123” in the second one.
- In the first browser (used by “Fritz”), add a task for “Maria”. The second browser (used by “Maria”) should be notified and the newly created task should be listed.

3.11 Client-to-Client Communication

This example shows how clients can talk to each other over the server.

Imagine that the “taskapp” website requires a functionality to compute a checksum over all attachments of a task. For some reason, the checksum should be computed in an application that acts like a client to our server. We will use the “taskapp-client-j” project as a host for the checksum function.

The code of this example can be found at [TASKAPP] in branch “Client2Client”.

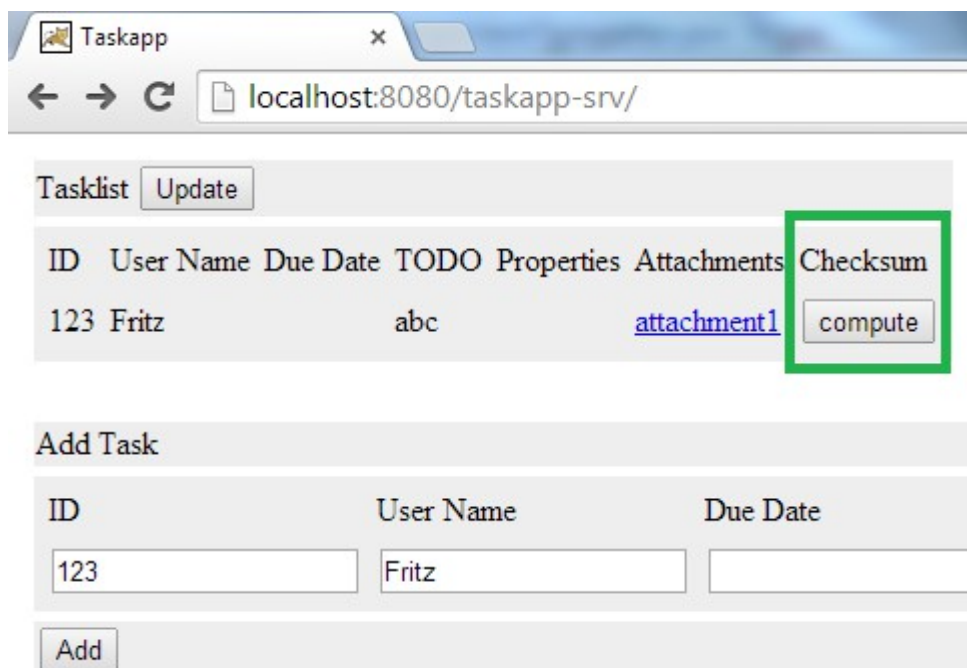


Abbildung 1: Screenshot of the Compute Checksum Example

Define an Interface for the Checksum Service

- Create a new Java class in project “taskapp-api” at “/src/task.app” named “CalculationService”.

```
/**
 * @since 1.0.0.1
 */
public interface CalculationService extends Remote {

    public int computeSimpleChecksum(List<InputStream> streams) throws RemoteException;

}
```

- Define Functions to Register the Service with the Server. Edit class „TaskappService“, add the following definitions:

```
/**
 * @since 1.0.0.1
 */
void registerCalculationService(CalculationService calc) throws RemoteException;

/**
 * @since 1.0.0.1
 */
CalculationService getCalculationService() throws RemoteException;

/**
 * @since 1.0.0.1
 */
List<InputStream> getTaskAttachments(int taskId) throws RemoteException;
```

- Run the BYPS generator: Right-click on “byps.xml”, “Run As – Ant Build”.
- Refresh project “taskapp-ser”

Provide an Interface Implementation

- Create a new Java class in project “taskapp-client-j” at “/src/task.app.client” named “CalculationServiceImpl”.

```
public class CalculationServiceImpl extends BSkeleton_CalculationService {

    @Override
    public int computeSimpleChecksum(List<InputStream> streams) throws RemoteException {
        if (streams == null) throw new IllegalArgumentException(
            "Parameter streams must not be null");

        int sum = 1;
        try {
            for (InputStream is : streams) {
                sum = computeChecksum(sum, is);
            }
        } catch (IOException e) {
            throw new BException(BExceptionC.IOERROR, "Failed to read attachment.", e);
        }
        return sum;
    }

    private int computeChecksum(int sum, InputStream is) throws IOException {
        try {
            int b = 0;
            while ((b = is.read()) != -1) {
                sum = 31 * sum + b;
            }
        } finally {
            is.close();
        }
        return sum;
    }
}
```

- In class „TaskappMain“, an instance of „CalculationServiceImpl“ has to be attached to the „BClient“ object. Furthermore the „taskapp-client-j“ should now wait for incoming computation request.

Create a new function in „TaskappMain“ as follows:

```
private static void calc(BClient_Taskapp bclient)
    throws RemoteException, InterruptedException {
    JFrame frame = new JFrame("Calculation Service");
    try {
```

```

        CalculationServiceImpl calc = new CalculationServiceImpl();
        bclient.addRemote(calc);
        bclient.getTaskService().registerCalculationService(calc);

        JOptionPane.showMessageDialog(frame, "Waiting for calculation requests.");
    }
    finally {
        bclient.getTaskService().registerCalculationService(null);
        frame.dispose();
    }
}

```

- Invoke this function in „main“ if the command line argument is „-calc“:

```

...
else if (command.equals("-list")) {
    listTasks(bclient);
}
else if (command.equals("-calc")) {
    calc(bclient);
}
...

```

Provide Checksum Service Registration

- Edit „TaskServiceImpl“ in project „taskapp-srv“
- Add static member to hold the computation service:

```
private static CalculationService calculationService;
```

- Add functions for registration:

```

@Override
public void registerCalculationService(CalculationService calc)
    throws RemoteException {
    calculationService = calc;
}

@Override
public CalculationService getCalculationService() throws RemoteException {
    return calculationService;
}

```

- Add a function for returning the attachments of a task:

```

@Override
public List<InputStream> getTaskAttachments(int taskId) throws RemoteException {

    List<TaskInfo> tasksOfUser = getTasks();

    for (TaskInfo t : tasksOfUser) {
        if (t.getId() == taskId) {
            return t.getAttachments();
        }
    }

    throw new BException(BExceptionC.INTERNAL, "Task not found");
}

```

Extend the web page

- Add a column „Checksum“ to the „taskList“ table:

```

<table id="taskList">
  <tr>
    <td>ID</td>
    <td>User Name</td>
    <td>Due Date</td>

```

```
 TODO | Properties | Attachments | Checksum |
```

- Place a button in the new column for each result row:

```

function insertTaskRow(tab, task) {
    var colIdx = 0;
    var row = tab.insertRow(tab.rows.length);
    row.insertCell(colIdx++).innerHTML = task.id;
    row.insertCell(colIdx++).innerHTML = task.userName;
    ...

    var btnId = "btn" + tab.rows.length;
    row.insertCell(colIdx++).innerHTML = '<input type="button" id="' + btnId + '" '
        + 'value="compute" onclick="computeChecksum(' + task.id + ')" />';
}

```

- The button calls the following function when clicked:

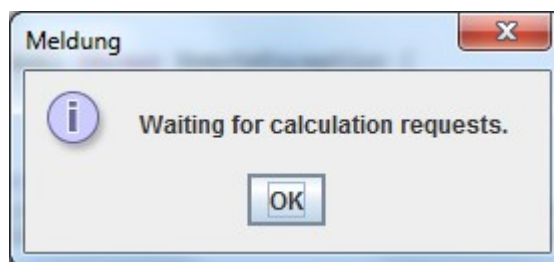
```

function computeChecksum(taskId) {
    var atts = bclient.taskService.getTaskAttachments(taskId);
    var calc = bclient.taskService.getCalculationService();
    if (calc) {
        calc.computeSimpleChecksum(atts, function(sum, ex) {
            alert("Checksum=" + sum + ", error=" + ex);
        });
    }
    else {
        alert("Calculation service not connected.");
    }
}

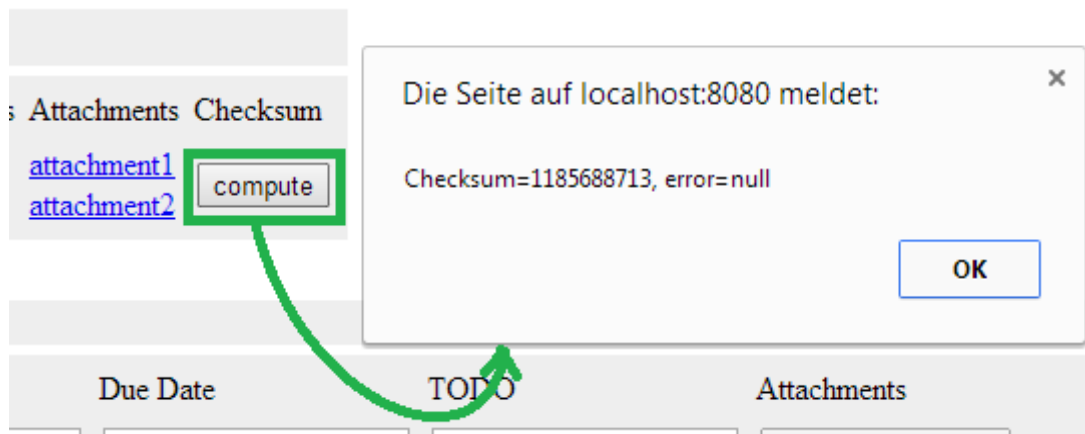
```

Try it out

- Start the server
- Add a new debug configuration for the „taskapp-client-j“ project with parameter „-calc“. This can be achieved as shown in section 3.5.1 .
- Start the „taskapp-client-j“



- Open the browser and navigate to <http://localhost:8080/taskapp-srv/>, login as „Fritz“ with password „abc“.
- Add a task with attachments for user „Fritz“
- Click button „Update“
- Click button „compute“, the checksum is shown in an alert box:



3.12 Develop C++ Client Applications

The required libraries and C++ includes can be found in the file “byps-lib.zip” at [BYPS]. In order to download this file at GitHub, click on “byps-lib.zip” and then “View Raw”. Unpack this ZIP archive into an arbitrary directory, e.g. c:\lib\byps. This directory is in the following sections referred to as [LIBDIR].

3.12.1 Java Development

A Java Software Development Kit version 7.0 [JDK] or newer and a Java IDE for Java EE is required. This documentation uses the Eclipse IDE [ECLIPSE].

The following libraries from [LIBDIR]/java are required to build and execute programs.

JAR file	Purpose
bypshttp-ov.jar	Run-time library for BYPS communication over HTTP.
bypsgen.jar	Generator for BYPS communication layers (required only for development)
commons-fileupload-1.2.2.jar	Support for HTML form-based file upload (required only on server-side)
commons-io-2.3.jar	Utilities for IO functionality (required only on server-side)
commons-logging-1.1.1.jar	Abstract logging framework
log4j-1.2.15.jar	Logging framework
tools.jar	Compiler for Java and Javadoc, part of the JDK (required only for development)

3.12.2 C# Development

At least Microsoft Visual Studio 2012 Express is required. The .NET run-time must be 4.0 or newer.

Projects have to refer to the DLLs in the Debug or Release sub-directory of [LIBDIR]/csharp. The BYPS communication classes have to be added in source code to the target projects.

3.12.3 C++ Development

Mircrosoft Visual Studio 2012 Express or newer is required. The generated code uses C++11 language features (e.g. lambda expressions).

Support for QT 5.0 is under development.

The BYPS communication classes have to be added in source code to the target projects.

3.12.4 JavaScript Development

The oldest supported Browser versions are currently unknown. It has been tested with Chrome Version 33, Firefox Version 26 and IE 11.

The following JavaScript libraries are required:

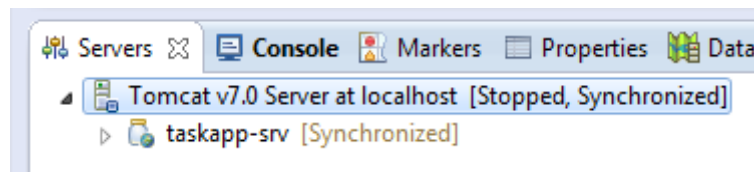
JavaScript file	Purpose
byps.js	BYPS communication layer
json2.js	JSON serialization, see [JSON]

4 Eclipse Tips

4.1 Client Cannot Connect to Server, HTTP 404

Sometimes, Eclipse fails to keep the code deployed on the Tomcat server up to date. Then it helps to remove the project “taskapp-srv” from the server and add it again. Perform the following steps in order to do this:

- Select the “Servers” view:



- Stop “Tomcat...” if it is running.
- Expand “Tomcat...”, right-click on “taskapp-srv”, select “Remove”.
- Right-click “Tomcat...”, “Publish”
- Right-click “Tomcat...”, “Add and Remove...”, select “taskapp-srv”, click “Add >”, click “Finish”
- Right-click “Tomcat...”, “Publish”
- Start “Tomcat...”

5 Bibliography

References

BYPS: Wolfgang Imig, Binary Portable Serialization, <https://github.com/wolfgangimig/byps>

JDK: ORACLE, JDK, <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

ECLIPSE: Eclipse Foundation, Eclipse IDE for Java EE Developers,
<https://www.eclipse.org/downloads/>

TOMCAT: Apache Software Foundation, Apache Tomcat, <http://tomcat.apache.org/download-70.cgi>

LIBDIR: Wolfgang Imig, Directory with BYPS libraries and C++ includes,
<https://github.com/wolfgangimig/byps/blob/master/byps-lib.zip>

JSON: Crockford, Java Script Object Notation, <http://www.json.org/>

TASKAPP: Wolfgang Imig, BYPS Taskapp Example, <https://github.com/wolfgangimig/taskapp>