

# **APIs for Integrating with FuegoBPM**

**Fuego, Inc.**

---

# APIs for Integrating with FuegoBPM

by Fuego, Inc.

Published January, 2005 - Version 5.5. Revision 10 - June, 2006.

Copyright © 2001-2006 Fuego, Inc.

## APIs for Integrating with FuegoBPM

Copyright 2001-2005 Fuego, Inc. All rights reserved.

This documentation is subject to change without notice. This documentation and the software described in this document contains proprietary trade secrets and confidential information of Fuego, Inc. and is also protected by U.S. and other copyright laws and applicable international treaties. Use of this documentation and the software is subject to the license agreement between you and Fuego, Inc. If no such license agreement exists, you may not use this documentation and software in any manner whatsoever. Unauthorized use of the documentation or software, or any portion of it, will result in civil liability and/or criminal penalties. U.S. Patent Pending.

Fuego, Fuego 4, Component Manager, Process Designer, Work Portal, Orchestration Engine, Execution Console, Process Analyzer, Organization Administrator are trademarks or registered trademarks of Fuego, Inc.

FuegoBPM 5, FuegoBPM Studio, FuegoBPM Designer, FuegoBPM Enterprise Administration Center, FuegoBPM Work Portal, FuegoBPM Portal Console, FuegoBPM Archive Viewer, FuegoBPM Logviewer, FuegoBPM Express Server, FuegoBPM Enterprise Server, FuegoBPM Application Server Edition, FuegoBPM Web Console, FuegoBPM Process Analyzer, FuegoBPM Data Store, FuegoBPM Dashboard, FuegoBPM BAM, FuegoBPM Portlets, FuegoBPM Suite, FuegoBPM Deployer, FuegoBPM Failover, FuegoBPM VCS, FuegoBPM Ant Tasks, FuegoBPM FDI, FuegoBPM Help Viewer, FuegoBPM Server are trademarks or registered trademarks of Fuego, Inc.

InstallAnywhere is a registered trademark of Zero G Software, Inc. Solaris and Java are trademarks of Sun Microsystems, Inc. Windows is a registered trademark of Microsoft Corporation.

All other trademarks, trade names, and service marks are owned by their respective companies.

---

---

---

---

## Table of Contents

1. Introduction .....	7
2. Working with PAPI .....	11
Introduction to PAPI .....	11
PAPI - How to .....	24
PAPI Properties .....	24
Set PAPI Properties .....	26
Connect to a session .....	27
Create filters .....	27
Get instances .....	32
Get an Instance's Events AuditTrail .....	32
Get all participants that worked with the instance .....	33
Get an Instance's Process .....	33
Get Roles that managed an instance based on Events .....	34
Get Activities that managed an instance based on Events .....	34
Get the instance's creation and termination time .....	35
Get an Instance's Audit Trail .....	35
Grab instances .....	35
Create an instance .....	36
Run a Global activity .....	36
Run an External Interactive activity .....	37
Run an activity .....	38
PAPI Examples .....	39
3. WAPI .....	41
Introduction to WAPI .....	41
HTML Process API .....	41
HTML Process API Examples .....	61
URLForAction Java Class .....	91
4. Extending the FuegoBPM Work Portal .....	92
Introduction to Extending the Portal .....	92
Why not just use PAPI instead .....	92
Understanding the Work Portal .....	93
A Web Application .....	93
Directory Structure .....	94

How it works .....	96
Customizing the Work Portal .....	96
The portal.properties file .....	97
CSS .....	99
Custom Images .....	102
Modifying the JSPs .....	105
WAPI .....	114
Main JSPs' layouts .....	116
instancesView.jsp .....	117
instanceDetail.jsp .....	117
search.jsp .....	118
JSP Descriptions .....	118
activityDoc.jsp .....	119
applicationsView.jsp .....	120
attachmentsView.jsp .....	121
auditTrail.jsp .....	121
changePassword.jsp .....	122
checkinAttachment.jsp .....	122
checkinWorkingAttachment.jsp .....	123
editAttachment.jsp .....	124
editOptions.jsp .....	125
executionLock.jsp .....	126
fileTypeAssociations.jsp .....	127
folder.jsp .....	127
instanceDetail.jsp .....	128
instancesView.jsp .....	129
login.jsp .....	131
logout.jsp .....	132
newAttachment.jsp .....	132
newNote.jsp .....	133
participantList.jsp .....	134
processDoc.jsp .....	135
search.jsp .....	136
send.jsp .....	139
sendTo.jsp .....	140
servererror.jsp .....	141
sessionExpired.jsp .....	142
viewAttachment.jsp .....	143

viewNote.jsp .....	144
viewWorkingAttachment.jsp .....	144
welcome.jsp .....	145

---

# Chapter 1. Introduction

## APIs for Integrating with FuegoBPM

FuegoBPM provides different APIs for external integration:

- **PAPI**
- **PAPI-WS**
- **WAPI**

## PAPI

Participants of FuegoBPM processes need an interface to interact with the Processes.

FuegoBPM provides a web-based work portal for this purpose, by which users can, among other things:

- See the list of instances pending on their inbox
- Search for instances by different criteria
- Save the searches with a name (views)
- View the details of an instance, including the audit trail
- Select an instance (assign it to oneself, effectively "locking" it)
- Execute tasks. That is, execute an Interactive activity of a Process on a particular instance

The FuegoBPM Work Portal uses the **Process-API** (**PAPI**, provided by Fuego) to communicate to the FuegoBPM Server.

This Java-based API can also be used by external applications to programmatically integrate with a FuegoBPM process (do searches, create new instances, send notifications, etc.) including some of the actions that a participant would normally do through the Work Portal.

So, a replacement for the FuegoBPM Work Portal could be built on top of PAPI, but all the features provided by the Work Portal are lost.

The most important piece of functionality that is lost when **not** using the Work Portal is the **execution framework** for interactive components. This means, the execution of client-side components in Interactive activities, like **FuegoBPM Screenflows, JSPs launched from a FuegoBPM process** or **FuegoBPM Presentations**.

Another point to keep in mind is that the authentication to the FuegoBPM Server is done explicitly in PAPI. When using the Work Portal, it normally shows a login screen to the participant, or the single-sign-on framework takes care of it.

## PAPI-WS

FuegoBPM exposes a subset of PAPI via a SOAP web service. The most common operations are available, and it may be a better alternative than using PAPI straight.

Some advantages are:

- Use of an Independent Programming language
- Simpler, easier to use
- No requirements on the client side (except for a standard SOAP library/runtime). When using PAPI, you need a few .jar files, plus connectivity configuration to access the Fuego Directory

Some disadvantages might be:



- Not all the API are exposed
- There is no easy way to serialize Java objects
- There is a performance hit

## WAPI (HTML-PAPI)

As explained above, the client-side execution framework of the Work Portal is not available when using PAPI(WS). In other words, the mechanism that renders FuegoBPM Presentations and other UI-related components is provided by the FuegoBPM Work Portal.

Fuego provides a way to leverage this functionality even when participants do not use the FuegoBPM Work Portal. This method is called WAPI, or the HTML Process API.

This framework provides a servlet that accepts HTTP requests to execute some actions (like running a global activity, executing a task). So, an external application may be presented to the participants with an HTML form with its 'action' URL pointing to this servlet. It expects some parameters to identify the operation requested and other

arguments.

If the requested operation runs an interactive activity with client-side components (p.e., a FuegoBPM Screenflow with a presentation) the participant will be interacting with this servlet until the execution of that task is finished. Then, the end user is automatically redirected to the original application's URL (this url is actually another parameter to the servlet).

This would be an example of such a web form:

```
<form method="get"
```

```
ACTION="http://host/portal/servlet/ExecutionDispatcher">
<input type="hidden" name="processId" value="/MyProcess">
<input type="hidden" name="activityId"
      value="/MyGlobalActivity">
<input type="hidden" name="actionId" value="RUN_GLOBAL">
<input type="hidden" name="fuego.portal.logoutURL"
value="http://host/originatingApp">
<input type="submit" name="Submit"
      value="Press to Submit">
</form>
```

It provides a simple way of reusing the client-side execution framework provided by the FuegoBPM Work Portal from another application.

By not using the FuegoBPM Work Portal, the functionality provided by it is clearly lost. For example: login/out screens, inbox presentation (with selectable and sortable columns), the UI for performing searches, being able to save them, creation of bookmarks, among others.

But, it's perfectly feasible to build a custom solution for replacing the Work Portal.

PAPI and PAPI-WS provide a programmatic way of communicating to the FuegoBPM Server and performing all kind of operations (searching, aborting, executing, creating instances, etc)

If there is a need for leveraging the UI components provided by FuegoBPM (screenflows, presentations, etc) but outside of the Engine, then the HTML Process API (WAPI) is the answer.

By combining both PAPI(WS) and WAPI it is possible to build a solid solution. It will obviously require additional work, since some functionality normally provided by the FuegoBPM Work Portal needs to be implemented, but it is a feasible alternative.

---

# Chapter 2. Working with PAPI

## Introduction to PAPI

### What is PAPI ?

The Fuego Process Application Programming Interface (PAPI) is the API provided and implemented by Fuego to interact with deployed processes. The processes accessed can be on one or more FuegoBPM Execution Engines. PAPI is a Client-Server API since the API delegates the execution of the requested operations to the correct engine.

PAPI classes let you handle operations over deployed processes in a group of FuegoBPM Execution Engines. This API interacts with the FuegoBPM Execution Server to programmatically do things such as:

- Create a new instance in a process
- List instances the user is authorized for
- Abort instance
- Attach a file to an instance
- Audit an instance
- Edit instance's attachment
- List deployed process activities
- Get latest version of process deployed
- List instance attachments
- Notify instance
- Run an instance's task

- Select an instance
- Suspend an instance

PAPI is a 100 % Pure Java API and can be invoked by any JVM supporting JVM version 1.4.2 or above.

## When should we use PAPI ?

PAPI is primarily used by Java Applications that need interaction with processes deployed in one or more FuegoBPM Enterprise Execution Servers. This is not a functionality to use with the FuegoBPM Studio.

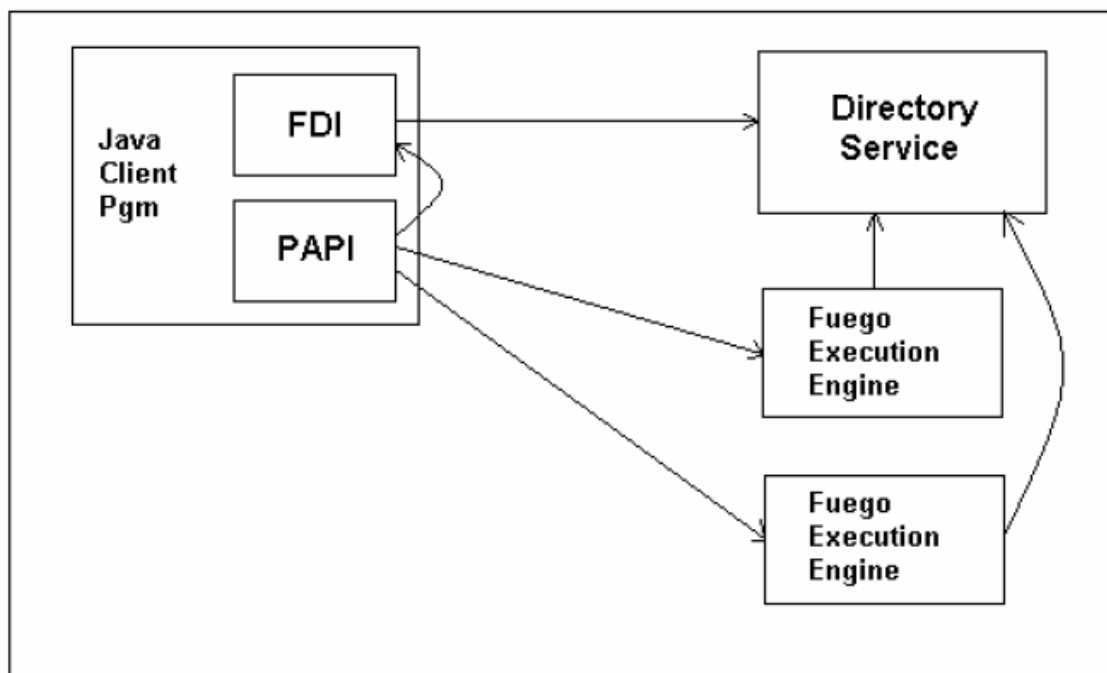
## What other Fuego applications or APIs use PAPI ?

PAPI is Fuego's lowest level API. As such, it is used by higher level APIs that provide greater abstraction and less complexity. Some of these Fuego Applications and APIs are:

- *FuegoBPM Work Portal Web Application*: FuegoBPM Work Portal uses PAPI extensively to interface with a number of Fuego Execution Engines simultaneously. The Work Portal provides a user interface to work with business process interactive activities easily.
- *Fuego WAPI*: WAPI is a Web API implemented with Java Servlets that provides connectivity with an engine through the HTTP transport protocol. The Servlets composing WAPI in turn use PAPI to actually execute the actions to process deployed in a Fuego Execution Engine. WAPI is an extension to Fuego's Work Portal Web Application and as such it is bundled with it. This API is very useful when Fuego needs integration from HTML Forms or JSPs.
- *Fuego Web Services Web Application*: This is a Web Application provided

by Fuego that allows Web Service Client Applications to connect using HTTP to Fuego processes and execute actions on them. This Web Application in turn uses PAPI to connect to the Fuego Execution Engine.

## PAPI Architecture Overview



A Java Client Program can use FuegoBPM Process API (PAPI) to connect to a Directory Service. As multiple FuegoPM Execution Servers can share the same Directory Service, FuegoBPM PAPI can be used to transparently connect to each of them.

Within the Directory Service, FuegoBPM stores information related to the organization structure where the processes are deployed as well as all the metadata about the processes that are published and deployed.

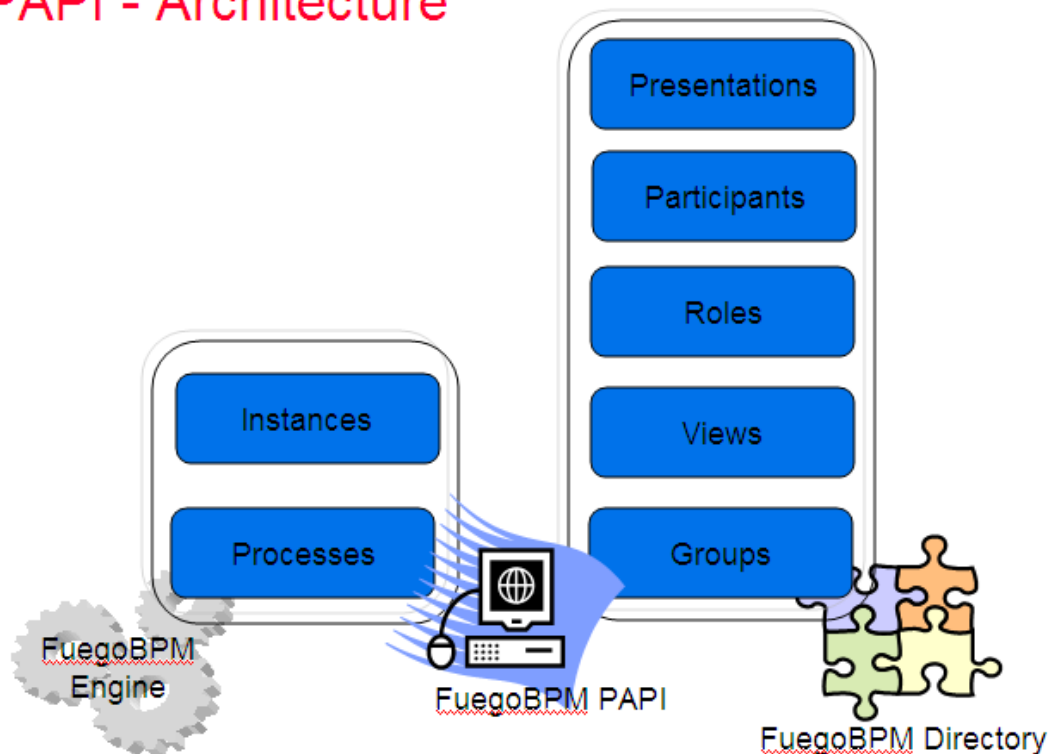
It is important that initially, PAPI connects to the Directory Service because the client Java Program needs to first authenticate the

participant in order to obtain a session with a FuegoBPM Execution Server. Once the authentication has been granted, the client Java Program using PAPI will be able to start using PAPI calls to any FuegoBPM Execution Server deployed to the connected Directory Service.

The following picture depicts from where PAPI retrieves the information.

- Instances and Processes are retrieved from the Engine, and
- Participants, Roles, Groups, Presentations and views are retrieved from the FuegoBPM Directory Service:

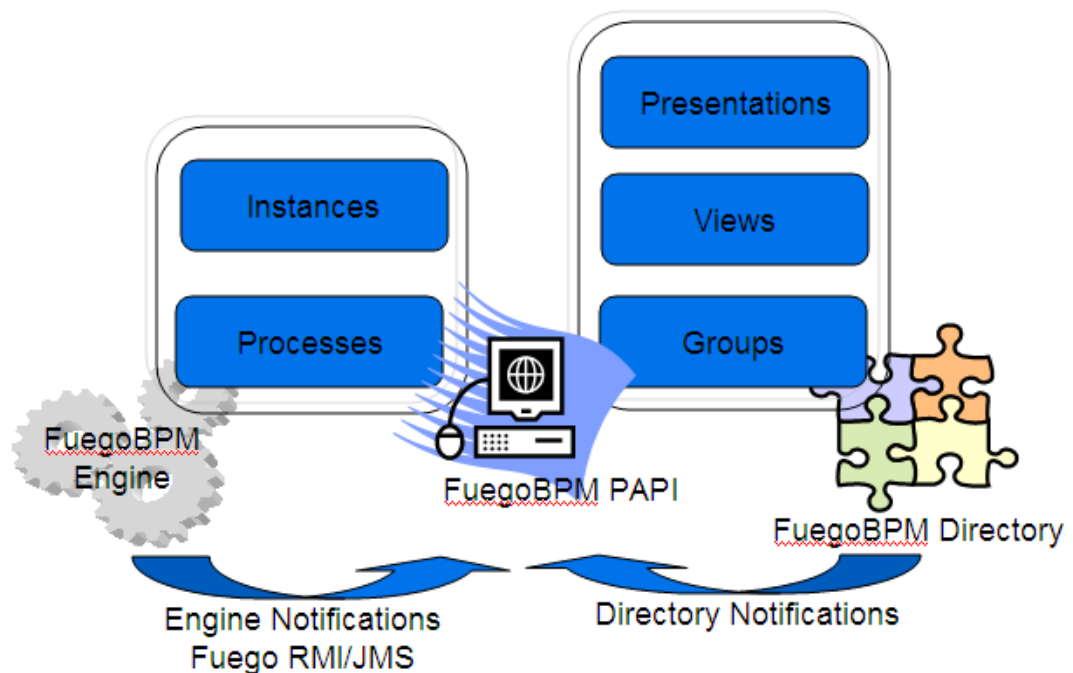
## PAPI - Architecture



## PAPI Notifications

As PAPI recovers information from two sources, notifications to PAPI are provided in two different ways, as depicted below:

## PAPI – Notifications



- **Engine Notifications:** the interval of time between notifications are defined in **FuegoBPM Console**. Within the Engine properties, **Others** tab, in the *Latency between notifications* property. This interval applies only when using a standalone Engine. In a J2EE environment, notifications are subject to the queue management.
- **FDI Notifications:** the interval of time between notifications are defined using the PAPI method **ProcessService.startUpdater**. See *PAPI javadoc* for more information.
- **Participant and Roles** refresh occur at login time.

## API Primer

### Requirements

In order to successfully compile and execute a PAPI program, it is important to comply with the following minimal requirements and instructions provided below.

### JVM

Use a JVM 1.4.2 or greater, to successfully compile and execute a Java PAPI program.

### Compilation Jars

When compiling a Java PAPI program you will need to include the following Java Jar files into the java compiler classpath. These Jar files are:

- ftpapi.jar - (in the {Fuego 5.5 home directory}/lib directory)
- ftlib.jar - (in the {Fuego 5.5 home directory}/lib directory)
- fuegocore.jar – (in the {Fuego 5.5 home directory}/lib directory)

### Compiling a Java PAPI Program

The Syntax for compiling a Java PAPI program is:

```
C:\papi>{JVM 1.4.2 path}\bin\javac -classpath
"{Fuego 5.5 home directory}\lib\ftpapi.jar;
{Fuego 5.5 home directory}\lib\fuegocore.jar;
{Fuego 5.5 home directory}\lib\ftlib.jar"
{Java Source filename}
```

Where



- {JVM path} - The JVM path. For example: e:\j2sdk1.4.2.
- {Fuego 5.5 home directory} - The Fuego 5.5 path. For example: e:\fuego5.5\enterprise.
- {Java Source filename} - You are running the command from the directory where the class is located.

The command line to compile the PAPI program in this example is:

```
C:\papi>e:\j2sdk1.4.2\bin\javac -classpath
"e:\fuego5.5\enterprise\lib\ftpapi.jar;
e:\fuego5.5\enterprise\lib\fuegocore.jar;
e:\fuego5.5\enterprise\lib\ftlib.jar" MyProgram.java
```

After successfully compiling the Java PAPI program, the file called MyProgram.class is created in the c:\papi directory.

## Runtime Jar Files

When running a Java PAPI program, you will need to include the following Java Jar files into the java classpath. These Jar files are:

- fuegopapi-client.jar
- {JDBC Driver} (this is the JDBC Driver for the Directory Service Provider)
- jms.jar (for J2EE environment)
- ejb-2.0.jar (for J2EE environment)

## Running the Java PAPI Program

In order to avoid unnecessary missing plugin warning messages, it is

recommended the jars needed to execute a Java Client Program using PAPI are copied to a new folder or directory accessible by the Java Client Program. We will assume this directory is c:\papi\lib for simplicity.

Based on the assumption above, the command line to run the Java Client Program using PAPI should be:

```
C:\papi>{JVM path}\bin\java -classpath ".;  
{Papi lib directory}\fuegopapi-client.jar;  
{Papi lib directory}\{JDBC Driver}"  
{Java Class filename} 'propertiesFile'
```

Here is how this Java program is invoked and what the arguments mean:

- {JVM path} - The JVM path is e:\j2sdk1.4.2.
- {Papi lib directory} - The directory where the PAPI runtime jar files were copied over from the Fuego Enterprise Installation. In our example: c:\papi\lib.
- {JDBC Driver} - JDBC Driver for the Directory Service Provider (e.g. Oracle, Microsoft SQL Server, or DB2). If you are connecting to a Directory Service that is accessed via JNDI (e.g.: iPlanet, Microsoft Active Directory), there is no need to include a JDBC driver in the classpath.
- {Java Class filename} - You are running the command from the directory

where the class you just created is located.

- *propertiesFile*: this is a file that is passed as the first argument of your PAPI program. In this file you can define:

- Directory Service
- User
- Password, etc.

For example you can have a properties file as the following:

```
fuego.directory.file=/development/directory.properties
fuego.directory.id=default
fuego.papi.cache.size=5000
test.user.id=user
test.user.password=password
test.process.id=/Process#Default-1.0
test.activity.name=Interactive
test.argument.name=argument
test.argument.value=argumentValue
test.instance.correlation=correlationName
test.instance.key=correlationValueSeparatedByComma
```

And this properties file is passed as the first argument of the java program.

From the java program, the different properties values are recovered and used to be able to connect:

```
import java.util.*;
import java.io.*;
import fuego.papi.*;
import fuego.papi.exception.*;

public class PapiTest
{
    public static void main(String[] args)
    {
        try {
            String propertiesFile = args[0];

            // Load test properties file (include papi properties)
```

```
Properties testProperties = new Properties();
testProperties.load(new FileInputStream(propertiesFile));

String processId = testProperties.getProperty
    ("test.process.id");
String userId = testProperties.getProperty
    ("test.user.id");
String userPass = testProperties.getProperty
    ("test.user.password");
String activityName = testProperties.getProperty
    ("test.activity.name");
String argumentName = testProperties.getProperty
    ("test.argument.name");
String argumentValue = testProperties.getProperty
    ("test.argument.value");
String correlation = testProperties.getProperty
    ("test.instance.correlation");
String instanceKey = testProperties.getProperty
    ("test.instance.key");

// Create a new fuego.papi.ProcessService
ProcessService service =
    ProcessService.create(testProperties);

// Create a fuego.papi.ProcessServiceSession
ProcessServiceSession session =
    service.createSession(userId, userPass, "localhost");
```

## Directory Service Configuration file (directory.properties)

To successfully interface with a process, you must first connect to the Directory Service. The connection and binding parameters are specified as arguments to the Java PAPI Program.

For example, if the Directory Service is an oracle then, the Fuego Directory Service URL is  
oracle://myserver:1521/schema=oracledi45,sid=MYSERVER.

There are other binding parameters stored in a configuration file called "directory.properties" located in the {Papi lib directory}\..\conf directory. This file should contain the following

properties:

```
provider.{Directory Service Provider}.anonymous-user=theuser
provider.{Directory Service Provider}.anonymous-password=
password
```

where {Directory Service Provider} can be:

- Oracle: oracle
- MS SQL Server: mssqlserver
- IBM DB2: db2 or db2type2
- Microsoft Active Directory: mad
- iPlanet: iplanet

This configuration file is loaded relative to the location of the ftlib.jar Jar file. For example, this file (in the classpath) is referenced from c:\papi\lib, the configuration file is then loaded from c:\papi\lib\..\conf\directory.properties which makes the directory c:\papi\conf.

## PAPI Class Diagrams

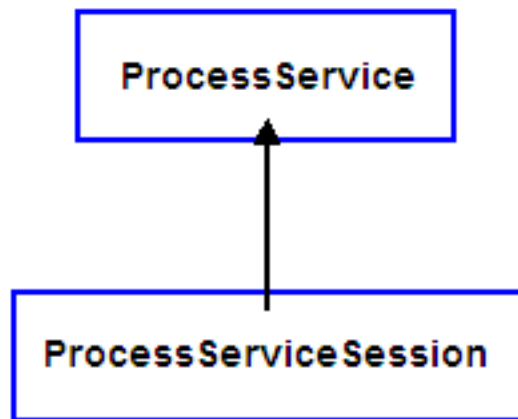
The classes constituting PAPI are divided in 3 groups:

- Operation Classes: These classes trigger actions to be performed by a Fuego Execution Engine. An example of this would be ProcessService.
- Data Classes: These classes are usually the result of invoking an action over an Operation class or metadata classes. An example of this would be Process, Activity and Task.

- Exception Classes: These classes are used to map exceptions while executing the Operation classes.

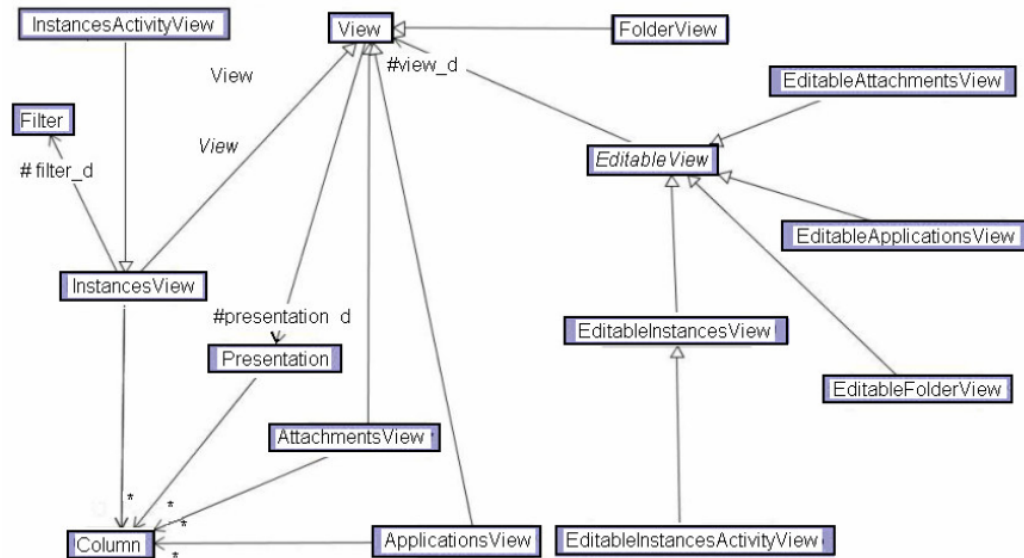
## Operation Classes

Shown below is the Operation Class diagram.



The first thing to do when using PAPI is to create a **ProcessService** object. This object is needed in order to create a **ProcessServiceSession** object. Once the session has been successfully obtained, you will be able to use its methods to perform all the operations needed to interact with deployed processes, views and filters.

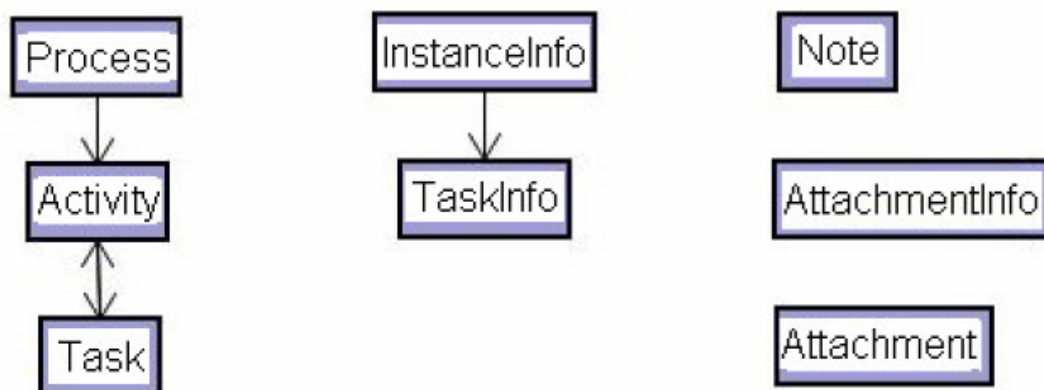
The diagram below depicts the relationship of the View and Filter classes.



**Views** and **Filters** are the other two abstractions within PAPI you can apply operations. For example, the instances in the Work Portal are retrieved using Filters. These Filters in turn contain attributes to be retrieved from a process.

## Data Classes

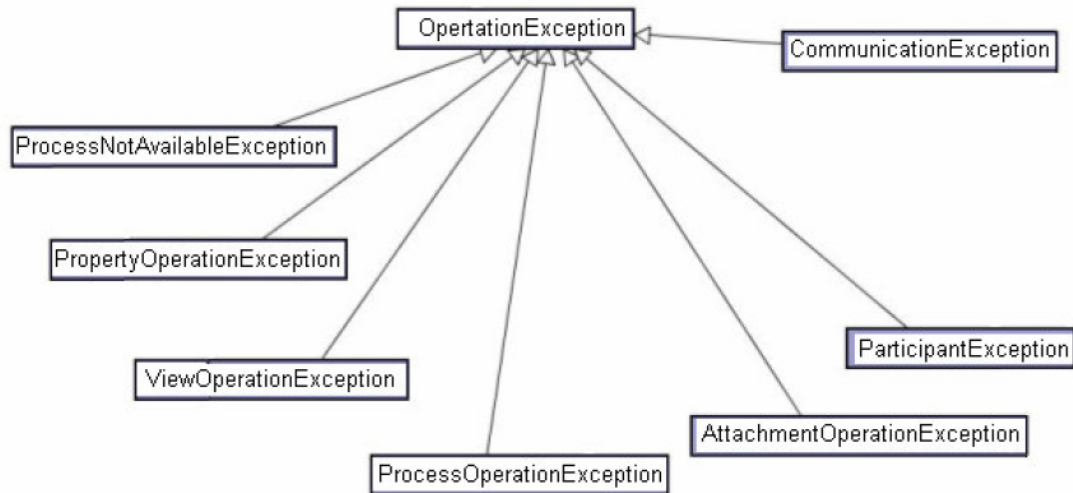
Shown below is the Data Class diagram.



These classes represent the relationship of Data classes that are normally returned when invoking a method in an Operation class.

## Exception Classes

Shown below is the Exception Class diagram.



Depending on the operation performed, PAPI can return any of the exceptions inherited from the **OperationException** class.

## PAPI - How to

### PAPI Properties

PAPI Properties are used when creating a connection through the *ProcessService*. Available properties are:

- `ProcessService.DIRECTORY_ID`
- `ProcessService.DIRECTORY_PROPERTIES_FILE`
- `ProcessService.DIRECTORY_PROPERTIES_RESOURCE`
- `ProcessService.DIRECTORY_PROPERTIES_URL`
- `ProcessService.INSTANCE_CACHE_SIZE`



- ProcessService.WORKING\_FOLDER
- ProcessService.DOCUMENTATION\_FOLDER

Also *custom* properties can be set to specify any special condition you want to manage through your PAPI program.

Check the properties defaults in the PAPI javadoc.

## ProcessService.DIRECTORY\_ID

This property is used at connection creation time to indicate the name of the FuegoBPM directory to which the PAPI program will connect.

## ProcessService.DIRECTORY\_PROPERTIES

These properties are three. The user can only set one of them. They overlap one to each other in a certain order if the user has not defined one.

These properties are used to indicate where is located the *directory.properties* configuration file. Depending where the file is located you must set a different property.

Property ProcessService.	Description
DIRECTORY_PROPERTIES_FILE	Use this property to specify where is the directory.properties file in the file system. You must indicate the full path.. For example: /develop/config/directory.properties, \$FUEGO/conf/directory.properties
DIRECTORY_PROPERTIES_RESOURCE	Use this property to define which is the name of the properties file. This file has to be in the resources, in our case within the CLASSPATH. It can appear alone or within a jar file.

Property <b>ProcessService.</b>	Description
DIRECTORY_PROPERTIES_URL	Use this property to define the URL where the directory.properties file is located.

## **ProcessService.INSTANCE\_CACHE\_SIZE**

This property is used to set the number of alive instances per process that will be maintained in the cache.

If in a certain moment, the quantity of alive instances is greater than the number set in this property then, the cache becomes an "opened cache". When the cache is *opened* the instances will be get from there on from the FuegoBPM Server. Once that the cache becomes "opened" it remains opened.

## **ProcessService.WORKING\_FOLDER**

This property is used to indicate where the catalog jars will be loaded. These catalogs are the ones corresponding to the projects deployed in the FuegoBPM Server. These catalog jars, are all loaded at initialization time with the creation of the ProcessService.

## **ProcessService.DOCUMENTATION\_FOLDER**

This property indicates the directory where the process documentation that the user requires will be loaded.

This documentation is in HTML format. It is loaded on demand when the user makes the requirement, for example:

**ProcessServiceSession.getProcessDocumentation(Locale),**  
**ProcessServiceSession.getActivityDocumentation(Locale).**

## **Set PAPI Properties**

```
java.util.Properties properties =  
    new java.util.Properties();
```

```
properties.setProperty(ProcessService.DIRECTORY_ID,
                        "default");
properties.setProperty(
    sService.DIRECTORY_PROPERTIES_RESOURCE,
    "directory.properties");
properties.setProperty(ProcessService.INSTANCE_CACHE_SIZE,
                        "2000");
properties.setProperty(ProcessService.WORKING_FOLDER,
                        "/tmp");
properties.setProperty("customProperty", "customValue");

fuego.papi.ProcessService processService =
    fuego.papi.ProcessService.create(properties);
```

## Connect to a session

```
Properties properties = new Properties();

//Set the directory.properties file if not using the default
properties.setProperty(
    ProcessService.DIRECTORY_PROPERTIES_FILE,
    "/fuego5.5/enterprise/conf/directory.properties");

// Create a Process Service
ProcessService processService =
    ProcessService.create(properties);

// Create the session for the participant
fuego.lib.ConnectionPassport passport =
    processService.createPassport("test");
passport.setPassword("passwod");

ProcessServiceSession session =
    processService.createSession(passport, "localhost");
```

## Create filters

```
// Create a filter to search for the instance.
Filter filter = ProcessService.createFilter();
```

## By Participant and/or Status

The search scope on participant, instance's status and activity is done by using the SearchScope object.

If no search scope is define on this elements, the filter uses the defaults that is ParticipantScope.ALL, StatusScope.ALL.

```
// In this example the instance is in any Role
// (selected or not) and in any Status
filter.setSearchScope(new SearchScope(ParticipantScope.ALL,
                                       StatusScope.ALL, null));
```

The Scope can be defined by:

1. **ParticipantScope:** Defines in which Roles the instance should be, selected or not. Possible values are:
  - a. ALL: All participants.
  - b. ALL\_IN\_ROLE: All the instances in the participant Roles
  - c. PARTICIPANT\_ROLES: Instances selected by me and instances not selected by any participant in the participant roles
  - d. PARTICIPANT: Only the ones selected by the participant
2. **StatusScope:** Defines the Status in which the instance should be when the filter is executed. Possible values are:
  - a. ONLY\_INPROCESS: Only instances that are In Process
  - b. ONLY\_COMPLETED: Only instances that are Completed
  - c. ONLY\_ABORTED: Only instances that are Aborted -

- d. **INPROCESS\_AND\_COMPLETED:** Instances that are Completed or In Process -
  - e. **INPROCESS\_AND\_ABORTED:** Instances that are In Process or Aborted
  - f. **ABORTED\_AND\_COMPLETED:** Instances that are Aborted or Completed
  - g. **ALL:** Instances in any status
3. **ActivityScope:** Defines the Activity in which the instance should be when the filter is executed.

**For example:**

- To retrieve all the instances that are being executed within a process

```
new SearchScope(ParticipantScope.ALL,  
                StatusScope.ONLY_INPROCESS, null).
```

- To retrieve all the instances that the participant is enabled to work with

```
new SearchScope(ParticipantScope.PARTICIPANT_ROLES,  
                StatusScope.ONLY_INPROCESS, null);
```

- To retrieve all the instances in a specific Activity

```
new SearchScope(ParticipantScope.ALL, StatusScope.ALL,  
                "activityName").
```

Not all combinations of Participant and Status scopes are valid. Combining participant with a StatusScope that does not include the instance status IN PROCESS builds an excluyent condition. Aborted and Completed instances have not Role.

## By Predefined Variables

Predefined Variables can be added by using the VarDefinition class or by getting them from the session.

The predefined variable id has to be indicated when it is obtained using the VarDefinition. The possible variable ids for the predefined variables are defined in the VarDefinition class.

```
// Getting the time in which the instances was RECEIVED  
// using the ID of the predefined variable: RECIEVED_ID.  
// This attribute is used to set the condition like:RECEIVED  
// predefined varaible equal to 'now'.  
  
filter.addAttribute(VarDefinition.getDefaultVarDefinition(  
                    VarDefinition.RECEIVED_ID),  
                    Comparison.IS, new Time());
```

If you prefer to get the variable from the session do:

```
// Retrieve the predefined variable for Activity  
VarDefinition variable =  
    session.getVar(VarDefinition.ACTIVITY_ID);
```

## By External Variable

External Variables have to be obtained first from the session to be added later as an attribute to the filter condition.

```
// Create a filter to retrieve all instances that have a
// defined external variable set to a specific value.
// Retrieve the external variable called "external"

VarDefinition variable = session.getVar("external");

// Define the VALUE that the instance's external variable
// must have.

filter.addAttribute(variable, Comparison.IS, "Hello");
```

## By setting Parametric conditions

```
// Add a variable
// (External or predefined, see sections above)
// as parametric. The last parameter set as true
// indicates that the variable added to the filter condition
// is parametric.

attributeVariable = filter.addAttribute(variable,
    Comparison.GREATER_THAN, new Integer(5), true);

// Getting all the filter attributes defined as parametric.
// In our example the list contains only one attribute as
// we defined as parametric only the external variable
// "MyVariable".

List parametricAttributes = filter.getParametricAttributes();

// Searching the parametric attribute to modify.
int size = parametricAttributes.size();

for (int i = 0; i < size; i++) {
    FilterAttribute attribute =
        (FilterAttribute) parametricAttributes.get(i);

    if (attribute.getVariableId().equalsIgnoreCase(
        "MyVariable")){
        attribute.setValue(new Integer(22));
    }
}
```

```
}  
}
```

## Get instances

Previous Steps

- Create a FILTER

```
//Get instances for a given session and filter  
  
// Execute the filter over the required process  
// (using the consolidated process ID).  
InstanceInfo[] instances = new InstanceInfo[0];  
  
try {  
    instances = session.getInstancesByFilter(  
        new String[] { "/MyProcess" }, filter);  
}  
catch (BatchOperationException e) {  
    e.printStackTrace();  
}  
  
if (instances != null) {  
    return instances[0];  
}  
else {  
    throw new RuntimeException(  
        "Instance not found.\n" + filter);  
}
```

## Get an Instance's Events AuditTrail

Previous Steps

- Get a SESSION
- Get an INSTANCE



```
// Get all the instance's events

InstanceEvent[] events = new fuego.papi.InstanceEvent[0];
events = session.getInstanceEvents(instance.getId());
```

## Get all participants that worked with the instance

Previous Steps

- Get an Instance's EVENTS

```
// Get the Participants that interacted with the instance
// (based on the events)
Set participants = new TreeSet();

for (int i = 0; i < events.length; i++) {
    participants.add(events[i].participantId);
}

String[] result = (String[]) participants.toArray(
    new String[participants.size()]);
printArray("Participants", result);
```

## Get an Instance's Process

Previous Steps

- Get a SESSION
- Get an INSTANCE

```
// Get instance's process
Process process=session.getProcess(instance.getProcessId());
```

## Get Roles that managed an instance based on Events

Previous Steps

- Get PROCESS
- Get Instance's EVENTS

```
// Get the Roles in which the instance was (based on
// the events)

Set roles = new TreeSet(String.CASE_INSENSITIVE_ORDER);

for (int i = 0; i < events.length; i++) {
    String    activityName = events[i].activity;
    Activity activity = process.getActivity(activityName);
    roles.add(activity.getRole());
}

String[] result =
    (String[]) roles.toArray(new String[roles.size()]);
```

## Get Activities that managed an instance based on Events

Previous Steps

- Get PROCESS
- Get Instance's EVENTS

```
// Get the Activities in which the instance was (based on
// the events)
Map activities = new TreeMap();

for (int i = 0; i < events.length; i++) {
    String activityName = events[i].activity;
    activities.put(activityName,
        process.getActivity(activityName));
}
```

```
Activity[] result =  
    (Activity[]) activities.values().toArray(  
        new Activity[activities.size()]);
```

## Get the instance's creation and termination time

### Previous Steps

- Get an INSTANCE

```
// Get the instance's Creation and Termination Time.  
Time creationTime = instance.getCreationTime();  
Time terminationTime = null;  
  
if (instance.isCompleted() || instance.isAborted()) {  
    terminationTime = instance.getReceptionTime();  
}  
  
System.out.println("Creation time : " + creationTime);  
System.out.println("Termination time : " + terminationTime);
```

## Get an Instance's Audit Trail

```
// Get the instance's Audit Trail and print it out.  
AuditTrail auditTrail = new AuditTrail(session,  
                                       instance.getId());  
auditTrail.load();  
auditTrail.print(System.out);
```

## Grab instances

### Previous Steps

- Get an INSTANCE

```
// Grab an Instance
session.instancesGrab(new InstanceStamp[ ]
                        { instances[0].getStamp() },
                        "MyGrabActivity");
```

## Create an instance

### Previous Steps

- Get a SESSION

```
// Create the arguments to be passed to the
// Begin activity according to the Argument Mapping
// to be used.
// In the example there is an Argument Mapping called
// "ExternalIn" that expects the argument called
// "xmlOrder". "myxmlOrder" contains the information
// to be passed through the argument.

        Arguments arguments = Arguments.create();
        arguments.putArgument("xmlOrder", myxmlOrder);

        InstanceInfo instance = null;

// Create an Instance in the process "Orders" using
// the argument mapping called "ExternalIn" and pass the
// arguments set previously.
        instance = session.createProcessInstance
                        ("/Orders", "ExternalIn", arguments);
```

## Run a Global activity

### Previous Steps

- Get a SESSION

```
// Run a Global activity.
// In the example, run the global activity "GetProducts"
// within the "Orders" process. The global activity's output
```

```
// arguments are received in "arguments"
Arguments arguments = Arguments.create();

session.runGlobalActivity("/Orders/GetProducts", arguments);

// The global activity returns an output argument called
// "globalArg". The local variable "info" is set with the
// information contained in it

String info = (String) arguments.getArgument("globalArg");
```

## Run an External Interactive activity

### Previous Steps

- Get a SESSION
- Get an INSTANCE

To Run an Interactive activity implemented as an external task, the task has 2 FBLs, the **prepare** one and the **commit** one.

They are run when either of the following methods are invoked.

### Prepare method

```
// In the following the instance's activity is an the
// interactive activity implemented as an external task.
// The "prepare" FBL returns an argument called
// "preArgOut". No arguments are passed to the FBL.

Arguments arguments = Arguments.create();

// Executes the "prepare" method of the interactive
// activity (implemented as External)
arguments = session.prepareExternalActivity
    (instance.getId(),
     instance.getActivityName(),
     Arguments.create());
```

```
// "arguments" contains the output arguments that
// the "prepare" method of the interactive activity returns
// In the example, one of the output argument is called
// "preArgOut". The local variable "info" is set with the
// information contained in it

String info = (String) arguments.getArgument("preArgOut");
```

## Commit method

```
// The commit FBL expects to receive an argument
// called "comArgIn". No arguments are returned
// from the FBL.
// Sets the "comArgIn" argument with the "info"

arguments = Arguments.create();
arguments.putArgument("comArgIn", info);

// Executes the "commit" method of the interactive
// activity (implemented as External)
session.commitExternalActivity(instance.getId(),
                               instance.getActivityName(),
                               arguments);
```

## Run an activity

### Previous Steps

- Get a SESSION
- Get an INSTANCE

Interactive activity includes: Global activity (that manages instances), Global creation activity and Interactive activity.

```
// Run an activity.
// The activity's output arguments are
// received in "arguments"
session.runActivity(instance.getId(),
```

```
        instance.getActivityName(), arguments);  
  
// The activity returns an output argument called  
// "message".  
// The local variable "info" is set with the  
// information contained in it  
String info = (String) arguments.getArgument("message");
```

### Another example

```
instance = session.getInstance(instance.getId());  
  
fuego.papi.Arguments arguments = Arguments.create();  
arguments.putArgument("inArgument", "MyInArgument");  
arguments.putArgument("inOutArgument", "MyInOutArgument");  
instance = session.runActivity(instance.getId(),  
                                instance.getActivityName(), arguments);  
  
String result = arguments.getStringArgument(Arguments.RESULT);  
String action = arguments.getStringArgument(Arguments.ACTION);  
String inOut = arguments.getStringArgument("inOutArgument");  
String in = arguments.getStringArgument("inArgument");
```

*Arguments.RESULT* and *Arguments.ACTION* are the predefined variables *result* and *action* returned by default from the tasks or BP-Method.

## PAPI Examples

Find the following projects and java examples that use PAPI in the FuegoBPM Enterprise installation directory within the *client/papi/samples* directory.

### AuditTrail.fpr

This project is used to create events that can be recovered using the **AuditTrail.java** (distributed in the project's root directory). You can find, as well the **Filter.java** program that is a guide on how you can set a filter to search for instances and; the **Grab.java** program that is a guide on how to grab an instance.

In these components you can find how to:

- Create a session
- Create a Filter
- Get an instance
- Get an instance's process
- Get an instance's events
- Get the Roles that managed an instance (based on Events)
- Get all Participants that worked with the instance
- Get the Activities that managed an instance (based on Events)
- Get an instance's creation and termination time
- Get an instance's Audit Trail
- Grab an instance

## Ordering.fpr

This project is used to manage Orders and Quotations. Several activities within them are invoked from the external component **Ordering.java** (distributed in the project's root directory).

In this component you can find how to:

- Create an instance
- Run a Global activity
- Run an Interactive activity implemented with an External task
- Run an Activity



---

## Chapter 3. WAPI

### Introduction to WAPI

The *Web-based Work Portal* is a Java Web Application developed following the Servlet 2.3 and JSP 1.2 specification.

WAPI, Work Portal Application Programming Interface, is an API that provides calls to interact with the servlets that govern the Work Portal logic.

WAPI provides two different ways to interact with the FuegoBPM Work Portal.

- External to the Work Portal, through the *HTML Process API*, and
- Internal to the Work Portal, through the *URLForAction Java Class*.

### HTML Process API

This feature provides you with the capability of executing actions and tasks on process instances from an HTML page using Work Portal functionality. They are:

- Global activities execution
- Instance actions execution
- Instance tasks or item execution

### Required Configuration

You have to define a participant named *guest* .

This participant becomes the anonymous FuegoBPM Studio user to

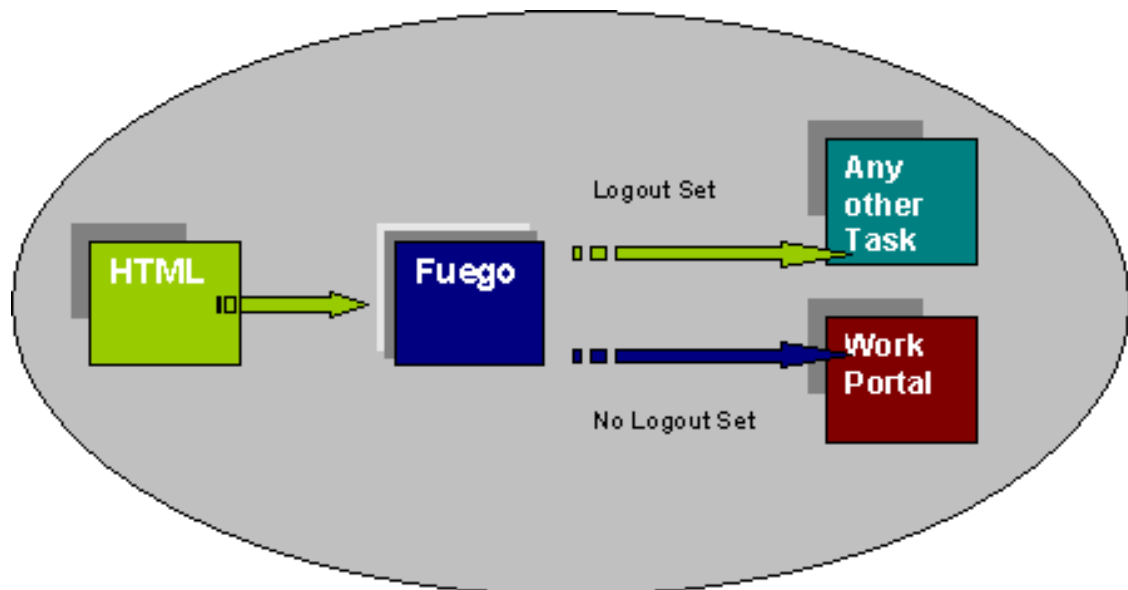
log in to Work Portal. Its definition is not mandatory, and it is only used when executing Global Creation activities. If this user is not defined, a user and password will be required at execution time.

## Description

As mentioned previously, this feature provides you with the capability of using Work Portal functionality through a URL and parameters.

Using the non-exclusive sessions (anonymous guest user) or his own (depending on the function), the final user logs in to Work Portal and performs the programmed action, task, activity. Logging out when finishing will depend on the established configuration.

Using this functionality the developer may, from his HTML page, connect and perform actions on a FuegoBPM Studio process through Work Portal and then continue with another task outside FuegoBPM Studio. Likewise, the developer may let the user stay inside Work Portal to go on working.



Work Portal invocation and execution of process activities, actions or tasks from an HTML page must always be done from an HTML Posting Form block. The structure of this block varies depending on

what is being done.

## Non-exclusive sessions / Anonymous Participant

The anonymous participant configured as the guest participant in the Organization is used to execute actions that do not require an exclusive session. That is why the non-exclusive session or anonymous participant is used only for the execution of Global activities. Its definition is not mandatory.

If the anonymous user is used to log in and to perform any activity in Work Portal, the Audit Trail will show that this participant was the one that executed the action (only valid for creation).

In order to be able to use the anonymous participant in the execution of any Global activity, it must belong to the role that matches the abstract role of the process in which the activity is defined.

The combination of two attributes within the form block code in the HTML page presents specific Portal functionality to end users. These attributes are:

- **fuego.portal.userNESession** - set to true, which requires the use of the non-exclusive session through the anonymous participant, if any, that was configured as a guest in the organization.
- **fuego.portal.logoutURL** - set with a URL, which indicates to log the user out and to go to this URL afterward.

Attribute	Data	Behavior
fuego.portal.userNESession	fuego.portal.logoutURL	
Set to TRUE	Set with a URL	The anonymous user

Attribute	Data	Behavior
		guest defined in the organization is searched for. If it is not defined, a user name and password are required through the login dialog. After the user ends processing, the session is closed and sent to the URL specified in the logoutURL attribute. <b>Note:</b> if the user sets the "Remember user" check box, next time, this login dialog is not opened.
Set to FALSE or not present	Not present	If a user name is required through the login dialog, after the completion of the processing, the user will remain logged into the Web Portal and is able to continue working within it.
Set to FALSE or not present	Set with a URL	Instead of looking for the anonymous participant, the user will always be required through the login dialog.
Set to TRUE	Not present	The anonymous participant will remain

Attribute	Data	Behavior
		logged in to the Web Portal. The final user will be able to process any activity that does not require an exclusive session, such as Global activities. When the final user intends to process any action on an instance, a participant for exclusive session will be required before continuing to work.

## Working on Enterprise environment

When you move your project using HTML process API, you need to configure the anonymous participant in the *directory.properties* file. The *guest* user is only valid in FuegoBPM Studio environment.

1. Go to the file located at :  
\$INSTALLATION\_DIRECTORY\\webapps\\portal\\WEB-INF\\directory.properties
2. Set the following properties:

```
directory.default.preset.portal-anonymous.participant=guest
directory.default.preset.portal-anonymous.
    participant_password=guest
```

Be sure that the user you are configuring in this file as anonymous

participant to log in into the FuegoBPM Work Portal has the correct roles assigned.

Do not forget to configure the *non-exclusive session* as well in the *portal.properties* file as in the Studio environment.

## User Authentication Steps

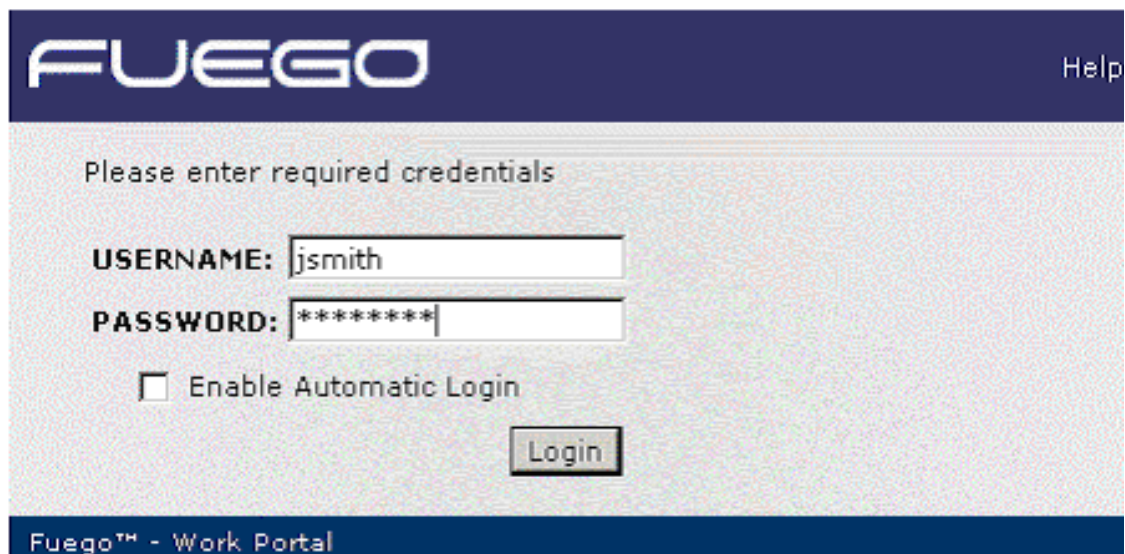
### Global Activity execution

If the attribute `fuego.portal.userNESession` is included and set to true, the definition of the anonymous user guest is searched for in the organization structure (participant and password). If the anonymous user guest is defined, this user is the one that is logged into Work Portal.

If no anonymous user has been defined or `userNESession` is set to false, then a participant is required to log in through the login dialog.

### Action and Item execution

A participant is always required to log in through the login dialog.



The screenshot shows the Fuego Work Portal login interface. At the top, there is a dark blue header with the 'FUEGO' logo on the left and a 'Help' link on the right. Below the header, the text 'Please enter required credentials' is displayed. There are two input fields: 'USERNAME:' with the value 'jsmith' and 'PASSWORD:' with the value '\*\*\*\*\*'. Below these fields is a checkbox labeled 'Enable Automatic Login' which is currently unchecked. A 'Login' button is positioned to the right of the checkbox. At the bottom of the dialog, there is a dark blue footer with the text 'Fuego™ - Work Portal'.

## Global Activities Execution

This option provides the capability of creating process instances or executing Global activities as queries. Global Automatic activities are not available, as this framework is a way of using Work Portal functionality and Work Portal has no way of reaching automatic activities. If a Global Automatic activity is executed, an error will be displayed.

Similar functionality can be simulated with the usage of web services or PAPI. However, the advantages of this feature over the usage of the web services or PAPI are that:

- user interaction is possible; something that cannot be done with web services
- no programming is required; something that has to be done in order to use PAPI

## HTML Form block programming

```
<form method="post" ACTION="PortalURL/servlet ">
```

*Refer to the common HTML code explanation section 4.6.*

The following attributes are generally hidden, as they are the parameters sent to the servlet.

***activityId*** :

```
<INPUT TYPE="hidden" name="activityId"
value="ou/processName/variation-version/activityName">
```

The *activityId* attribute represents the process Global activity that will

be executed. The *activityId* value always has the following structure:

- ou: Organizational Unit in which the process has been deployed. If the process is deployed for non-specific ou, leave it blank.
- processName: name of the Process which the Global activity belongs to.
- variation-version, where:
  - variation: variation given when the process was deployed.
  - version: version given when the process was deployed.

### Note



Variation and version have to be concatenated by a dash character "-".

- activityName: name of the Global activity to execute.

### Example: with Organizational Unit.

```
<INPUT TYPE="hidden" name="activityId"
      value="/BuenosAires/OrderFulfillment#
            Default-1.0/verifyCredit">
```

where

- ou = BuenosAires,
- processName = OrderFulfillment
- variation\_version = Default-1.0 (variation = Default, version =



1.0)

- activityName = verifyCredit

**Example: without Organizational Unit.**

```
<INPUT TYPE="hidden" name="activityId"
      value="/OrderFulfillment#Default-1_0/verifyCredit">
```

where

- ou = ,
- processName = OrderFulfillment
- variation\_version = Default-1.0 (variation = Default, version = 1.0)
- activityName = verifyCredit

***actionId:***

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL">
```

The actionId attribute will always have the value "RUN\_GLOBAL" when a global activity is executed.

***fuego.portal.useNESession:***

```
<INPUT TYPE="hidden"
      name="fuego.portal.useNESession" value="true">
```

The `fuego.portal.userNESession` attribute set to true enables the intention of using the defined anonymous participant. If it is not defined, then a FuegoBPM participant will be required through the login dialog.

***fuego.portal.logoutURL:***

```
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURL" value="logoutOk.html">
```

*Refer to the common HTML code explanation section 4.5.*

***fuego.portal.logoutURLError:***

```
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURLError"
      value="logoutError.html">
```

"Refer to the **HTML common code** explanation."

***Arguments:***

Arguments are included in the attributes list if the Global activity to be executed has defined input arguments. If the activity arguments have default values and they are not sent from the HTML page, the default will be used. The argument attributes in the form block of the HTML page may be hidden or not, according to the application.

There is a rule to name the arguments in the form block.

"arg\_" + argumentName

Suppose that the activity receives two arguments, let's name them `argument1` and `argument2`. Then, the name of the attributes in the form block will be: **arg\_argument1** and **arg\_argument2**.

```
Argument1: <input type="text"
              name="argArgument1">
Argument2: <input type="text"
              name="argArgument2">
```

*Finally the known submit button.:*

```
<INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="create a instance">
</form>
```

### Note










See the **Examples** section to find a complete example.







## Execute an action for an Instance

This option allows the execution of any action that is provided by Work Portal. The final user always has to log in to Work Portal with a participant user name and password.

The list of possible actions is:

Action Name	Work Portal Icon	Description
PROCESS		Processes the activity. If the activity has more than one task, then the instance panel is displayed so that the user can process any or all of its tasks.
COMPLETE		Moves the instance to the next activity in the

Action Name	Work Portal Icon	Description
		process. Dimmed until all mandatory tasks for an instance at the current activity are processed.
SEND_TO		Allows the user to assign the instance to a specific user and sends it to the next activity in the process.
BACK		Allows the user to return to the activity where an exception occurred and to continue processing the instance from that point. <b>Note:</b> This button is only available provided that an exception handler was included in the design of the business process and an exception in the instance invoked the exception handler.
ABORT		Terminates and deletes an instance. All processing ceases and the instance is removed from the business process.
SUSPEND		Pauses the instance at the indicated activity.
RESUME		Resumes the instance and allows it to continue.

Action Name	Work Portal Icon	Description
		running.
GRAB		Sends an instance to a Grab activity.
UNGRAB		Releases an instance from a Grab activity.
SELECT		Assigns the instance to the participant for that activity and blocks other users from processing the instance.
UNSELECT		Unselects the instances that the participant has selected to work.
SELECT_ITEM		The participant selects a task or item within an activity for a specific instance to work on it.
UNSELECT_ITEM		The participant frees the task on an instance that had been previously selected.

## HTML Form block programming

For the lines below, please refer to the **HTML common code** explanation section.

```
<form method="post" ACTION="servlet/instanceActions">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURL" value="logoutOk.html">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURL_Error"
      value="logoutError.html">
```

In general, the attributes described below are hidden, but it will depend on the application.

*instanceStampId:*

```
<INPUT TYPE="hidden" name="instanceStampId"
      value="instanceStampIdValue">
```

The instanceStampId is the name of the instance. The instanceStampIdValue must be made up of:

- processName
- variation\_version, ( variation and version have to be concatenated by a dash character "-")
- instanceNumber
- thread (number of copy)
- activityName, for example  
"instanceActions/Default-1\_0/6/0/firstActivity"

A way of composing this value is using a Method, if possible depending on the process and application. Truncating the value of the instanceID instance variable from the "@" character and concatenating the activity name, described as follows:

```
instanceStampId = instanceId.substring(0,
      instanceId.indexOf("@")) +
      "/confirmSubscription";
```

```
<INPUT TYPE="hidden" name="actionId" value="actionIdValue">
```

***actionId:***

The actionId attribute represents Work Portal functionality intended to execute. The actionIdValue must contain any of the possible actions in the table described in the previous section.

- PROCESS
- COMPLETE
- SEND\_TO
- BACK
- ABORT
- SUSPEND
- RESUME
- GRAB
- UNGRAB
- SELECT
- UNSELECT
- SELECT\_ITEM
- UNSELECT\_ITEM

When Action ID is SELECT\_ITEM or UNSELECT\_ITEM

```
<INPUT TYPE="text" name="itemId" value="0">
```

Represents the number of tasks comprised within the Activity tasks/items list. The list starts in 0.

When Action ID is GRAB or UNGRAB

```
<INPUT TYPE="text" name="grabActivityId"
value="grabActivity">
```

- **Grab:** represents the name of the grab activity in the process where the instance must be sent to. (A process might contain more than one grab activity.)
- **Ungrab:** represents the name of the grab activity by which the instance has been grabbed.

Finally, the submit button:

```
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action">
</form>
```

### Note



See the **Examples** section to find a complete example.

## Item/task execution

This option allows the execution of a task for an instance in an Activity. It is only valid if the activity has at least one defined task.

The execution of an item may send arguments as the execution of



global activities. Arguments will be included in the attributes list. The argument attributes in the form block of the HTML page may be hidden or not, according to the application.

There is a rule to name the arguments in the form block.

"arg\_" + argumentName

Suppose that the activity receives two arguments, let's name them argument1 and argument2. Then, the name of the attributes in the form block will be: **arg\_argument1** and **arg\_argument2**.

```
Argument1: <input type="text" name="argArgument1">  
Argument2: <input type="text" name="argArgument2">
```

## HTML Form block programming

For the lines below, please refer to the **HTML common code** explanation section.

```
<form method="post" ACTION="servlet/ExecutionDispatcher">  
<INPUT TYPE="hidden"  
      name="fuego.portal.logoutURL" value="logoutOk.html">  
<INPUT TYPE="hidden"  
      name="fuego.portal.logoutURL_Error" value="logoutError.html">
```

***actionId:***

```
<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM">
```

The actionId attribute will always have the value "RUN\_ITEM" when an activity task is the one to be executed.

***instanceStampId:***

```
<INPUT TYPE="hidden" name="instanceStampId"
      value="instanceStampIdValue">
```

The `instanceStampId` represents the exact identification of an instance in a process.

The `instanceStampIdValue` has to be made up of:

- `processName`
- `variation-version` (variation and version have to be concatenated by a dash character "-")
- `instanceNumber`
- `thread` (number of copy)
- `activityName`, for example  
"/instanceActions#Default-1.0/6/0/firstActivity"

A way of composing this value is using a Method statement, if possible depending on the process and application. Truncating the value of the `instanceID` instance variable from the "@" character and concatenating the activity name, as follows:

```
instanceStampId = instanceId.substring(0,
      instanceId.indexOf("@")) +
      "/confirmSubscription" ;
```

***taskDesc:***

```
<INPUT TYPE="hidden" name="taskDesc" value="taskName">
```

The attribute taskDesc must contain the name of the activity task to be executed.

*itemId:*

```
<INPUT TYPE="hidden" name="itemId" value="0">
```

Represents the number of tasks within the Activity tasks/items list. The list starts in 0.

*Finally the submit button* :

```
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action">
</form>
```

## Note



See the **Examples** section to find a complete example.

## HTML common code

This section explains the common code of the HTML form block.

See each option HTML code to contextualize it.

*post:*

It is recommended to use the "post" method.

```
<form method="post" ACTION="PortalURL/servlet ">
```

***ACTION:***

ACTION must refer to the ExecutionDispatcher servlet, that is, Work Portal servlet in charge of instances execution.

- PortalURL: varies according to Work Portal to which is required to connect. For example: *http://localhost:8080/portal*
- servlet: depends on the action being executed
  - For **PROCESS, COMPLETE, SEND\_TO, BACK, ABORT, SUSPEND, RESUME, GRAB, UNGRAB, SELECT, UNSELECT, SELECT\_ITEM, UNSELECT\_ITEM** the URL should use the *instanceActions*.
  - For **RUN\_ITEM y RUN\_GLOBAL** the URL should use the *executionDispatcher*.
- The complete value for the ACTION attribute could be:

```
ACTION=  
"http://localhost:9595/htmlProcessAPI/servlet/  
instanceActions"  
or  
ACTION=  
"http://localhost:9595/htmlProcessAPI/servlet/  
executionDispatcher"
```

***fuego.portal.logoutURL:***

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="logoutOk.html">
```

The `fuego.portal.logoutURL` attribute is not mandatory. And it indicates which is the html to be displayed when finished.

***fuego.portal.logoutURL*****Error**:

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="logoutError.html">
```

The `fuego.portal.logoutURL` attribute is not mandatory and it indicates which is the html to be displayed in the event that an error occurs. If it is not defined, then the one defined in the `logoutURL` attribute will be used.

For Examples, please refer to HTML Process API Examples.

## HTML Process API Examples

The processes examples and the set of html pages are distributed with FuegoBPM Studio installation. You can find them under the *sample* directory of the installation.

### Three options

#### Process

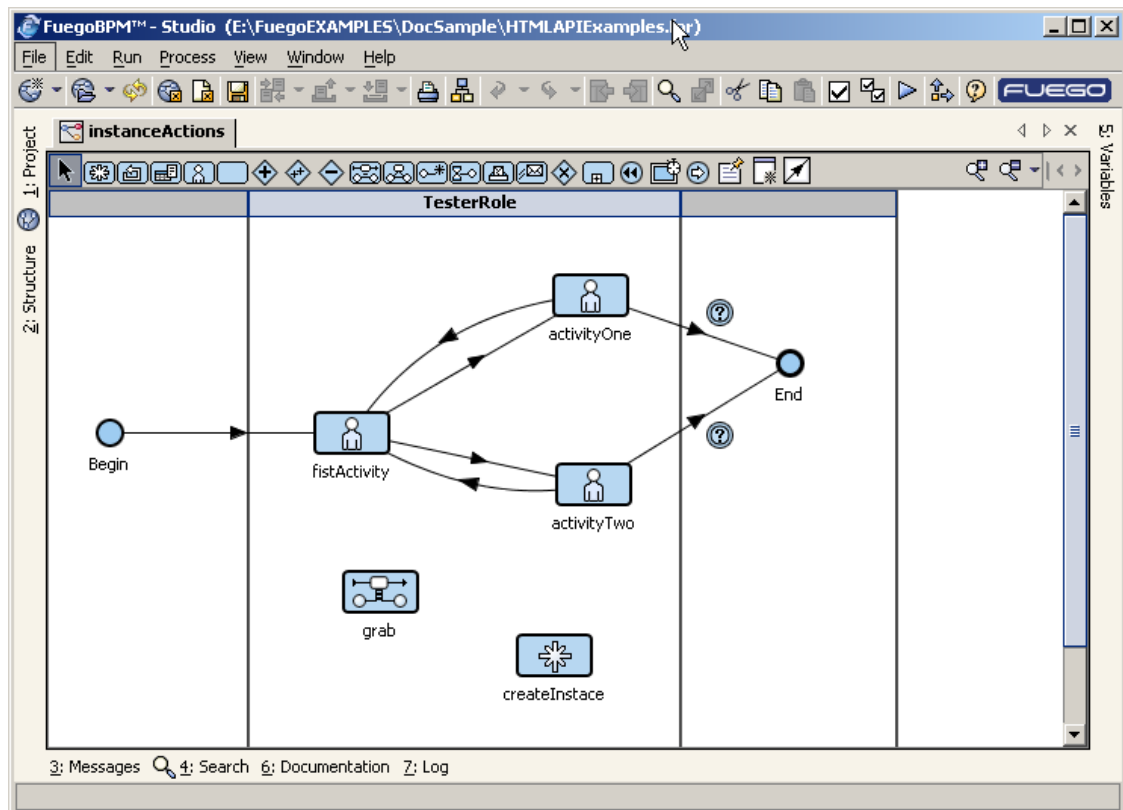
The example process has been designed by modeling the three possible actions to execute from an HTML page using the framework.

The global activities execution has been modeled through a Global Creation activity.

The flow of the process allows for the execution of different actions

for an instance, such as complete, process, select, grab, back, and so on.

The item execution example is contained in the activity *firstActivity*, as it has two tasks to perform.

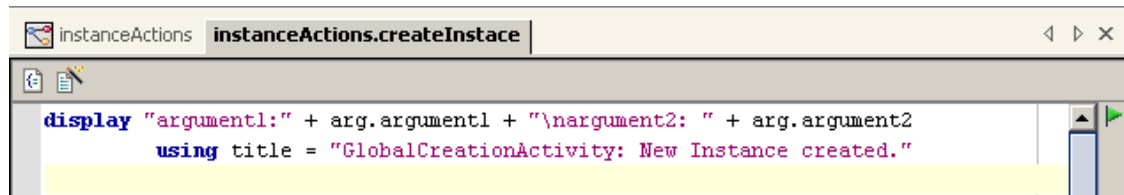


## Business Process Methods

### *Global Activities Execution*

As previously mentioned, the example is based on a Global Creation activity. The `createInstance` activity receives two arguments, `argument1` and `argument2`. These arguments have to be sent as parameters from the form post method of the HTML pages. Remember that those arguments from the html should always bear the "arg\_" prefix (`arg_argument1.`) (\*).

The Method is very simple, since it only displays the values received in the argument variables.



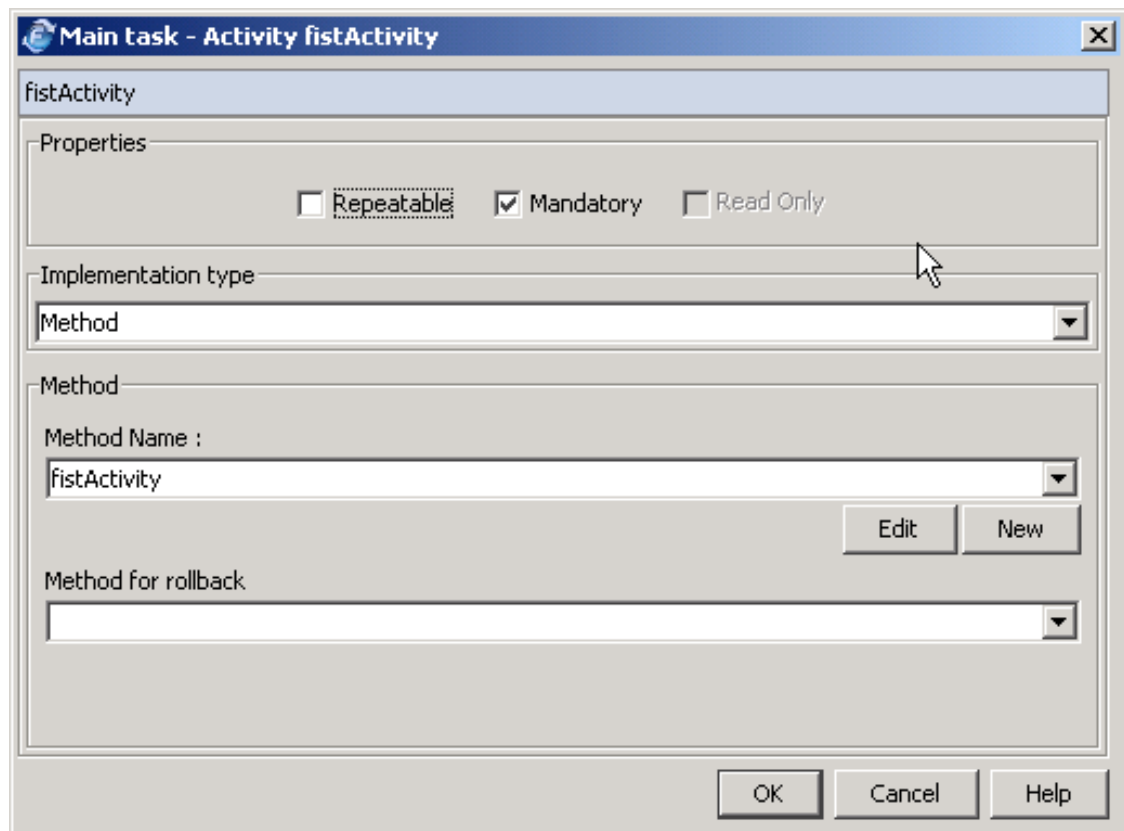
Reference (\*) about arguments is related to reference (6) of the HTML pages section.

## Instance Action Execution

The simulation is done with an instance located in the activity *firstActivity* of the process. Many actions can be executed on it.

## Instance Task Execution

The process simulates this option through the task *firstTask*, in the *firstActivity*.



The task's BP-Method only contains a display sentence to verify that it is being executed.

```
display "First Task"
```

## HTML pages

### InstanceActions

This HTML page contains three sections. Each section corresponds to each execution option available for use.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Custom Logout Url Test</h1>

<!-- Global Creation Activity-->
<hr>
<h3>Test Global creation using anonymous access.</h3>
(1) <form method="post"
    ACTION=
"http://localhost:8585/portal/servlet/ExecutionDispatcher">
(2) <INPUT TYPE="hidden" name="activityId"
    value="/instanceActions#Default-1.1
        /createInstance" />
(3) <INPUT TYPE="hidden" name="actionId"
    value="RUN_GLOBAL" />
(4) <INPUT TYPE="hidden"
    name="fuego.portal.useNESSession" value="true"/>
(5) <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
    value="../instanceActionsLogoutOk.html"/>
    <INPUT TYPE="hidden"
    name="fuego.portal.logoutURL_Error"
    value="../instanceActionsLogoutError.html" >
(6) Argument1:<input type="text" name="arg_argument1"><br>
    Argument2:<input type="text" name="arg_argument2"><br>
    <br>
    <INPUT TYPE="SUBMIT" NAME="Submit"
    VALUE="create an instance" />
</form>
```



```

<hr>

<!-- Instance Actions -->
<h3>Test Instance actions</h3>
(1) <form method="post"
    ACTION=
"http://localhost:8585/portal/servlet/instanceActions">
(5) <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
    value=" ../instanceActionsLogoutOk.html" />
    <INPUT TYPE="hidden"
        name="fuego.portal.logoutURLerror"
        value=" ../instanceActionsLogoutError.html" />
    <table>
    <tr>
        <td>instanceStampId:</td>
(7) <td><INPUT TYPE="text" name="instanceStampId"
        value="/instanceActions#Default-1.1
            /11/0/fistActivity"
            size="40"></td>
    </tr>
    <tr>
        <td>actionId:</td>
(8) <td><select name="actionId">
        <option>PROCESS</option>
        <option>COMPLETE</option>
        <option>SEND_TO</option>
        <option>BACK</option>
        <option>ABORT</option>
        <option>SUSPEND</option>
        <option>RESUME</option>
        <option>GRAB</option>
        <option>UNGRAB</option>
        <option>SELECT</option>
        <option>UNSELECT</option>
        <option>SELECT_ITEM</option>
        <option>UNSELECT_ITEM</option>
        </select>
        </td>
    </tr>
    <tr>
        <td>itemId:</td>
(9) <td><INPUT TYPE="text" name="itemId" value="0"/>
    (Note: Only used in SELECT_ITEM and UNSELECT_ITEM)
    </td>
    </tr>
    <tr>
        <td>grabActivityId:</td>
(10) <td><INPUT TYPE="text" name="grabActivityId"
        value="grab"/>
        (Note: Only used in GRAB and UNGRAB)
    </td>

```

```

        </tr>

        </table>
        <br>
        <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action"/>
        </form>
        <hr>

        <!-- Item Execution -->
        <h3>Run Item</h3>
(1)  <form method="post"
        ACTION=
"http://localhost:8585/portal/servlet/ExecutionDispatcher">
(5)  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
        value="../instanceActionsLogoutOk.html"/>
        <INPUT TYPE="hidden"
        name="fuego.portal.logoutURLError"
        value="../instanceActionsLogoutError.html"/>
        <table>
(11)<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM"/>
        <tr>
                <td>instanceStampId:</td>
(7)  <td><INPUT TYPE="text" name="instanceStampId"
        value="/instanceActions#Default-1.1
        /1/0/fistActivity"
        size="40"></td>
        </tr>
        <tr>
                <td>taskDesc:</td>
(12) <td><INPUT TYPE="text" name="taskDesc"
        value="firstTask"/>
        </td>
        </tr>
        <tr>
                <td>itemId:</td>
(12) <td><INPUT TYPE="text" name="itemId" value="0"/>
        </td>
        </tr>

        </table>
        <br>
        <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action"/>
        </form>
        <hr>

</html>

```

***instanceActionsLogoutOK***

This is the page specified in the logout ok URL of the main page. In any of the three processing options in the main html page, instanceActions, the instanceActionsLogoutOk HTML would be displayed when the execution ends without error conditions.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Instance Action Test</h1>

Action successfully completed.

<form action="instanceActions.html">
<input type="submit" value="continue">
</form>

</html>
```

### *instanceActionsLogoutError*

This is the page specified in the logout error URL of the main page. In any of the three processing options in the main html page, instanceActions, the instanceActionsLogoutError HTML would be displayed when the execution ends under error condition.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Instance Action Test</h1>

Action completed <b>unsuccessfully</b>.

<form action="instanceActions.html">
<input type="submit" value="continue">
</form>

</html>
```

---

## References

### *INSTANCE CREATION SECTION:*

- **(1)**

```
<form method="post"
ACTION="http://localhost:8080/portal/
        servlet/ExecutionDispatcher">
```

The use of the execution dispatcher of portal local host is being indicated.

- **(2)**

```
<INPUT TYPE="hidden" name="activityId"
value="/instanceActions#Default-1.1/createInstace"/>
```

The global activity to execute is the createInstace activity that belongs to the instanceAction process (see the Process section for further details). Change the variation and version, to the corresponding to your deployed process.

- **(3)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL"/>
```

The actionId should always have the value RUN\_GLOBAL to execute a global activity.

- **(4)**

```
<INPUT TYPE="hidden" name="fuego.portal.useNESession"
      value="true"/>
```

The intention to use the anonymous participant, if it is defined, is being indicated. If it is not defined, a FuegoBPM participant will be required through the Work Portal login dialog.

- **(5)**

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
value="../instanceActionsLogoutOk.html"/>
<INPUT TYPE="hidden" name="fuego.portal.logoutURLError"
value="../instanceActionsLogoutError.html"/>
```

If the execution ends without or under error condition, these are the HTML pages to be displayed.

- **(6)**

```
Argument1: <input type="text" name="arg_argument1">
Argument2: <input type="text" name="arg_argument2">
```

This is the argument section. The global creation createInstance activity receives two arguments as parameters. In the HTML page,

the values for this parameters are requested to the end user. This reference is related to the (\*) reference in the Method section.

### Note



See how the argument name is composed when passing the parameter through the form block (arg\_ + argumentName.)

#### ***INSTANCE ACTION SECTION:***

- **(7)**

```
<INPUT TYPE="text" name="instanceStampId"
      value="/instanceActions#Default-1.1/11/0/fistActivity"
      size="40">
```

The instanceStampId is being set, where:

- process : instanceActions,
- variation : Default
- version : 1\_1
- instance number : 11
- instance copy : 0
- activity : fistActivity.

If you try to execute this example, remember to change these values according to the specific moment you are testing. The variation and version may vary depending on how many times you have deployed the project and activity name may vary while instance flows through the process. The instance number may vary if you are testing the

same action process on new created instances.

- **(8)**

```
<select name="actionId">
    <option>PROCESS</option>
    <option>COMPLETE</option>
    <option>SEND_TO</option>
    <option>BACK</option>
    <option>ABORT</option>
    <option>SUSPEND</option>
    <option>RESUME</option>
    <option>GRAB</option>
    <option>UNGRAB</option>
    <option>SELECT</option>
    <option>UNSELECT</option>
    <option>SELECT_ITEM</option>
    <option>UNSELECT_ITEM</option>
</select>
```

The sample HTML page contains the list of possible actions to be selected by the end user. It will depend on the application, but generally this should be a hidden attribute.

- **(9)**

```
<INPUT TYPE="text" name="itemId" value="0"/>
```

In this example, the task to be executed is the first in the list of the activity.

- **(10)**

```
<INPUT TYPE="text" name="grabActivityId" value="grab"/>
```

In this example, the grab activity defined in the process to where the instance will be sent is named “grab”.

***RUN ITEM SECTION:***

- **(11)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM"/>
```

When an activity task is being executed, the actionId attribute should always have the RUN\_ITEM value.

- **(12)**

```
<INPUT TYPE="text" name="taskDesc" value="firstTask"/>
```

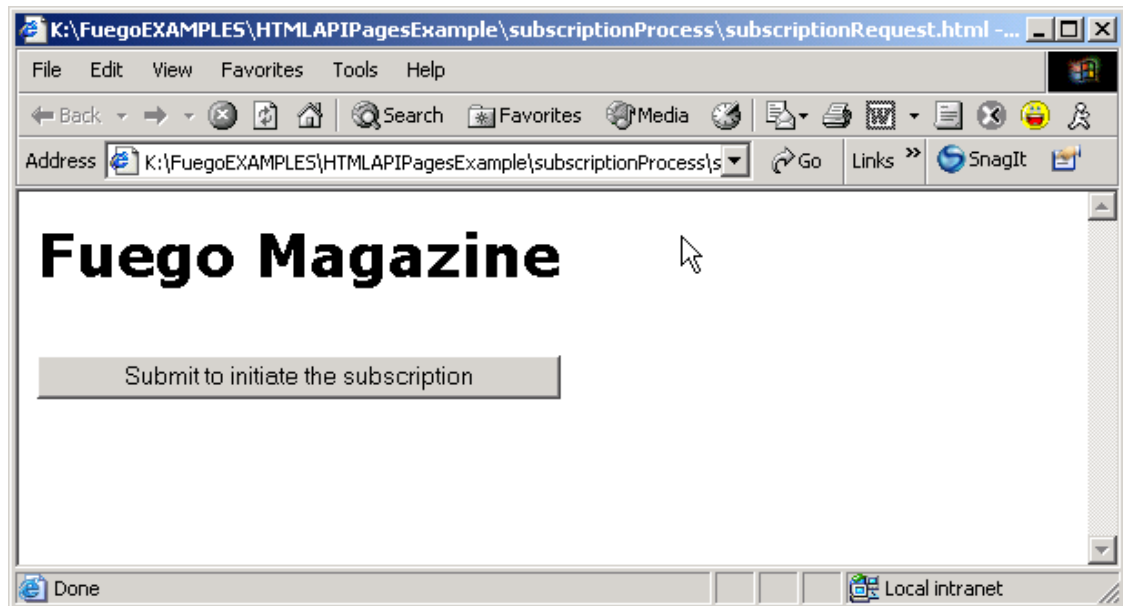
The task that will be executed in this example is the firstTask task of the fistActivity (see the Process section for further details).

## Subscription process example

### Process

The example process simulates the subscription to a Magazine. The company has decided to provide a web solution to allow users to subscribe to the magazine.





The input and approval of each subscription is executed and controlled by a FuegoBPM process. To input the subscription the user executes a global creation activity from a web page.

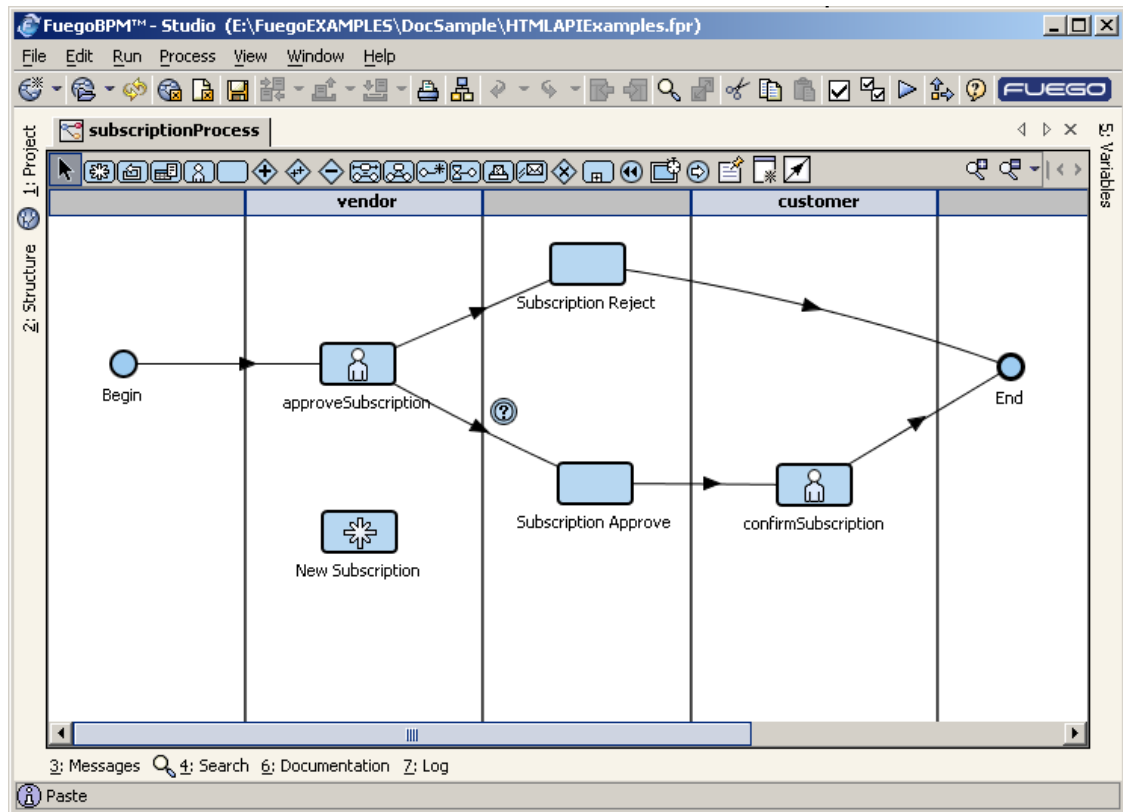
Using the framework interface the user creates an instance in the FuegoBPM process through the *NewSubscription* global creation activity, which invokes and executes the *inputSubscription* screenflow.

The approval is done through the *approveSubscription* activity. According to the decision made by the analyst to approve the subscriber or not, such person receives a mail explaining the decision. This activity executes the *approveSubscripSC*.

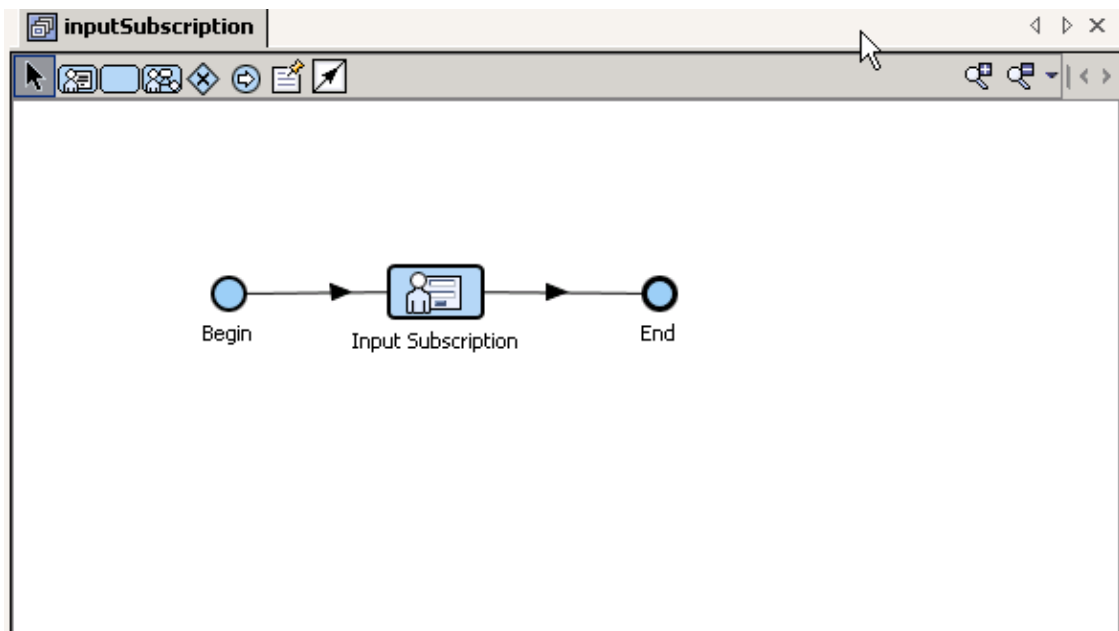
Two automatic activities, *Subscription Approve* and *Subscription Reject*, are the ones that prepare the final interaction with the user. If the subscription has been approved the mail will contain a final html code so that the user can accept or reject the terms of subscription and confirm it.

If it is rejected, the mail will contain a text explaining that the subscription has not been approved.

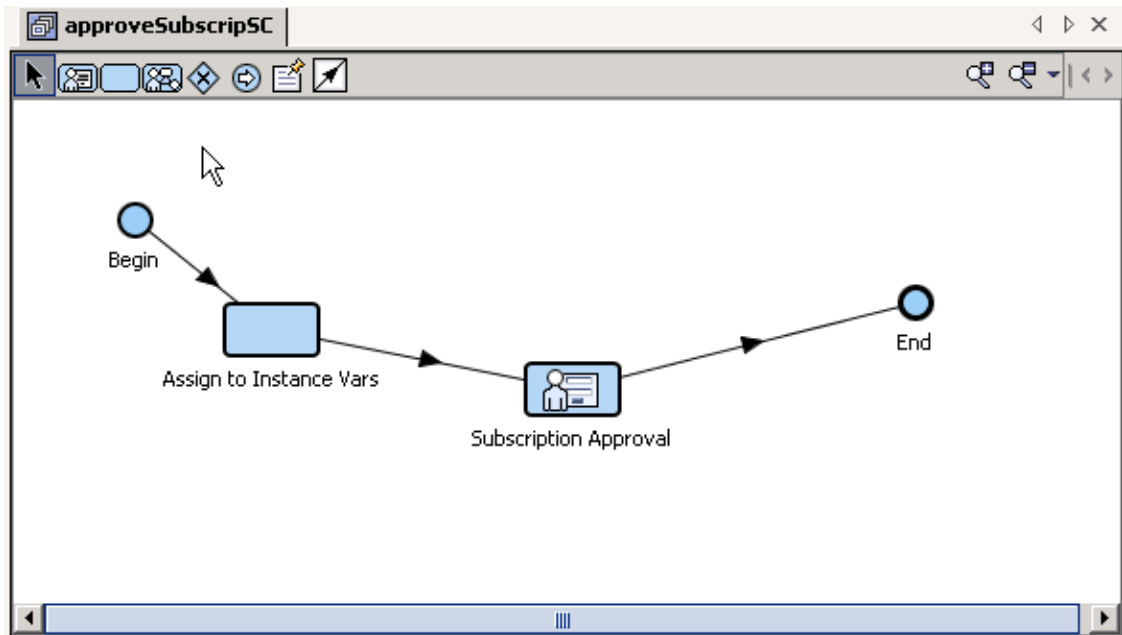
## **subscriptionProcess**



## inputSubscription screenflow



## approveSubscripSC screenflow

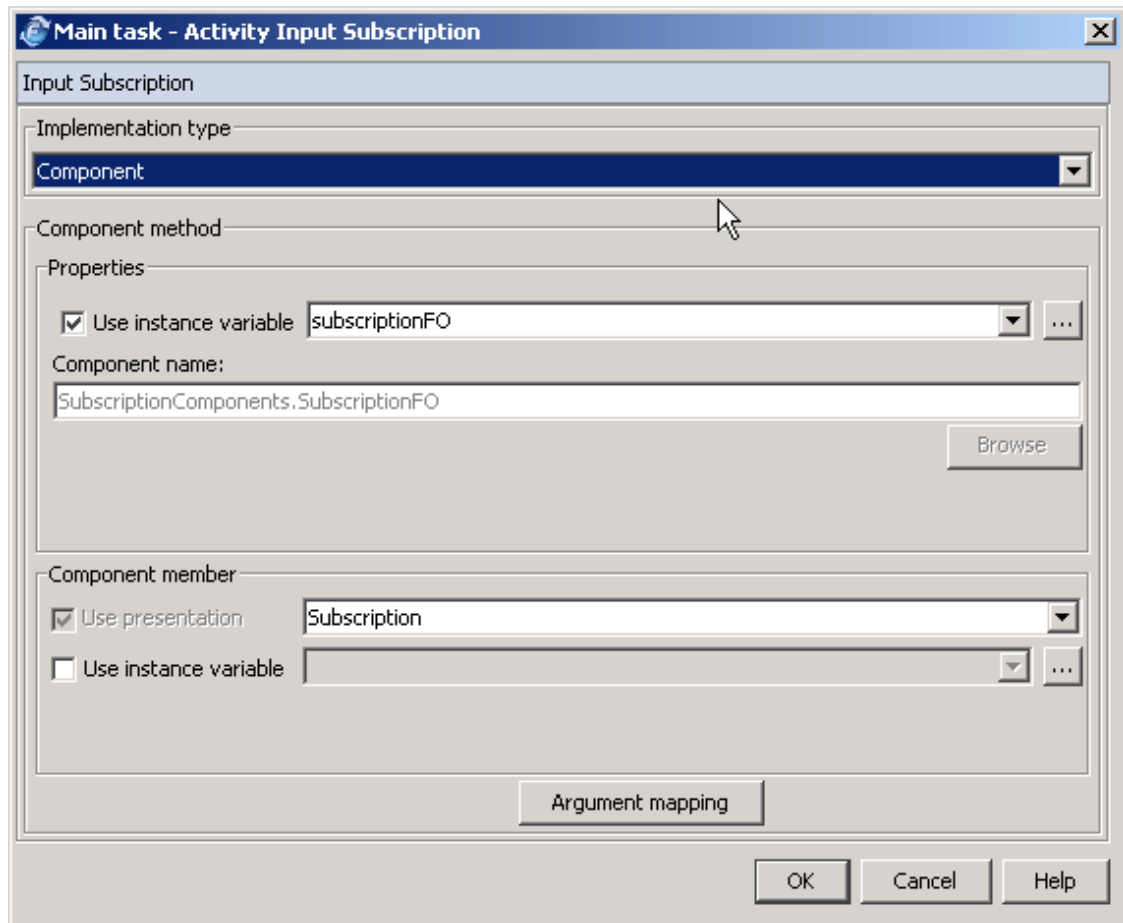


## Business Process Methods

### *NewSubscription Activity:*

This activity has no Business Method related. It invokes the *inputSubscription* screenflow.

Through the *Input Subscription* interactive component call of the screenflow the required data is input by using the SubscriptionFO Fuego Object.



***approveSubscription Activity:***

The screenflow *approveSubscripSC* receives as an input argument the Fuego Object containing the subscription data, which is loaded into an instance variable (see Argument mapping).

The *Subscription Approval* activity of the screenflow, displays the information and requires an approval or rejection.

**Main task - Activity Subscription Approval**

Subscription Approval

Implementation type: Input

Input dialog title: Subscription Approval. Instance Variables

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	Surname	participantSurname (String)	Text
<input checked="" type="checkbox"/>	Mail	participantMail (String)	Text
<input checked="" type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input checked="" type="checkbox"/>	Fax	participantFax (String)	Text

Buttons: Approve Cancel Reject

Assign selected button to: button (String)

Cancel button is:

OK Cancel Help

According to the decision taken by the analyst, approve or reject, the selected button is send to the caller activity of the main process as the result to send the subscription to automatic activities that send a message to the subscriber informing if his/her request has been approved or not.

### ***Subscription Approve Activity***

Prepares the mail to be sent to the subscriber indicating that the subscription has been approved. Asks for the final acceptance or rejection of the terms and conditions and creates the FuegoBPM participant.

```
// Mail body.
mailBody = "<html><head><body style='font-family:
           Verdana, Arial, Helvetica'>"
mailBody = mailBody +
           "Hi," + participantName + "<br><br>"
```

```
mailBody = mailBody +
    "Your subscription was successfully approved.
    If you accept the terms, you must
    click the accept
    button and login one time.<br>"
mailBody = mailBody +
    "<form method='post' action='" +
        formAction + ">"
mailBody = mailBody +
    "<input type='hidden' name='instanceStampId'
        value='" +
        instanceStampId + ">";
mailBody = mailBody +
    "<input type='hidden' name='fuego.portal.logoutURL'
        value='" +
        logoutUrl + ">" ;
mailBody = mailBody +
    "<input type='hidden'
        name='fuego.portal.logoutURLerror' value='" +
        logoutUrlError + ">";
mailBody = mailBody +
    "<input type='hidden' name='actionId'
        value='COMPLETE'>";
mailBody = mailBody +
    "<input type='submit' name='accept'
        value='ACCEPT'>";
mailBody = mailBody + "</body></html>"

do
    session = DirectorySession.currentEngineSession

    ou = DirOrganizationalUnit.fetch(dir : session,
                                    id : "Fuego Inc./test")
    role = DirOrganizationalRole.fetch(session : session,
                                      id : "ExternalUser")

    roleAss[] = RoleAssignment.create(role : role,
                                      permissions : 0)

    dirHumanParticipant = DirHumanParticipant.create(
        session : session, id : participantId,
        firstName : participantName,
        lastName : participantSurname,
        displayName : participantName + " " +
            participantSurname,
        mail : participantMail,
        telephone : participantTelephone,
        fax : participantFax,
        password : participantPassword,
        ou : ou, rolesAssignment : roleAss,
        enabled : false)
```

```
    logMessage "SEVERE"
        using severity = SEVERE

    // Now send a mail to confirm the subscription.
    send Mail
        using from = "gloria@fuegolabs.com",
            subject = "Subscription Approved",
            message = mailBody,
            contentType = "text/html",
            @to = participantMail
    on e as Exception
        logMessage "Problems creating participant: " +
            e.message
            using severity = INFO
    end
```

```
    // Now send a mail to confirm the subscription.
    send Mail using to =participantMail,
        from = "magazine@fuego.com",
        subject = "Subscription Approved",
        message = mailBody,
        contentType = "text/html"

    on Exception
        logMessage "Problems creating participant: " +
            Exception.message using severity = INFO
    end
```

### ***Subscription Reject Activity***

Sends the mail to the subscriber indicating that his application has been rejected and sets the action to *ABORT* to terminate the instance in the process.

```
action = ABORT
send Mail using to =participantMail,
    from = "magazine@fuego.com",
    subject = "Subscription Rejected",
    message = "Hi," + participantName +
        "\n\n Your subscription has been rejected. \n Sorry",
```

```
contentType = "text/html"
```

### *confirmSubscription Activity*

It does not have a related Business Process Method. This activity is the one in which the instance resides after the mail indicating the approval of the subscription has been sent to the user. When he/she executes the final action, the instance will flow to the *End* activity.

## HTML pages

### *subscriptionRequest page*

This is the page used by the final user to apply for a subscription. It creates an instance in the subscriptionProcess process.

```
<HTML>

<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>

<FORM METHOD="post"
(1)  ACTION="http://localhost:8080/portal/
      servlet/ExecutionDispatcher">

(2)  <INPUT TYPE="hidden" name="activityId"
value="/subscriptionProcess/Default-1.0/createSubscription"/>

(3)  <INPUT TYPE="hidden" name="actionId"
value="RUN_GLOBAL"/>

(4)  <INPUT TYPE="hidden" name="fuego.portal.useNESession"
value="true"/>

(5)  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="../subscriptionRequestLogout.html"/>
      <INPUT TYPE="hidden"
      name="fuego.portal.logoutURLerror"
      value="../subscriptionRequestError.html"/>

<br>
(6) <INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="Submit to initiate the subscription"/>
```



```
</FORM>

</HTML>
```

### *subscriptionRequestError page*

This page is invoked from the subscriptionRequest page when the subscription was executed ending under an error condition (see reference (5) of the subscriptionRequest page code).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your Subscription process was completed
    <b>unsuccessfully</b>.
</body>

</html>
```

### *subscriptionRequestLogout page*

This page is invoked from the subscriptionRequest page when the subscription was executed ending without any error condition (see reference (5) of the subscriptionRequest page code).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your request is in process.
You will receive a mail with a notification.
</body>

</html>
```

---

### *subscriptionRequestFinish page*

This page is invoked from the HTML block in the mail sent to the user when the subscription was approved. It allows the user to know that his confirmation to the subscription has successfully ended (see reference (b) in the Method section of this example).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your Subscription process was successfully completed.
You will receive the first issue in a month.
</body>

</html>
```

## References

- (1)

```
<FORM METHOD="post"
ACTION="http://localhost:8080/portal/
servlet/ExecutionDispatcher">
```

The use of the execution dispatcher of portal local host is being indicated.

- (2)

```
<INPUT TYPE="hidden" name="activityId"
value="/subscriptionProcess#Default-1.0/createSubscription"/>
```

The global activity to execute is the *createSubscription* activity that belongs to the subscriptionProcess process. (see the Process section for further details). Through this execution, an instance is created and the approval process begins. Change the variation and version, to the corresponding to your deployed process.

- **(3)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL"/>
```

The actionId should always have the value RUN\_GLOBAL to execute a global activity.

- **(4)**

```
<INPUT TYPE="hidden" name="fuego.portal.useNESession"
value="true"/>
```

The intention to use the anonymous participant, if it is defined, is being indicated. If it is not defined, a FuegoBPM participant will be required through the Work Portal login dialog.

- **(5)**

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
```

```
value="../../subscriptionRequestLogout.html"/>
<INPUT TYPE="hidden" name="fuego.portal.logoutURLerror"
value="../../subscriptionRequestError.html"/>
```

If the execution ends without or under error condition, these are the HTML pages to be displayed respectively.

- **(6)**

Arguments input section. Submit button.

## Create a participant example

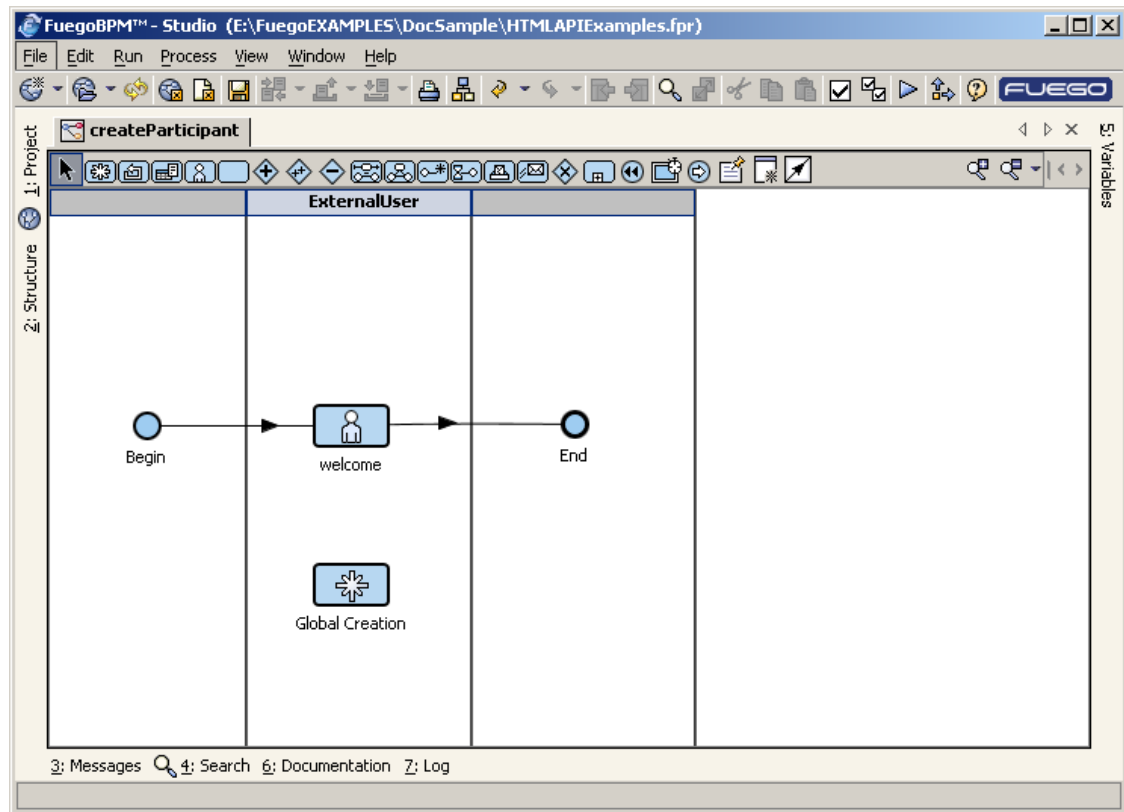
### Process

This process shows how to create a FuegoBPM participant. The Global activity create is executed by the end user from an html page. This example shows that user interaction with a FuegoBPM process from an HTML is possible.

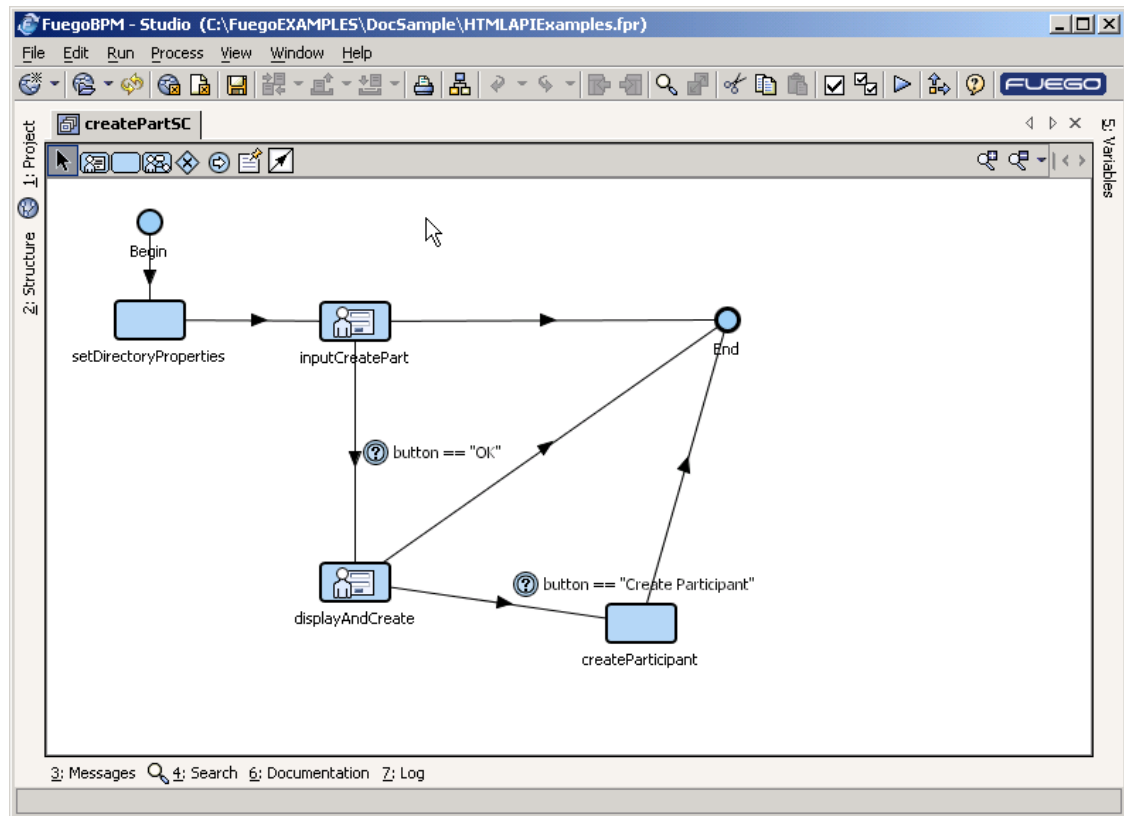
You will note that, from the HTML, the non-exclusive session will be used and as logout URL is set the one corresponding to Work Portal. This is a way of providing the end user with the possibility of creating his own participant and using it immediately after logging in to Work Portal.

The process is very simple. It implements in a Global activity a screenflow that handles the input and confirmation of the participant data. This global activity, *Global Creation*, is invoked from the HTML page.

### Process



## Screenflow



## Business Process Methods and activities definitions

The activity *Global creation* of the main process invokes the screenflow that handles the data input. Its definition is as follows:

The screenshot shows a dialog box titled "Main task - Activity Create Participant". The dialog is divided into several sections. The top section, "Create Participant", is highlighted in blue. Below it is the "Implementation type" section, which contains a dropdown menu currently showing "Screenflow" and a "Test" button. The next section is "Related Process", which includes a "Screenflow name" dropdown menu showing "createPartSC", along with "Edit" and "New" buttons. Below this is an "Argument mapping" section, which is currently empty. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

The process requires the user to input the information of the participant that he intends to create in FuegoBPM Studio. To do so, the interactive component call activity *inputCreatePart* is defined as follows:

inputCreatePart

Implementation type  
Input

Input dialog title: Participant data input. Fill in the form and press OK

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	Password	participantPassword (String)	Password
<input checked="" type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	Surname	participantSurname (String)	Text
<input type="checkbox"/>	Mail	participantMail (String)	Text
<input type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input type="checkbox"/>	Fax	participantFax (String)	Text

Buttons

Assign selected button to: button (String)

Cancel button is: Cancel

OK Cancel Help

The information just input is displayed so that the user can confirm or modify it if there are any mistakes. To do so, the interactive component activity *displayAndCreate* is defined as follows:



displayAndCreate

Implementation type: Input

Input dialog title: Participant data confirmation.

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	Surname	participantSurname (String)	Text
<input checked="" type="checkbox"/>	Mail	participantMail (String)	Text
<input checked="" type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input checked="" type="checkbox"/>	Fax	participantFax (String)	Text

Buttons:

Create Participant Cancel

Assign selected button to: button (String)

Cancel button is: Cancel

OK Cancel Help

Finally an automatic activity creates the participant using the data input by the user in the previous activities.

```

session = DirectorySession.currentEngineSession

ou = DirOrganizationalUnit.fetch(dir : session,
    id : "Fuego Inc./test")
role = DirOrganizationalRole.fetch(session : session,
    id : "ExternalUser")

roleAss[] = RoleAssignment.create(role : role,
    permissions : 0)

dirHumanParticipant = DirHumanParticipant.create(
    session : session,
    id : participantId,
    firstName : participantName,
    lastName : participantSurname,
    displayName : participantName + " " +
        participantSurname,

```

```
mail : participantMail,  
telephone : participantTelephone,  
fax : participantFax,  
password : participantPassword,  
ou : ou,  
rolesAssignment : roleAss,  
enabled : false)
```

## HTML pages

```
<HTML>  
  
<BODY>  
  
<h1>Create a Participant</h1>  
  
<FORM METHOD="get" ACTION=  
  "http://localhost:8585/portal/servlet/ExecutionDispatcher">  
  
  <INPUT TYPE="hidden" name="processId"  
    value="/createParticipant#Default-1.0"/>  
  
  <INPUT TYPE="hidden" name="activityId"  
    value="/CreateParticipant"/>  
  
  <INPUT TYPE="hidden" name="actionId"  
    value="RUN_GLOBAL"/>  
  
  <INPUT TYPE="hidden" name="fuego.portal.useNESession"  
    value="true"/>  
  
  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"  
    value="http://localhost:8585/portal/" />  
  
  <INPUT TYPE="SUBMIT" NAME="Submit"  
    VALUE="Press Button to hit 'Global Creation' Activity"/>  
  
</form>  
  
</BODY>  
  
</HTML>
```

# URLForAction Java Class

Refer to the URLForAction javadoc to get information on how to use this class.

---

# Chapter 4. Extending the FuegoBPM Work Portal

## Introduction to Extending the Portal

The *Web-based Work Portal* is a Java Web Application. It was designed using the JSP Model 2 Architecture in order to achieve a clean separation of presentation from content. This means that the Work Portal follows the MVC (Model-View-Controller) design pattern, keeping the presentation (view) in JSP pages, and encapsulating the Model and Control in a servlet library.

This design gives place to a clean way to change the presentation layer of the Work Portal. Basically, no more than the JSP pages need modifications.

The Work Portal architecture helps the developer change the views in a controlled way, and WAPI (Work Portal Application Programming Interface) provides calls to interact with the servlets that govern the Work Portal logic.

## Why not just use PAPI instead

PAPI is the Java interface used by client applications that need access to FuegoBPM processes. The applications in the FuegoBPM suite (like FuegoBPM Studio, FuegoBPM Console and FuegoBPM Work Portal) use PAPI to communicate with the process servers.

For most applications in which the user interface to the FuegoBPM system needs to be customized, personalizing the Work Portal and using WAPI is a wiser approach than using plain PAPI.

WAPI provides a higher level of abstraction on top of PAPI, and provides functionality that would otherwise be tedious to implement directly with PAPI, like:

- Authentication and PAM (Plugable Authentication Modules)
- User session control
- Instance search and filters
- Instance Attachments management
- User preferences

## Understanding the Work Portal

This chapter describes how the Work Portal is structured and explains how all the pieces work together. It provides the background knowledge needed to understand the Work Portal for later customization.

### A Web Application

The Work Portal is a standard Servlet Web Application, thus, it runs within the context of a Servlet/JSP engine. This engine provides the environment where the web application runs, and plays the role of a mediator between the HTTP server (commonly called webserver) and the web application.

FuegoBPM Enterprise provides a web application servlet embedded and configured to ease Work Portal installation. Using FuegoBPM Enterprise Administration Center you only need to configure the directory server and change some Work Portal preferences in order to give them the values that better fit your needs. See chapter **FuegoBPM Enterprise Administration Center** in the System Administration Guide documentation.

The webserver and the Servlet/JSP engine must be setup to work together. Then, the later should be configured to run the Work Portal as a web application.

As an example, take the Resin (<http://www.caucho.com>) engine. It's


a Servlet/JSP engine that can be configured to run with many web servers, like Apache (<http://httpd.apache.org>) for example. It also includes an embedded webserver for easy deployment and development.

To deploy new web apps in Resin, a new entry must be added in its XML-based configuration file. For instance, in order to deploy the Web Work Portal, an element like this one could be added into its configuration file (within a **host** element):

```
<host>
...
<!-- Work Portal 5.1 Application Entry -->
<web-app
id='/workPortal'
app-dir='/usr/local/fuego5.1/enterprise/webapps/portal'>
</web-app>
...
</host>
```

The previous web-app tells Resin about the new web application that is being added. In a few words, it specifies that the **/workPortal** URL prefix will be mapped to the web application located at the directory specified by the **app-dir** attribute.

### Note

 The directory in the previous example (line 6) is a common location for the Work Portal after its installation on a Unix system. On Windows, it would be something like the following:  
**app-dir=C:\fuego5.1\enterprise\webapps\portal**

Consult the [Fuego System Administrator's Guide] for more information on how to configure and deploy the Work Portal.

## Directory Structure

On a standard FuegoBPM Enterprise software installation, the Work Portal files are located under the

**fuego/enterprise/webapps/portal/** directory (where fuego is the Fuego installation directory). This is the Root directory of the Work Portal web application. Within this directory, the following sub-directories will be found:

- **css/** - CSS (Cascading Style Sheets) for the Work Portal. Modifying the CSS files is the easiest way to customize the look of the Portal. Within a CSS file, things like the font sizes, colors, margins and decorations can be changed. This directory includes example files with pre-defined styles for using with the Work Portal.
- **jsp/** - JSP pages which conform the views of the Work Portal.
- **img/** - Contains the set of images used in the views. The current set can be customized and changed at will, and new sets can be created.
- **WEB-INF/** - Following the Java Servlet Specification, this is the standard directory where the Work Portal web application meta-information is stored (the deployment descriptor file: WEB-INF/web.xml) together with the servlets' code (under the WEB-INF/classes/ subdirectory).
- **lib/** - External library files used by the Work Portal.
- **customjsp/** - JSP pages to be used with the fuego JSP component (from CIL).
- **wapi/** - Javadoc documentation for the WAPI interface.

For the purpose of modifying and extending the Work Portal's functionality or it's look and feel, the most relevant directories are: **css/**, **jsp/** and **img/**.

In the directory **fuego/enterprise/webapps/portal/WEB-INF** there is a file named *portal.properties*. This is the properties-file used to configure some aspects of the Work Portal. It's a plain-text

configuration file that can be modified with a text editor for customization.

## How it works

Once the Work Portal is configured and ready to run, a client can start using it by pointing the web-browser to the URL that is mapped to the Work Portal application (as configured in the webserver and servlet engine). For example:

**http://hostname/workPortal.**

When the client connects to the Work Portal URL, the index.html file in the Work Portal Root directory will be served to the client, who will be redirected to the Controller servlet, which takes control of the screen-flows from there on.

The Controller servlet, without more parameters, will start with the login page. From this point the whole Work Portal execution is, basically, a continuous flow of JSP pages that call a servlet which, in turn, displays another JSP.

Each JSP page has a defined purpose, and it gives the user links, buttons and other graphical means to perform different operations. This operations will actually be translated into calls to a servlet. This servlet will execute the operation requested (provided that all the required parameters for that operation were passed) and will call the next JSP page as needed.

Which JSP is to be called next is decided by the Controller servlet, taking into account things like the operation requested, parameters, context, possible errors and user preferences.

## Customizing the Work Portal

There are a few distinct ways to customize the Work Portal. Each one offers a different level of customization. Which ones to use depends on what needs to be accomplished.

These are the main ways the Work Portal can be customized, in



increased order of flexibility (and complexity):

- **portal.properties** file - Some of the Work Portal behavior and appearance can be customized by only changing the properties in this file.
- **CSS (Cascading Style Sheets)** - Modifying the CSS files is the easiest way to customize the look of the Work Portal. Within a CSS file, things like the font sizes, colors, margins and decorations can be changed. The big advantage is that only one file (the styles.css) needs to be modified.
- **Images** - The set of images (mainly .gif files) that the Work Portal uses can be changed as desired. This is also a very simple way of customizing the Work Portal, since it does not require programming knowledge either.
- **Modify JSP pages** - This approach is obviously more advanced than the previous two. It is both more complicated and more flexible. It involves changing the code within one or more of the standard JSP pages of the Work Portal.
- **Using WAPI calls** - This gives even more power and flexibility to the programmer. When modifying the JSP pages, or even creating new ones, WAPI calls provide access to the servlets' functionality.

## The portal.properties file

As stated before, the portal.properties configuration file is used to configure some aspects of the Work Portal. It's a self-documented plain-text configuration file that can be modified with a text editor for customization. It is located in the webapps/portal/WEB-INF/ directory.

The Work Portal reads this file to figure out, among other things, which .css file to use, the desired set of images and the set of JSPs to use. Things like temporary directories, log files, logo, session

time-out and more are also specified in this configuration file. Consult [Fuego System Administrator's Guide] for more information on how to configure and deploy the Work Portal.

## Warning



The portal.properties file is self documented and easy to understand. However, its information is required for the Work Portal to work properly. Thus, as with any configuration file, it is good practice to keep a copy of the original before doing modifications to it.

## Note



This file is read when the Work Portal starts up (when the servlet engine loads it). Thus, for any modifications to take effect the servlet engine must be told to reload the Work Portal web application.

## Application Title Logo

The default Fuego Logo located at the top left corner of the Work Portal pages can be easily changed by modifying the **fuego.portal.-apptitle.type** and **fuego.portal.apptitle.value** properties of the **portal.properties** file:

```
...
# LOGO
#####
# The apptitle.type is the type of logo.
# It can be "image" or "text"
# In case of image the value is the key in the ImageBundle
# In case of text the value is a title to be printed.
#####
fuego.portal.apptitle.type=<replaceable>image</replaceable>
fuego.portal.apptitle.value=<replaceable>LOGO</replaceable>
...
```

The logo image size is specified in the CSS file used by the FuegoBPM Work Portal. If your image has a different size, change it in the css file line *".logoimage {width: 160px; height: 23px; padding : 12px 12px 12px 12px;}"*

As the commentary explains, the value of `fuego.portal.apptitle.type` must be either `text` or `image`. The former is used to display plain text instead of an image logo.

In the case of `text`, the `fuego.portal.apptitle.value` property will contain the text to be displayed as the logo. Otherwise, when type `image` is used, it will specify the key of the image within the image bundle.

The following example will use the text *MyCompany Inc.* in place of the Fuego image logo:

```
...
fuego.portal.apptitle.type=text
fuego.portal.apptitle.value=MyCompany Inc.
...
```

## CSS

The Web Work Portal uses CSS (Cascading Style Sheets) to control the appearance and positioning of elements on the web pages.

Modifying the CSS files is the easiest way to customize the look of the Work Portal. Adapting the CSS may be enough to reflect the company standards or to match the look and feel of a particular legacy application.

The standard CSS file used by the Work Portal is **style5.css**, which is located under the **/webapps/portal/css/** directory. Within this directory, more example CSS files will be found.

The `portal.properties` configuration file contains a property that tells the Work Portal which CSS file to use. It can be modified to use another file. This is an excerpt of the standard `portal.properties` after installation:

```
...
#####
# Css
#The name of the cascade style sheet file
# This file must be in portal/css directory
# The default options are:
# style.css, style1.css, style2.css, style3.css, style4.css
#####
fuego.portal.stylesheet=style5.css
...
```

So, to make the Work Portal use a different style, it can be changed.  
Like:

```
...
#####
fuego.portal.stylesheet=ourStyle.css
...
```

This is the recommended procedure to create a personalized style-sheet file:

1. Make a copy of the style-sheet file that looks closer to what is needed. Remember that the style-sheet file should be located in the css/ directory for the Work Portal to find it.
2. Edit portal.properties again so that it uses the newly created style-sheet file.
3. Finally, make modifications to the new style-sheet file at will, until the desired look is achieved. Note that when the .css file itself is modified, there is no need to re-start the Work Portal.

For example, copy the styles.css file to mycompany.css. Edit portal.properties:

```
...
#####
# Css
# The name of the cascade style sheet file
# This file must be in portal/css directory
# The default options are:
# style.css, style1.css, style2.css, style3.css, style4.css
#####
fuego.portal.stylesheet=mycompany.css
...
```

And then, modify mycompany.css until you get the desired look and feel.

## Styles used by Work Portal

The following table can serve as a reference for modifying the styles in use by the Wprk Portal.

CATEGORY	STYLES	PURPOSE
HTML Tags	BODY, A, TD, LI, SELECT, TEXTAREA, INPUT	Redefine the look and feel of html tags
General	label, fixedlink, text-small, loginbg	Define multipurpose styles
Headers & Footers	apptitle, title, header, footer, headline1, headline2, headline3, welcome	Define the look&feel of header, footer and titles
Process	Tree ptreebg, ptreelevel0, ptreelevel1, ptreeleveldeprecatd, menulinks, menulinkshi, enulinksdeprecatd	Define the look and feel of the process tree text and its background
Dialog	dialogtitle, dialogtitle2, dialoglabel, dialogtext	Define look and feel of titles, labels and text for dialog windows

CATEGORY	STYLES	PURPOSE
Tables	tabletitle,    tabletitle2, tabletext2,    tabletitle3, tabletext3	Used for different Work Portal tables
Calendar	calendartitle1, calendartitle2, calendarOn, calendarOff	Manage the style of the calendar pop-up window

## Custom Images

The set (bundle) of images the Work Portal uses can also be customized.

The images are a mainly **.gif** files that reside somewhere in the **webapps/portal/img/** subdirectory of the Work Portal.

## Image Bundle

In order to facilitate modification, the images the Work Portal uses are not hardcoded in the JSP pages' code. Instead, a separate file (the ImageBundle file) keeps a mapping between a symbolic name for the image and the actual file name of the **.gif** file. So, making the Work Portal use different icons is just a matter of modifying that file.

The **ImageBundle** file should be located in the `/enterprise/webapps/portal/WEB-INF/classes` directory. The standard installation uses the **ImagesBundleSet1.properties** file by default.

The name of the file must end with the **.properties** suffix.

The ImageBundle file is a plain-text file in which each line follows a simple KEY=value structure.

Here is an excerpt of the default ImagesBundleSet1.properties:

```
...
ATTACH = ../img/set1/attach.gif
```

```
GRABOFF = ../img/set1/icons/graboff.gif
PROCESSOFF = ../img/set1/icons/processoff.gif
SELECTOFF = ../img/set1/icons/selectoff.gif
SUSPENDOFF = ../img/set1/icons/suspendoff.gif
RELOAD = ../img/set1/reload.gif
...
```

The KEYS are the symbolic names the Work Portal uses to identify each icon (left side of the "=" sign). The values are the actual path+file-name to the image file on disk (right side of the "=" sign).

As the previous excerpt suggests, the paths to the image files are relative to the **webapps/portal/jsp/** directory. Nothing forces the images to be in the **img/** directory, but it is a good practice to keep all Work Portal images under it to keep consistency.

### Note



Even in Windows environments, the forward slash "/" should work as a path separator.

The Work Portal knows which ImageBundle file to use because it takes it from the **fuego.portal.imageBundleFile** property specified in the **portal.properties** configuration file:

```
...
# Bundle Files #####
fuego.portal.imageBundleFile=ImagesBundleSet1
...
```

### Note



The file name specified should not include the .properties suffix. It will be automatically appended by the Work Portal before looking for the file.

## Creating a new ImageBundle

Following is the recommended procedure to create a new

## ImageBundle:

- Copy the supplied ImageBundle files into a new file. Remember that the file must be in the `classes/` directory, and its name should end with the `.properties` suffix. Example: `classes/MyCompanyImages.properties`.
- Modify `portal.properties` so that the new ImageBundle file is used instead of the standard one. Example:

```
...
# Bundle Files #####
fuego.portal.imageBundleFile=MyCompanyImages
...
```

- Create a new directory where the new icons and images will be placed. Preferably, make it a subdirectory of **img/**. Example: **img/mycompany/**.
- Copy the new image files into the directory created in the previous step.
- Modify the new ImageBundle file (**MyCompanyImages.-properties** in the previous example) so that the desired icons are taken from the new image files (located in the new subdirectory). Example: to use new **.gif** files the **ATTACH** and **RELOAD** icons:

```
...
ATTACH = ../img/mycompany/attach.gif
GRABOFF = ../img/set1/icons/graboff.gif
PROCESSOFF = ../img/set1/icons/processoff.gif
SELECTOFF = ../img/set1/icons/selectoff.gif
SUSPENDOFF = ../img/set1/icons/suspendoff.g
RELOAD = ../img/mycompany/reload.gif
```



...

Only the desired images need to be changed on a new **ImageBundle**. **Images** used on other bundles can be reused. Actually, this is one of the advantages of using a new ImageBundle file instead of just replacing the standard **.gif** files. As an example, the images in **img/misc/** are used in both standard bundles **ImagesBundleSet1.properties** and **ImagesBundleSet2.properties**.

## Modifying the JSPs

As explained previously, the JSPs that make up the Work Portal can also be modified to suit particular needs.

### Warning



You must be aware that changing the standard JSP pages that conform the Work Portal application, implies that when a patch of Work Portal is applied, all the customizations made to the standard pages should be applied to the new version of the page, otherwise your version might contain errors or problems already fixed by Fuego.

See **appendix A** for a description of how the main pages of the Work Portal are laid out visually. For a comprehensive description of each page refer to **appendix B**.

### Note



Changing the standard JSPs that conform the Work Portal requires programming knowledge on Java, JSP and some JavaScript.

## JSP Bundle

The JSPs could be changed directly. But, for greater flexibility, the Work Portal takes the set of JSPs to use from a JSPBundle file, which is similar to the ImageBundle files.

The JSPBundle file should be located in the **enterprise/webapps/portal/WEB-INF/classes** directory. The standard JSPBundle file is JSPBundle.properties. The name of the file must end with the .properties suffix.

This is an excerpt of the default JSPBundle.properties file:

```
...
INSTANCE_DETAIL = /jsp/instanceDetail.jsp
INSTANCES_VIEW = /jsp/instancesView.jsp
LOGIN = /jsp/login.jsp
LOGOUT = /jsp/logout.jsp
...
```

The KEYS are the symbolic names the Work Portal uses to identify each of the JSP pages (left side of the "=" sign). The values are the actual path+file-name to the JSP file on disk (right side of the "=" sign).

### Note



Even in Windows environments, the forward slash "/" should work as a path separator.

The JSPBundle the Work Portal will use must be specified in the portal.properties configuration file, with the **fuego.portal.jspBundleFile** property:

```
...
# Bundle Files #####
fuego.portal.jspBundleFile=JSPBundle
...
```

### Note



The file name specified should not include the .properties suffix. It will

be automatically appended by the Work Portal before looking for the file.

## Warning



It is worth note that not all of the JSPs used by the standard Work Portal have a corresponding key in the JSPBundle. That is because some *.jsp* files are included by other composite pages. This means that if any of the included pages is copied to a new filename, the reference in the parent page must be updated. For example, the dispatcherMenu.jsp is not a whole page in itself (and thus, has no key in the bundle), instead, it is included by other pages such as instancesView.jsp and attachmentsView.jsp . Similarly, footer.jsp has no key , but is included by many other pages.

## Creating a new JSPBundle

Following is the recommended procedure to create a new JSPBundle:

- Copy the supplied JSPBundle.properties file to a new file. Remember that the file must be in the **classes/** directory, and its name should end with the .properties suffix. Example: **classes/MyCompanyJSPs.properties**.
- Modify portal.properties so that the new JSPBundle file is used instead of the standard one. Example:

```
...
# Bundle Files #####
...
fuego.portal.jspBundleFile=MyCompanyJSPs
...
```

- Create a new directory where the new JSP pages will be placed. Preferably, make it a subdirectory of **jsp/**. Example: **jsp/mycompany/**.

- Copy the JSPs that need modification into the directory created in the previous step.
- Modify the new JSPBundle file (MyCompanyJSPs.properties in the previous example) so that the desired JSPs are taken from the new JSP files (located in the new subdirectory). Example: If the LOGIN and LOGOUT pages were to be modified:

```
...  
INSTANCE_DETAIL= /jsp/instanceDetail.jsp  
INSTANCES_VIEW = /jsp/instancesView.jsp  
LOGIN = /jsp/mycompany/login.jsp  
LOGOUT = /jsp/mycompany/logout.jsp  
...
```

Only the desired JSPs need to be changed on a new JSPBundle. JSPs used on other bundles can be reused. Actually, this is one of the advantages of using a new JSPBundle file instead of just replacing the standard JSP files.

## JSP Inside the JSPs

The JSPs of the Work Portal use some Java objects to interact with the servlets and the FuegoBPM Server. Basically, the types of objects used within the JSPs can be divided in:

- Standard Java objects (like java.lang.String)
- PAPI objects (such as fuego.papi.Process)
- Or, new object types introduced by the Work Portal:
  - **fuego.portal.SessionEnvironment**
  - **fuego.portal.UserOptions**

All of the top-level JSPs in the Work Portal include `waminit.jsp` (either directly or indirectly through `init.jsp`) before doing anything else. This is so because this page initializes some useful variables that are necessary on all the Work Portal pages. These include:

- **locale** - A standard `java.util.Locale` object, containing the current country and language. This reflects what the user has selected in his options screen. If there is currently no user logged in, or if the user has no Locale selected, the default values will be taken from the `portal.properties` file.
- **WamResources** - A `fuego.resources.MsgBundle` object, which contains all the text messages the Work Portal displays to the user. This makes the Work Portal a locale-sensitive application.
- **imagesUrl** - Also a `java.util.ResourceBundle` object, which is used to access the current `ImageBundle` mappings.
- **stylesheet** - A String with the name of the CSS file in use.
- **appTitleType** - A String value, as defined in `portal.properties`.
- **appTitleValue** - A String value, as defined in `portal.properties`.

### Secure/Non-secure Page

Some pages can not be displayed until the user is logged into the Work Portal system. But, pages like LOGIN, should be freely accessible.

As described earlier, some JSPs in the Work Portal include `init.jsp` (which in turn includes `waminit.jsp`) instead of including `waminit.jsp` directly. This is so because the user authentication is handled by the `init.jsp` page. Thus, pages should include `init.jsp` if they require the user to be logged in, otherwise, `waminit.jsp` is enough.

In addition to the variables initialized by `waminit.jsp` (described previously), `init.jsp` defines the following ones:

- **processServiceSession** - This is an object of type `fuego.papi.ProcessServiceSession`, from PAPI. It represents a session that allows users access every process operation such as get instances, run tasks, attachment operations, participants operations, etc..
- **userOptions** - This Work Portal specific object, of type `fuego.portal.UserOptions`, helps to deal with the current user's information and personal preferences.

The code in `init.jsp` also redefines some of the variables already created by `waminit.jsp`, taking into account that now a user is logged in. For example, the `locale` variable will have the user-specific `Locale` assigned.

## An example

The Work Portal provides the user with a menu of all the available views (views tree) on the left side of the screen.

The default behavior of the Work Portal is that when the user clicks on a particular instance view, the list of instances in that view are shown on the right side of the screen. So are the attachments when the clicked view is an attachments view and the applications when the clicked view is an applications view.

In the case of instances views, all the instances are shown as rows in a table where the columns are the instances' variables selected by the user for the presentation of that particular view. This simple example will show how to modify the way the instances are listed in the table, so that the instances with major priority are shown emphasized with a different background and font.

As the layouts in **appendix A** suggest, the instances information list is coded in the **`detailContent.jsp`** file. Since this is not a top-level page (instead, it is included by many other JSP pages), there's no key in the `JSPBundle` for this particular page. Therefore, the same filename will be used.

## Hands on

First of all, a backup copy of the original `detailContent.jsp` should be created.

Inspecting the **`detailContent.jsp`** file shows that after getting the request attributes with the information needed, the columns of the table are drawn.

After including the **`actionToolbar.jsp`**, the titles of the variables included in the presentation of the view are shown as columns. After that, a row is inserted in the table for each instance the view has, and the information of every variable of the instance is a cell of that row in the table.

Depending on the kind of the variable, the cell is drawn in different ways, the participant cell, for example has an icon beside the participant name to allow users select/unselect the instance from there. But all the cells are displayed using a class defined in a styles file.

Adding a new class for the instances with high priority, the only change in the jsp will be assign this new class for those instances whose priority is greater than NORMAL. So, the first thing we have to do is to add a new class in the style file we have set in the **`portal.properties`** file in order to emphasize the instances with high priorities. Remember that if you are using one of the style files provided by Fuego, you should customize the styles file as suggested in the section CSS on page.

```
...
/* Style of the web work portal's table cells */
.tablecell f font-family: Verdana, Arial,
    Helvetica, sans-serif;
font-size: 11px; color: #000000;
background-color: #DEE7F7g
/* Style of the web work portal's emphasized table cells */
.emphablecell f font-family: Verdana, Arial,
    Helvetica, sans-serif;
font-size: 11px; color: #000000; background-color: #FFCCCCg
...
```

Once the new class has been added to the styles file, the next step is modify the page. In the **detailContent.jsp** file, the class used to draw the cells is "tablecell". Our modification is made up of a change in the class use for the table cells in those cases where the priority is higher than normal. So, in the for loop that traverses the instances list we have to add this portion of code :

```
<%
  for (int i = 0; i < processInstances.length; i++) {
    fuego.papi.InstanceInfo ii = processInstances[i];
    if (grabActivity != null) {
      instanceActions.setProcessInstance(ii,
        ii.getActivity(),
        processServiceSession,
        grabActivity);
    } else {
      instanceActions.setProcessInstance(ii,
        ii.getActivity(),
        processServiceSession);
    }
    String cellclass = "tablecell";
    if (ii.getPriority() > FilterAttribute.NORMAL_PRIORITY)
    {
      cellclass = "emphtablecell";
    }
  }
%>
```

The previous lines assign the properly class to the variable cellclass. Then the variable is used to assign the class to the cell as follows :

```
...
</td>
<%
} else
  if (columns[j].getId().equals(VarDefinition.STATUS_ID)) {
%>
<td height="9" class="<%=cellclass%>">
  <table width="100%" border="0"
    cellspacing="0"
    cellpadding="0">
    <tr>
      <td height="9" width="100%"
        class="<%=cellclass%>">
```



```

        <%=formattedValue.length()==0)?
        "&nbsp;" :
        formattedValue%>
    </td>
    <td height="9"
        valign="middle"
        align="right"
        class="<%=cellclass%>">
<%
    if(ii.isGrabbed()) f %>
        <a href=
            ...
        </a>
<%
} else if(ii.isException()) {
%>
        <a href=
            ...
        </a>
<%
} else if(ii.isSuspended()) {
%>
        <a href=
            ...
        </a>
<%
} else if(instanceActions.isCompletable()) {
%>
        <a href=
            ...
        </a>
<%
    } %>
</td>
</tr>
</table>
</td>
<% } else { %>
    <td height="9" class="<%=cellclass%>">
        <%=formattedValue == null ||
        formattedValue.length()==0)? "&nbsp;" :
        formattedValue%>
    </td>
<%
    } %>
<%} %>
...

```

This is just an example of how, with even simple modifications, the JSPs can be altered to suit different needs. The more the programmer analyses the JSPs, the more insight he will get to

change them at will.

## WAPI

WAPI is the Workportal API. It provides a clean interface for accessing the Work Portal servlets' functionality.

### Note



In order to use WAPI, knowledge on Java, JSP, HTML and JavaScript is needed.

Currently, WAPI consists of one Java class: `URLForAction`. Each method of this class represents a service that will be provided by one of the Work Portal's servlets. Refer to the WAPI javadoc documentation for a detailed description of every method.

The way it works is simple: every WAPI method receives a `javax.servlet.http.HttpServletRequest` which must contain the required parameters for that call, and returns a URL (in String form) which will be a call to the correct servlet with the necessary parameters encoded.

That URL can be used for HTML form actions, and href links that give the user a graphical interface to fire that particular call.

## Example

Here is how a Global activity is called from the Work Portal:

The **`applicationView.jsp`** is displayed when the user selects an Applications View from the menu. At this point, the list of Global activities (also called applications) are displayed to the user. When the user clicks on a particular Global activity link, a servlet call must be made in order to actually execute that activity.

The URL to fire that servlet call is constructed with `URLForAction.runGlobalApplication(request)`. Here is an excerpt of **`applicationsView.jsp`**:

```
...  
<form method="post"  
action="<%= URLForAction.runGlobalApplication(request)%>"  
name="applicationsForm">  
<input type="hidden" name="activityId">  
</form>  
...
```

So, there is an HTML Form that will be posted to the URL created by the call to *URLForAction.runGlobalApplication(request)*. Note that the required parameter for *runGlobalApplication()* ( *activityId* ) will be sent when the form is submitted. Also note that the value for the *activityId* field is not set yet.

Here is the *applicationsView.jsp* code that creates the link for a Global activity:

```
...  
<a class="tablecell"  
href="javascript:runglobal('<%=activity.getId()%>')">  
<%=activity.getLabel(locale)%></a>  
...
```

The link calls the javascript function *runglobal(aid)*, which submits the form after setting the value of the *activityId* field:

```
...  
function runglobal(aid) {  
document.applicationsForm.activityId.value = aid;  
document.applicationsForm.submit();  
}  
...
```

## Context

After a call to a Work Portal servlet, a JSP page will be executed and

the resulting screen shown to the user. Which of the Work Portal's JSP will that be? That decision is made by the Controller servlet, as described in section *How it works* in chapter 2.

Normally, a particular servlet action that is requested (a method of `URLForAction`) will be followed by the same JSP. For example, after a call to `URLForAction.enterNote()`, the `newNote.jsp` page will commonly be loaded (as could be expected), and after `URLForAction.viewActivityDocumentation()` the execution of `activityDoc.jsp` will follow.

However, in some cases, the next page to be displayed is not always the same. It depends on some context information. For example, the user can select an instance from two different screens: from the list of instances screen (`instancesView.jsp`) or from the screen that shows a particular instance's information (`instanceDetail.jsp`). So, when calling the action that (un)selects an instance, the next JSP to be executed will depend on the context: it should return to the screen where the user was when (un)selecting the instance.

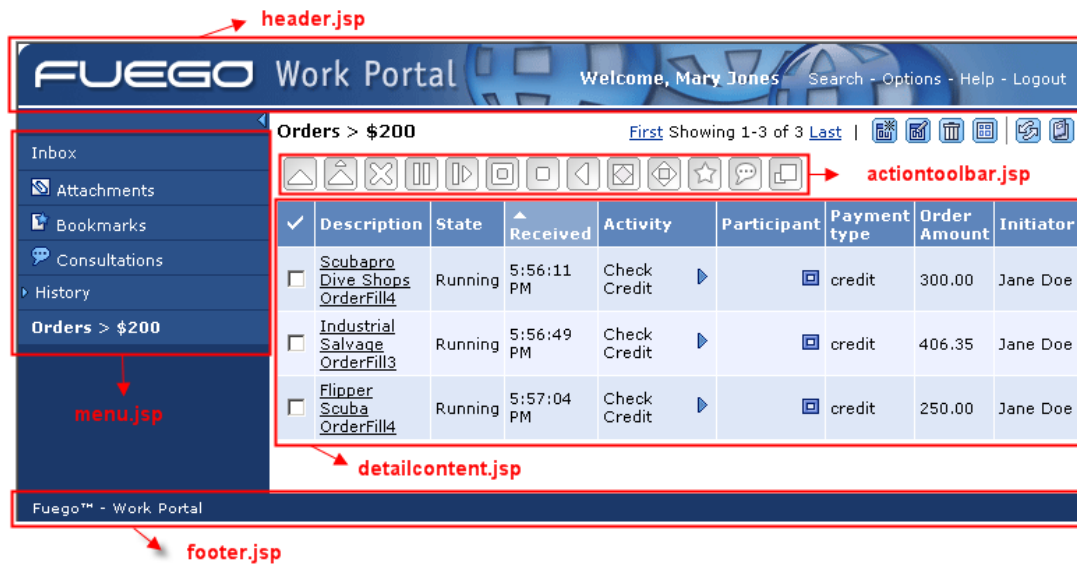
There might be some special cases in which the expected JSP page is not the one loaded right after a particular action. For example, if the user is working in the Work Portal and then remains idle for a while, the session would expire. Then, when the user tries to resume working, the login screen will be displayed (`login.jsp`). Note also that after the login, the Work Portal will take the user to the screen where he/she was when the session expired (and not to the default `welcome.jsp` that normally comes after a login).

When the Work Portal finds some error while processing a request, the `theservererror.jsp` page is executed. In the case of an unexpected Runtime Error (one that the Work Portal is not able to handle), the `error.jsp` page is executed instead. Actually, the `error.jsp` page is defined as the generic error-page in the Work Portal deployment descriptor file `web.xml`).

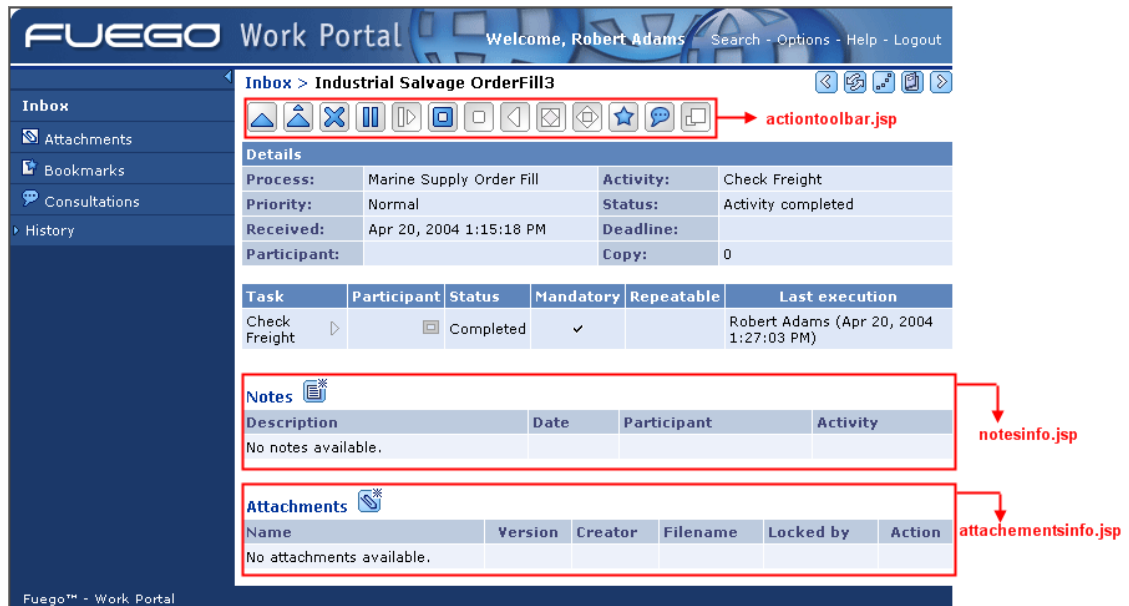
## Main JSPs' layouts

As introduced previously, most of the top-level JSP pages in the Work Portal are actually composite pages, meaning that they include other pages to compose the final layout. Following is a visual description of how the main Work Portal pages are composed of other JSP files.

## instancesView.jsp



## instanceDetail.jsp



search.jsp

## JSP Descriptions

This section describes each of the main JSP pages that conform the Work Portal. For each page, the KEY in the JSPBundle, the default file name, the parameters used, and a screenshot are exposed.

For each parameter listed, its Java-type is described as well as the key needed to access this parameter.

The following convention is used to specify the parameter keys:

- If the key is all uppercase, it means it is an attribute of the ApplicationConstants class. Example: REQUESTED PROCESS is ApplicationConstants.REQUESTED PROCESS
- If the key is between double quotes, it is a java String. Example: **title**
- If the key includes the class name, it is a static attribute of that class. Example: **FileTypeAssociations.ATTRIBUTE MODEL**

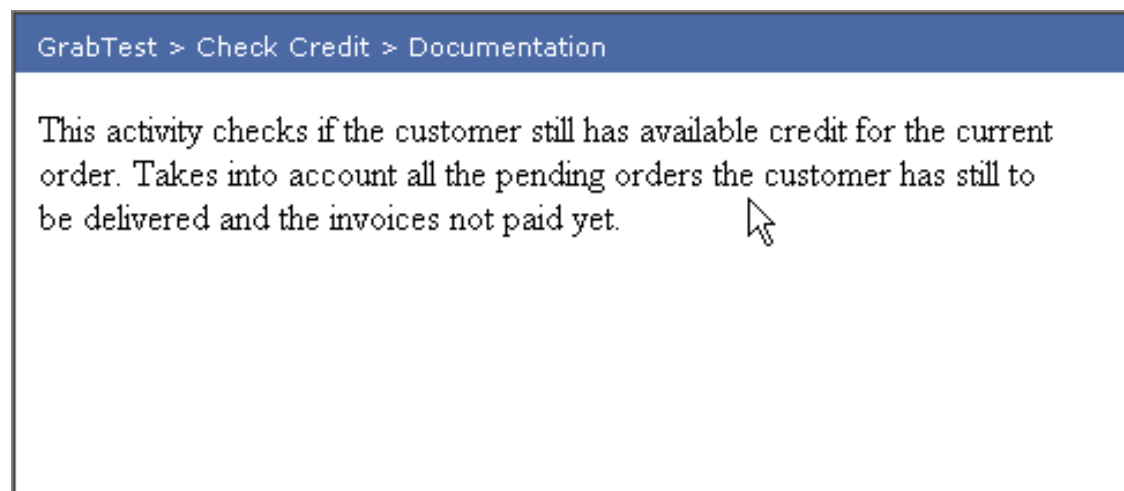
As an example, given the following parameters:

- `fuego.papi.Process` (REQUESTED PROCESS)
- `java.lang.Exception` (*authexception*)
- `java.util.Hashtable` (FileTypeAssociations.ATTRIBUTE MODEL)

here is the Java code that could be used in the JSP to get them:

```
fuego.papi.Process process =  
    (fuego.papi.Process) request.getAttribute  
        (ApplicationConstants.REQUESTED_PROCESS);  
Exception exception =  
    (Exception) request.getAttribute("authexception");  
java.util.Hashtable assocHash = (java.util.Hashtable)  
    request.getAttribute(FileTypeAssociations.ATTRIBUTE_MODEL);
```

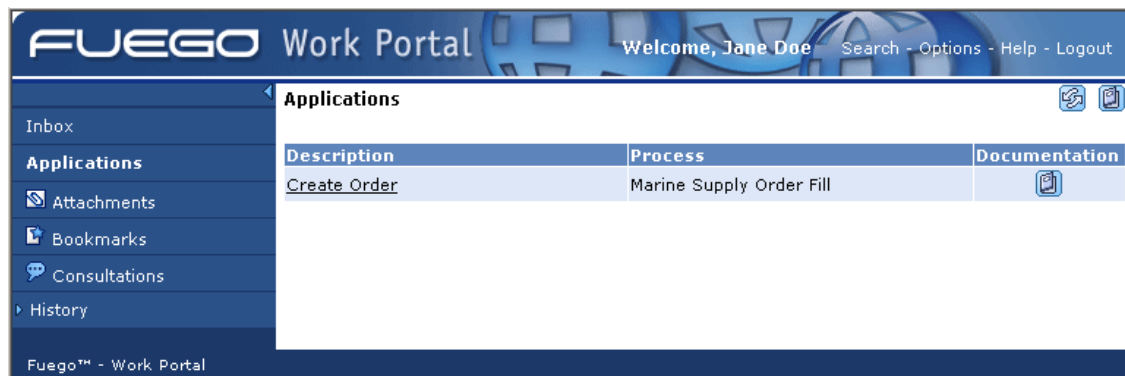
## activityDoc.jsp



- KEY: ACTIVITY\_DOCUMENTATION

- Default filename: /jsp/activityDoc.jsp
- Parameters:
  - java.lang.String (PROCESS\_NAME)
  - java.lang.String (ACTIVITY\_NAME)
  - java.lang.String (ACTIVITY\_DOC\_PATH)

## applicationsView.jsp



- KEY: APPLICATIONS\_VIEW
- Default filename: /jsp/applicationsView.jsp
- Parameters:
  - fuego.papi.ApplicationsView (REQUESTED\_VIEW)
  - fuego.papi.Activity[]  
(REQUESTED\_APPLICATIONS\_FOR\_CURRENT\_VIEW)
  - java.lang.String[]  
(REQUESTED\_ACTIVE\_PROCESSES\_FOR\_CURRENT\_VIEW)






## attachmentsView.jsp



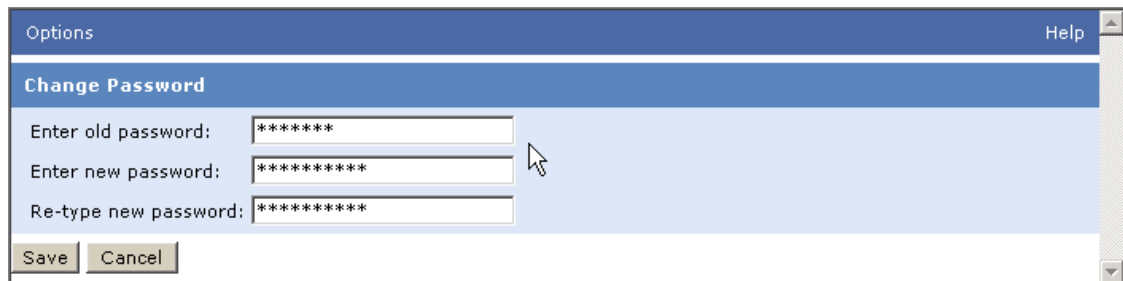
- KEY: ATTACHMENTS\_VIEW
- Default filename: /jsp/attachmentsView.jsp
- Parameters:
  - fuego.papi.AttachmentsView (REQUESTED\_VIEW)

## auditTrail.jsp

Audit trail <span>Help</span>				
<b>Marine Supply Order Fill &gt; Check Credit &gt; Industrial Salvage OrderFill4</b>				
Activity	Event	Responsible	Date	Copy
<input type="checkbox"/> <u>create</u>	 Completed		Dec 22, 2003 5:43:33 PM	0
	Creation	Jane Doe	Dec 22, 2003 5:43:33 PM	0
<input type="checkbox"/> <u>ReviewOrder</u>	 Completed		Dec 22, 2003 5:44:02 PM	0
	Enter	Account Manager	Dec 22, 2003 5:44:02 PM	0
	Item Execution	✓ John Smith	Dec 22, 2003 5:44:20 PM	0
	Exit	John Smith	Dec 22, 2003 5:44:24 PM	0
<input type="checkbox"/> <u>Check Credit</u>	 Processing		Dec 22, 2003 5:44:24 PM	0
	Enter	John Smith	Dec 22, 2003 5:44:24 PM	0
	Item Execution	✓ Mary Jones	Dec 22, 2003 5:45:42 PM	0
<input type="button" value="Close"/>				
Fuego™ - Work Portal				

- KEY: AUDIT\_TRAIL
- Default filename: /jsp/auditTrail.jsp
- Parameters:
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
  - fuego.papi.utils.AuditTrail (EVENTS)
  - java.lang.String (NODE\_ID)

## changePassword.jsp



The screenshot shows a web-based dialog box titled "Options" with a "Help" button in the top right corner. The main heading inside the dialog is "Change Password". Below this heading are three text input fields. The first field is labeled "Enter old password:" and contains seven asterisks. The second field is labeled "Enter new password:" and contains eight asterisks. The third field is labeled "Re-type new password:" and also contains eight asterisks. At the bottom left of the dialog are two buttons: "Save" and "Cancel". A mouse cursor is visible over the second input field.

- KEY: CHANGE\_PASSWORD\_OPTION
- Default filename: /jsp/changePassword.jsp
- Parameters:
  - java.lang.String (REQUESTED\_ERROR)

## checkinAttachment.jsp

The screenshot shows the Fuego Work Portal interface. At the top, there is a blue header bar with the 'FUEGO Work Portal' logo on the left, and 'Welcome, John Smith' followed by links 'Search - Options - Help - Logout' on the right. Below the header, a breadcrumb trail reads 'Industrial Salvage OrderFill1 > Attachment Check In'. The main content area features a 'REMARKS:' label next to a text input field containing the text 'Please that I have included Industrial Salvage Order #2 in this list, John'. Below the input field are 'Ok' and 'Cancel' buttons. At the bottom of the page, a dark blue footer bar displays 'Fuego™ - Work Portal'.

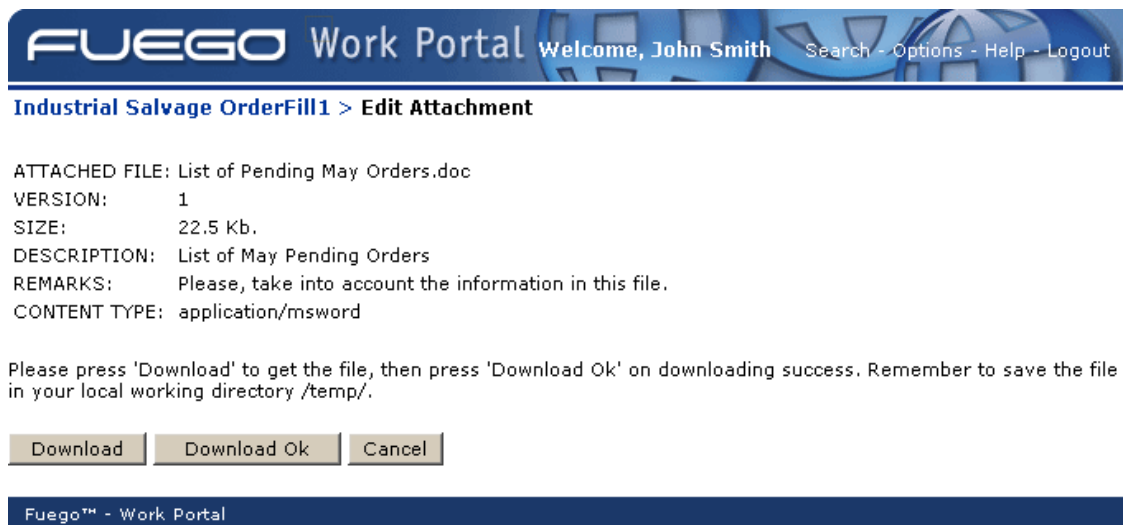
- KEY: CHECK\_IN\_INSTANCE\_ATTACHMENT
- Default filename: /jsp/checkinAttachment.jsp
- Parameters:
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
  - fuego.papi.Attachment (V\_ATTACHMENT)
  - java.lang.String (REMARKS)
  - java.util.Hashtable (ATTACHMENT\_SERVLET\_ERRORS)

## checkinWorkingAttachment.jsp

This screenshot is identical to the one above, showing the Fuego Work Portal interface with the 'Attachment Check In' dialog box. It includes the same header, breadcrumb trail, remarks input field, buttons, and footer.

- KEY: CHECK\_IN\_WORKING\_ATTACHMENT
- Default filename: /jsp/checkinWorkingAttachment.jsp
- Parameters:
  - fuego.papi.Attachment (V\_ATTACHMENT)
  - java.lang.String (REMARKS)
  - java.util.Hashtable (ATTACHMENT\_SERVLET\_ERRORS)

## editAttachment.jsp



- KEY: EDIT\_INSTANCE\_ATTACHMENT
- Default filename: /jsp/editAttachment.jsp
- Parameters:
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)

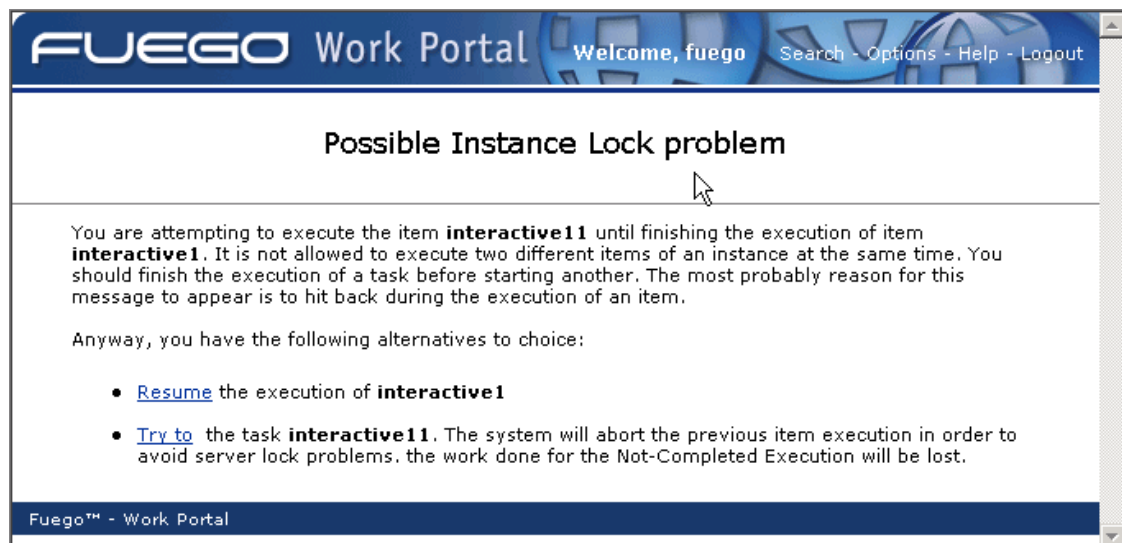
- fuego.papi.Attachment (V\_ATTACHMENT)

## editOptions.jsp

Options		Help
<b>User Information</b>		
Full Name:	fuego Fuego	
Login Name:	fuego	
E-mail:		
PASSWORD:	<a href="#">Change Password</a>	
<b>Browser settings</b>		
Enable Flash version menu:	<input type="checkbox"/>	
Enable DHTML support:	<input checked="" type="checkbox"/>	
<b>Settings</b>		
Sort instances by:	Received	
Instances order:	Ascending	
Show hidden views:	<input type="checkbox"/>	
Follow the Instance:	<input type="checkbox"/>	
Notify me by e-mail when new instances arrive:	<input type="checkbox"/>	
Keep instance view:	<input type="checkbox"/>	
Enable applet for attachment management:	<input type="checkbox"/>	
Enable remote scripting for FuegoObject presentations:	<input checked="" type="checkbox"/>	
Show applications:	In a folder	
User Working Directory:	/temp/	
	(Including last path separator, ie.: 'c:\temp\').	
Maximum number of searches in history:	10	
<b>Display options</b>		
Number of instances:	10	
Language:	English	
Country:	United States	
TimeZone:	GMT-03:00	
<input type="button" value="Save"/> <input type="button" value="Close"/>		

- KEY: EDIT\_OPTIONS
- Default filename: /jsp/editOptions.jsp
- Parameters:
  - java.lang.String[] (FT\_TIME\_ZONES)
  - boolean (ATTR\_PART\_AUTOLOGIN)
  - boolean (IS\_CHANGE\_PASSWORD\_SUPPORTED)
  - fuego.papi.Presentation.Column[] (DEFAULT\_COLUMNS)

## executionLock.jsp



- KEY: EXECUTION\_LOCK
- Default filename: /jsp/executionLock.jsp
- Parameters:

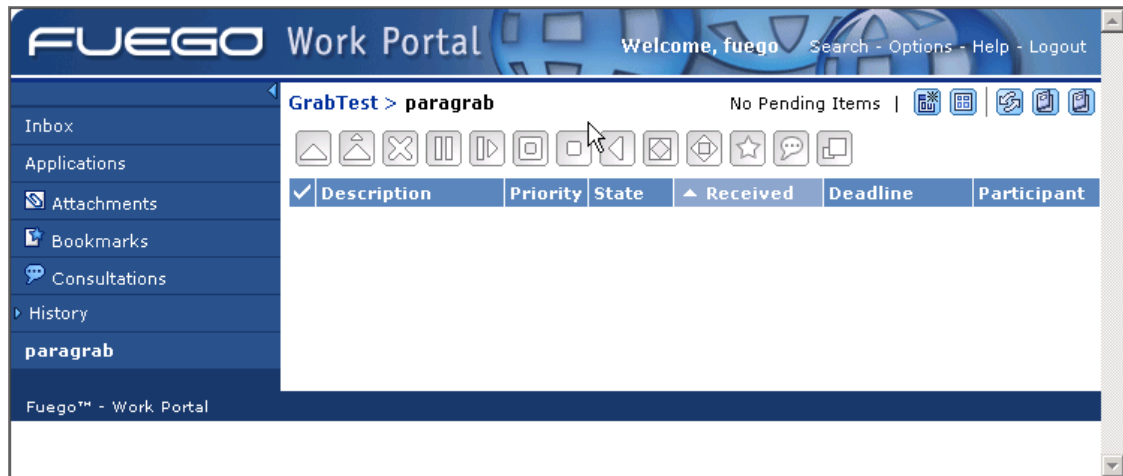
- java.lang.String (OLD\_ITEM\_KEY)
- java.lang.String (OLD\_ITEM\_DESCR)
- java.lang.String (NEW\_ITEM\_KEY)
- java.lang.String (NEW\_ITEM\_DESCR)

## fileTypeAssociations.jsp

The screenshot shows a Java Swing dialog box titled 'Options' with a 'Help' button in the top right corner. The main title bar is blue and contains the text 'File Type / Application Association'. Below this, the dialog is divided into three columns: 'Extension', 'Application Path', and 'Parameters'. In the 'Extension' column, there is a 'New' button above a text field containing 'doc'. In the 'Application Path' column, there is a text field containing 'C:\Program Files\Micro' and a 'Browse' button to its right. In the 'Parameters' column, there is an empty text field and a 'Delete' button to its right. At the bottom left of the dialog, there are 'Save' and 'Cancel' buttons.

- KEY: FILE\_TYPE\_ASSOCIATION\_OPTION
- Default filename: /jsp/fileTypeAssociations.jsp
- Parameters:
  - java.util.Hashtable (FileTypeAssociations.ATTRIBUTE\_MODEL)

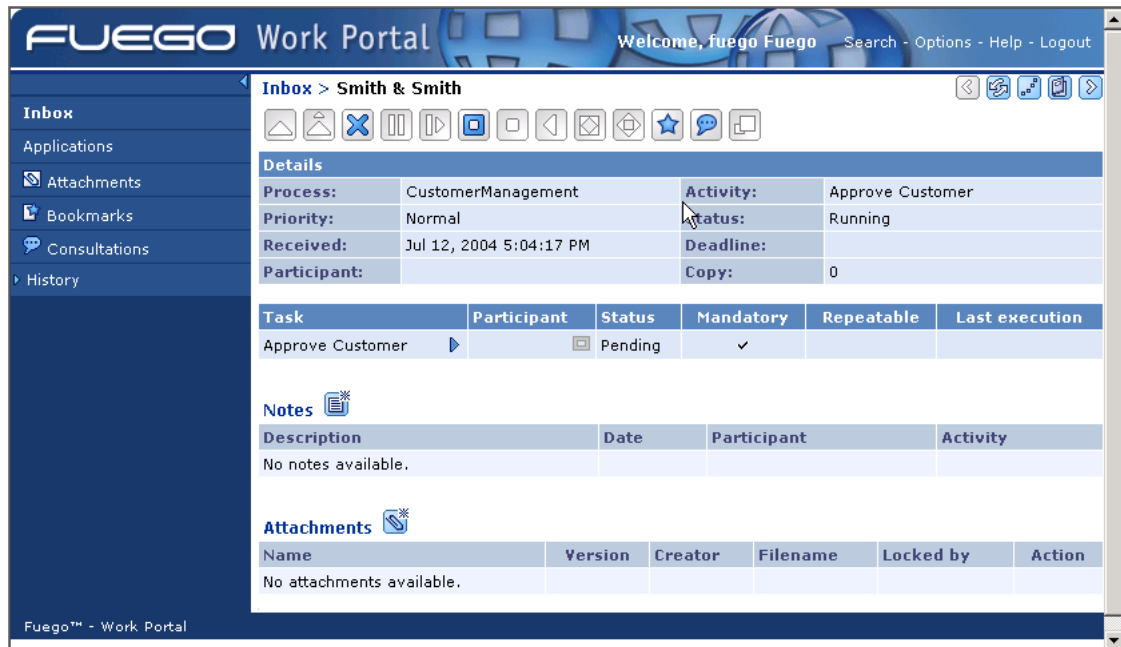
## folder.jsp



- KEY: FOLDER
- Default filename: /jsp/folder.jsp
- Parameters:
  - fuego.papi.View (REQUESTED\_VIEW)

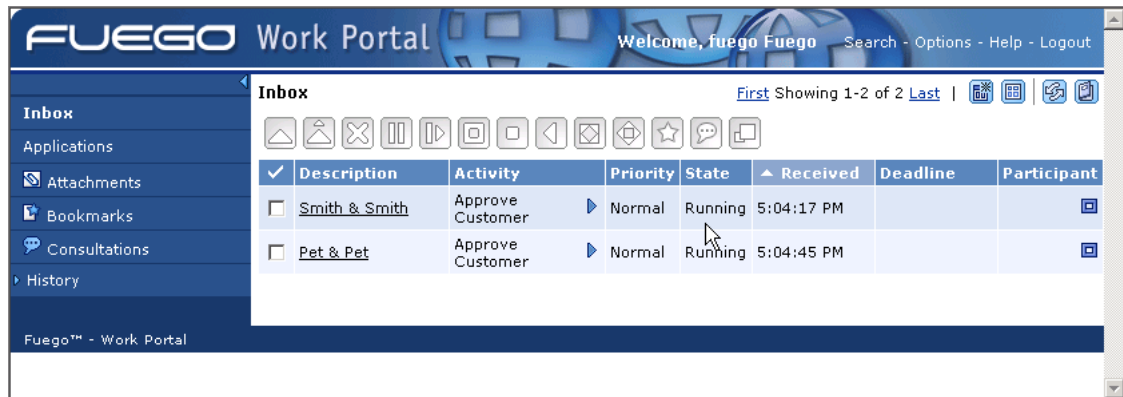
## instanceDetail.jsp





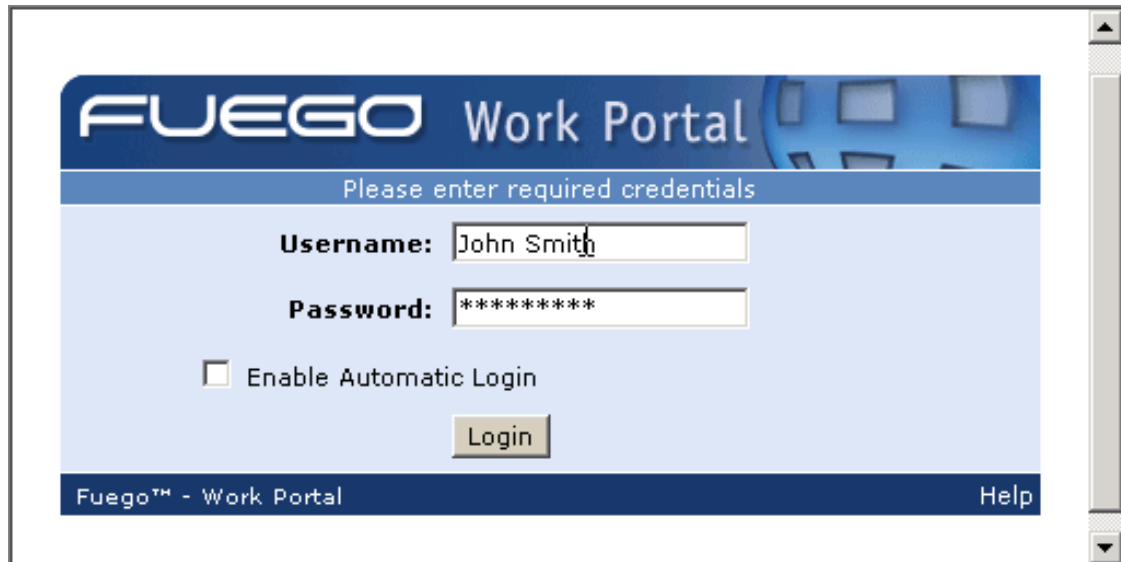
- KEY: INSTANCE\_DETAIL
- Default filename: /jsp/instanceDetail.jsp
- Parameters:
  - List (PROCESSES\_WITH\_SHOW\_IMAGE\_ACTIVITY)
  - fuego.papi.Process (REQUESTED\_PROCESS)
  - fuego.papi.Activity (REQUESTED\_ACTIVITY)
  - fuego.papi.InstanceInfo[] (REQUESTED\_INSTANCES\_COMPLETE\_SET)
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
  - fuego.papi.View (REQUESTED\_VIEW)

## instancesView.jsp



- KEY: INSTANCES\_VIEW
- Default filename: /jsp/instancesView.jsp
- Parameters:
  - List (PROCESSES\_WITH\_SHOW\_IMAGE\_ACTIVITY)
  - fuego.papi.Process (REQUESTED\_PROCESS)
  - fuego.papi.Activity (REQUESTED\_ACTIVITY)
  - fuego.papi.InstanceInfo[] (REQUESTED\_INSTANCES)
  - fuego.papi.View (REQUESTED\_VIEW)
  - boolean (SHOW\_PARAMETERS)
  - fuego.papi.Presentation (VIEW\_PRESENTATION)
  - java.lang.Integer (REQUESTED\_INSTANCE\_FROM)
  - java.lang.Integer (REQUESTED\_INSTANCE\_TO)
  - java.lang.Integer (REQUESTED\_INSTANCES\_COUNT)
  - java.lang.String (SORT\_BY)
  - java.lang.String (SORT\_ORDER)

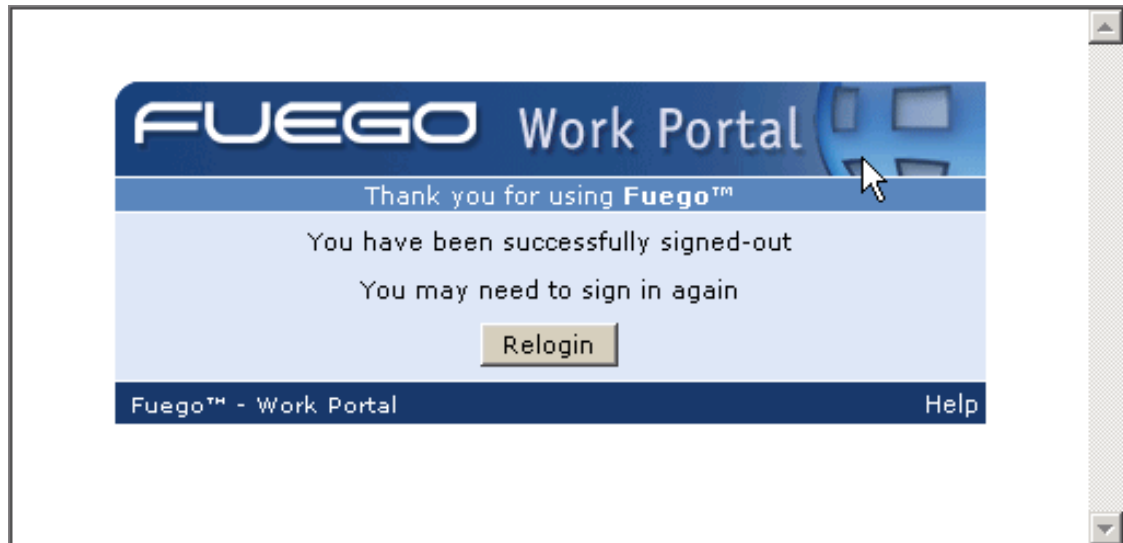
## login.jsp



The screenshot shows a web browser window displaying the Fuego Work Portal login page. The page has a blue header with the 'FUEGO Work Portal' logo. Below the header, a blue bar contains the text 'Please enter required credentials'. The main content area is light blue and contains two input fields: 'Username:' with the text 'John Smith' and 'Password:' with masked characters '\*\*\*\*\*'. Below these fields is a checkbox labeled 'Enable Automatic Login' and a 'Login' button. At the bottom of the page, a dark blue footer bar contains the text 'Fuego™ - Work Portal' on the left and a 'Help' link on the right.

- KEY: LOGIN
- Default filename: /jsp/login.jsp
- Parameters:
  - java.lang.String (USERNAME)
  - java.lang.String (ORIGINAL\_ABSOLUTE\_URL)
  - java.lang.String (POST\_STATE)
  - java.lang.String (ATTR\_LOGIN\_ERROR\_MSG)
  - java.lang.String (ATTR\_LOGIN\_ERROR\_DETAIL)
  - java.lang.String (ATTR\_LOGIN\_ERROR\_TECH\_DETAIL)
  - java.lang.String (ATTR\_LOGIN\_ERROR\_HAS\_DETAIL)
  - java.lang.String (ATTR\_LOGIN\_ERROR\_HAS\_TECH\_DETAIL)
  - boolean (DEVELOPMENT\_ENVIROMENT)

## logout.jsp



- KEY: LOGOUT
- Default filename: /jsp/logout.jsp
- Parameters: NONE

## newAttachment.jsp

**FUEGO Work Portal** Welcome, John Smith Search - Options - Help - Logout

**Industrial Salvage OrderFill1 > New Attachment**

ATTACH FILE: C:\Documents And Settings\My Documents\ Browse... ☒ Advanced options

CONTENT TYPE: ☐ auto-detect ☐ select from list: plain text (text/plain) ☒ enter manually: /application/msword

YOU CAN BROWSE TO SELECT THE FILE.

DESCRIPTION: List of Pending May Orders

REMARKS: Please, take into account the information in this file.

Ok Cancel

Fuego™ - Work Portal

- KEY: NEW\_INSTANCE\_ATTACHMENT
- Default filename: /jsp/newAttachment.jsp
- Parameters:
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
  - java.lang.String (TITLE)
  - java.lang.String (REMARKS)
  - java.util.Hashtable (ATTACHMENT\_SERVLET\_ERRORS)

## newNote.jsp

The screenshot shows the FUEGO Work Portal interface. The top header bar is blue with the text 'FUEGO Work Portal' and 'Welcome, John Smith'. To the right of the header are links: 'Search - Options - Help - Logout'. Below the header, the main content area has a blue bar with the text 'Scubapro Dive Shops OrderFill2 > New Note'. In the center, there is a text input field containing the text: 'Mary, please doublecheck this customer's credit limit to make sure it has not been exceeded. Thanks.'. Below the input field are two buttons: 'Ok' and 'Cancel'. At the bottom of the page, there is a blue footer bar with the text 'Fuego™ - Work Portal'.

- KEY: NEW\_NOTE
- Default filename: /jsp/newNote.jsp
- Parameters:
  - fuego.papi.IstanceInfo (REQUESTED\_INSTANCE)
  - fuego.papi.Activity (REQUESTED\_ACTIVITY)
  - java.lang.String (ACTIVITY\_ID)

## participantList.jsp

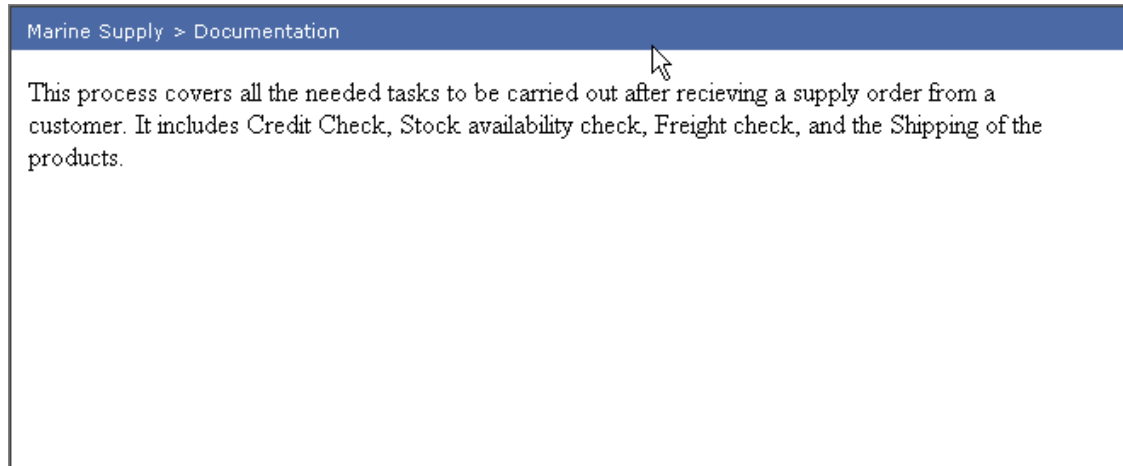


A screenshot of a web browser window showing a table with three columns: Name, UID, and E-mail. The table contains five rows of data. The first four rows have blue hyperlinks in the Name column. The E-mail column is empty for the first four rows and contains the text 'null' for the last row.

Name	UID	E-mail
<a href="#">Ann Parker</a>	Ann Parker	
<a href="#">Fuego Fuego</a>	fuego	
<a href="#">John Smith</a>	John Smith	
<a href="#">Sam Baldwin</a>	Sam Baldwin	
<a href="#">sa</a>	sa	null

- KEY: PARTICIPANT\_LIST
- Default filename: /jsp/participantList.jsp
- Parameters:
  - java.lang.String (CN)
  - java.lang.String (HIDDEN\_CN)
  - java.lang.String (UID)
  - fuego.papi.Participant[]  
(REQUESTED\_PARTICIPANT\_ITERATOR)
  - java.lang.String (PARTICIPANT\_SORT\_BY)
  - java.lang.String (PARTICIPANT\_SORT\_ORDER)
  - java.lang.String (MATCHING\_NAME)

## processDoc.jsp



- KEY: PROCESS\_DOCUMENTATION
- Default filename:/jsp/processDoc.jsp
- Parameters:
  - java.lang.String (PROCESS\_NAME)
  - java.lang.String (PROCESS\_DOC\_PATH)

## search.jsp



Search OptionsHelp

Search Clear Close

Processes

☐ BAMDashBoard
☐ CommercialInformation
☒ CustomerManagement
☐ OrderFulfillmentDashBoard
☐ Stock Administration

Filter Options

Get Instances Assigned To: All

Case Sensitive Matching: ☐

Include Instances

☒ In process
☐ Completed
☐ Aborted

Conditions

Match all of the following ☐

Add Condition: Activity +

2:09:42 PMShowing 1-2 of 2

Description	Activity	Priority	State	Received	Deadline	Participant
<u>Pet &amp; Pet</u>	Assign Credit Limit To Customer	Normal	Running	Jul 12 05:21:12 PM		
<u>Smith &amp; Smith</u>	Approve Customer	Normal	Running	2:09:27 PM		

Search Clear Close

Filter Description

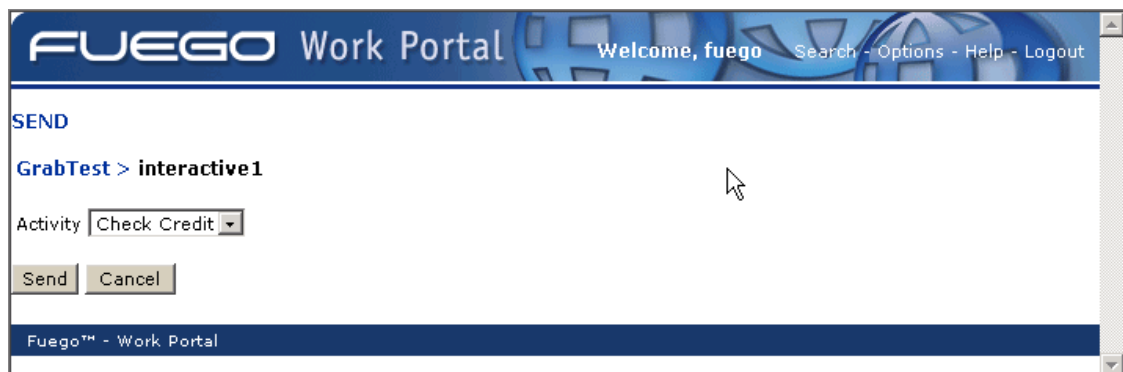
Fuego™ - Work Portal

- KEY: SEARCH
- Default filename:/jsp/search.jsp
- Parameters:
  - java.lang.Boolean (SEARCH\_INCLUDE\_ABORTED\_INSTANCES)
  - java.lang.Boolean (SEARCH\_INCLUDE\_COMPLETED\_INSTANCES)

- java.lang.Boolean  
(SEARCH\_INCLUDE\_IN\_PROCESS\_INSTANCES)
- java.lang.String[] (REQUESTED\_SORTED\_VIEWID\_ARRAY)
- java.lang.String (SHOW\_RESULTS)
- java.lang.String (SHOW\_PRESENTATION)
- java.lang.String (NEW\_PRESENTATION\_ID)
- java.lang.String (VIEW\_ID\_CHANGED)
- java.lang.String (OLD\_VIEW\_ID)
- java.lang.ArrayList (REQUESTED ACTIVITY LIST)
- java.lang.String (IS\_HIDDEN)
- fuego.papi.View (REQUESTED VIEW)
- java.lang.String (SEARCH\_VIEW\_ID)
- java.lang.String (PARENT\_VIEW\_ID)
- fuego.papi.Filter (REQUESTED\_FILTER)
- java.lang.String[] (SELECTED\_PROCESSES)
- java.lang.Integer (REQUESTED\_INSTANCE\_FROM)
- java.lang.Integer (REQUESTED\_INSTANCE\_TO)
- java.lang.Integer (REQUESTED\_INSTANCES\_COUNT)
- java.lang.String (SORT\_BY)
- java.lang.String (SORT\_ORDER)
- fuego.papi.Presentation (REQUESTED\_PRESENTATION)
- fuego.papi.Process [] (REQUESTED\_PROCESS\_ITERATOR)

- `fuego.papi.Presentation []` (AVAILABLE\_PRESENTATIONS)
- `fuego.papi.Presentation.Column []` (AVAILABLE\_COLUMNS)
- `fuego.papi.Presentation.Column []` (SELECTED\_COLUMNS)
- `java.util.ArrayList` (SELECTED\_COLUMNS\_IDS)
- `java.lang.String` (PROCESS\_IS\_FIXED)
- `fuego.papi.Process` (REQUESTED\_PROCESS)
- `fuego.papi.VarDefinition[]` (PROCESS\_VARS)

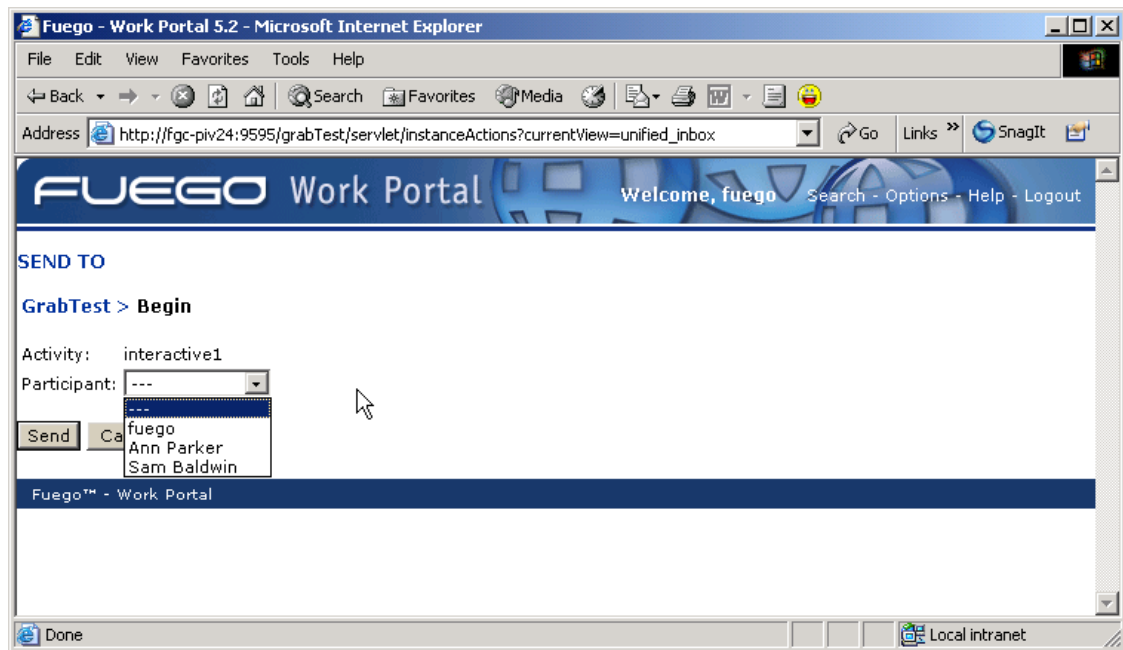
## send.jsp



- KEY: SEND
- Default filename: /jsp/send.jsp
- Parameters:
  - `fuego.papi.InstanceInfo` (REQUESTED\_INSTANCE)
  - `fuego.papi.Activity` (REQUESTED\_ACTIVITY)

- java.util.List (TARGET\_ACTIVITIES)
- fuego.papi.Process (REQUESTED\_PROCESS)
- java.lang.String (PROCESS\_NAME)
- java.lang.String (ACTIVITY\_ID)
- java.lang.String (INSTANCE\_STAMP\_ID)
- java.lang.String (NEXT\_INSTANCE\_STAMP\_ID)

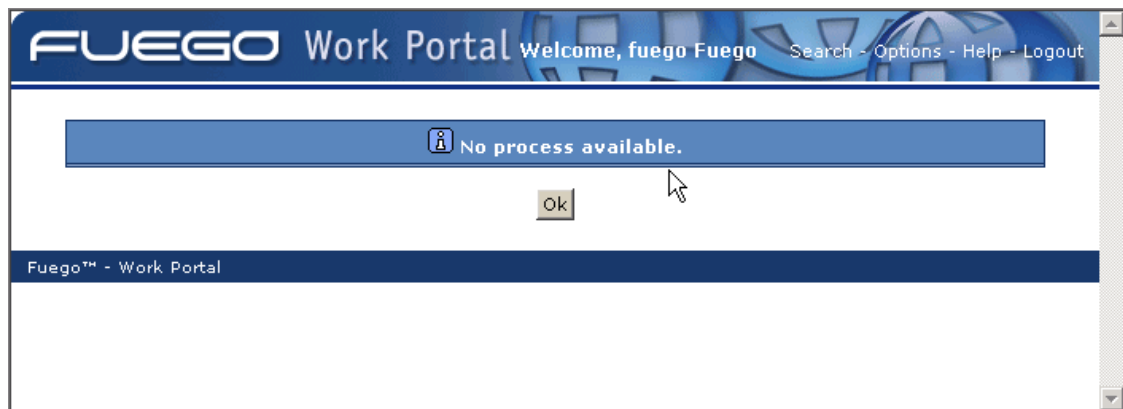
## sendTo.jsp



- KEY: SEND\_TO
- Default filename: /jsp/sendTo.jsp
- Parameters:

- fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
- fuego.papi.Activity (REQUESTED\_ACTIVITY)
- fuego.papi.ParticipantsForActivities (TARGET\_PARTICIPANTS)
- fuego.papi.Process (CURRENT\_PROCESS)
- java.lang.String (PROCESS\_NAME)
- java.lang.String (ACTIVITY\_ID)
- java.lang.String (INSTANCE\_STAMP\_ID)
- java.lang.String (NEXT\_INSTANCE\_STAMP\_ID)

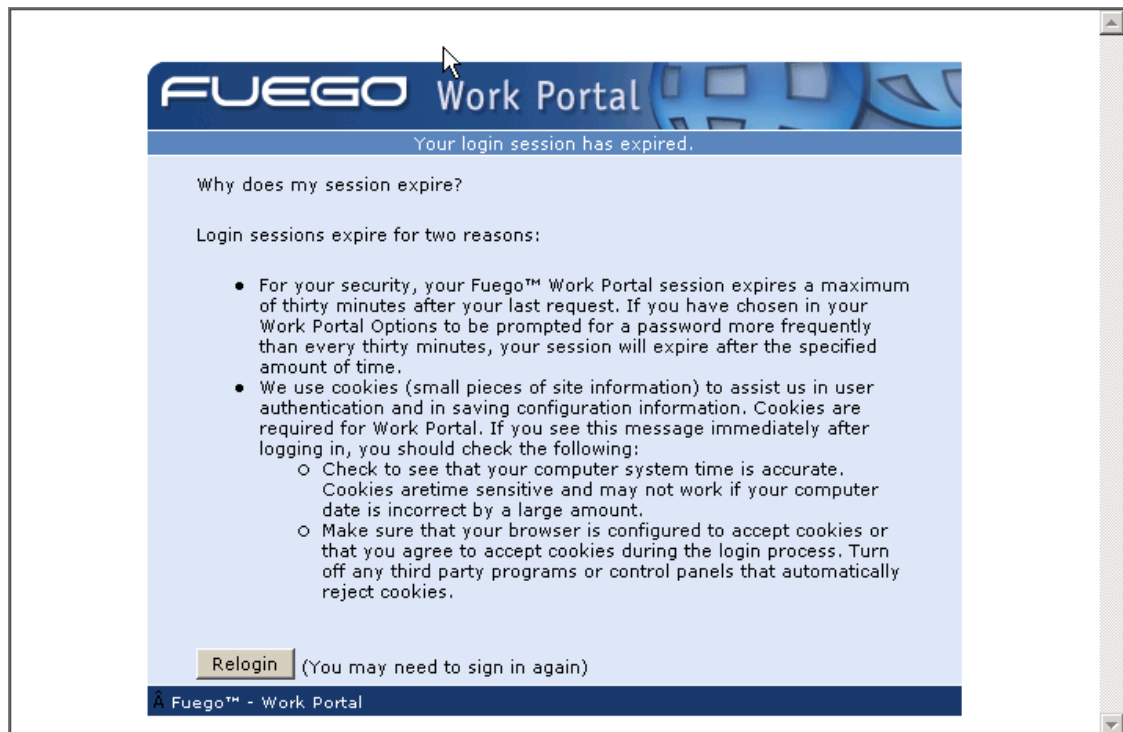
## servererror.jsp



- KEY: SERVER\_ERROR
- Default filename: /jsp/servererror.jsp
- Parameters:
  - java.lang.Integer (TYPE)

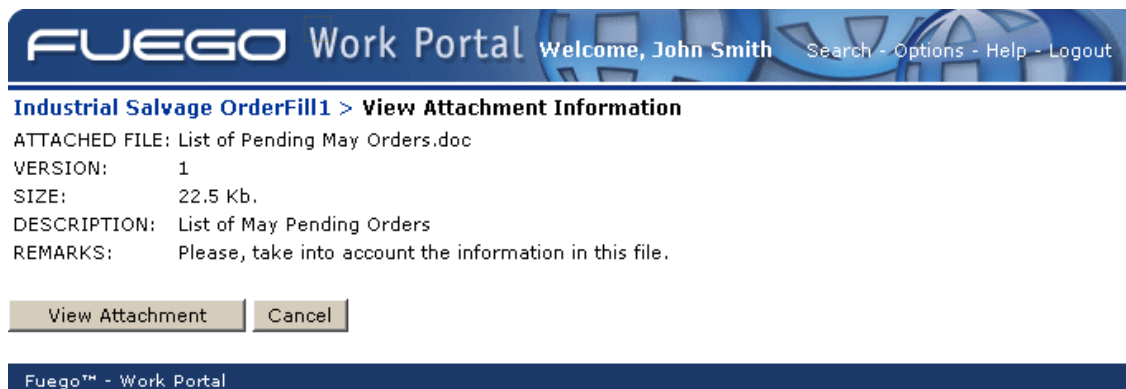
- java.lang.String (REF)
- java.lang.String (METHOD)
- java.lang.String (PRESERVE\_STATE)
- java.lang.Boolean (ONLY\_BACTH\_ERROR\_LIST)
- java.lang.String (DETAIL)
- java.lang.Boolean (HAS\_DETAIL)
- java.lanf.String (HIDDEN\_EXCEPTION\_INFO)
- java.lang.Boolean (IS\_BATCH\_OPERATION\_EXCEPTION)
- java.util.HashMap (REQ\_HASH\_ERROR)

## sessionExpired.jsp



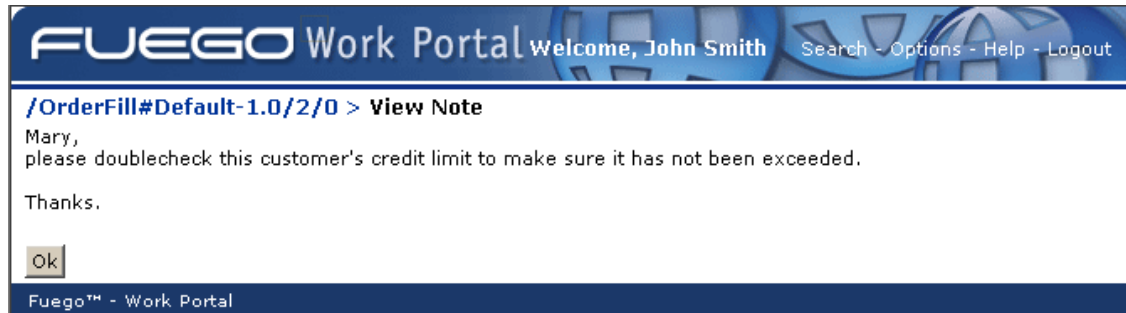
- KEY: SESSION\_EXPIRED
- Default filename: /jsp/sessionExpired.jsp
- Parameters:
  - java.lang.String (ORIGINAL\_ABSOLUTE\_URL)
  - java.lang.String (REF)
  - java.lang.String (PRESERVE\_STATE)

## viewAttachment.jsp



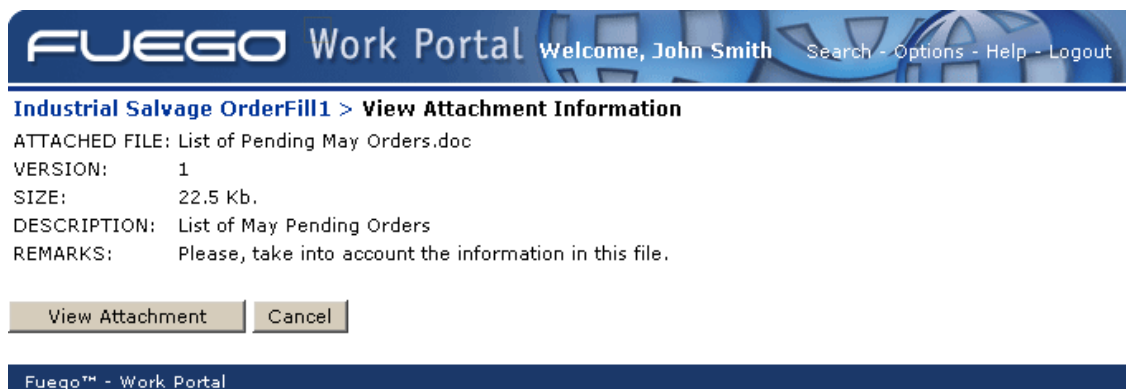
- KEY: VIEW\_INSTANCE\_ATTACHMENT
- Default filename: /jsp/viewAttachment.jsp
- Parameters:
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)
  - fuego.papi.Attachment (V\_ATTACHMENT)

## viewNote.jsp



- KEY: VIEW\_NOTE
- Default filename: /jsp/viewAttachment.jsp
- Parameters:
  - java.lang.String (REQUESTED\_REMARK)
  - fuego.papi.InstanceInfo (REQUESTED\_INSTANCE)

## viewWorkingAttachment.jsp

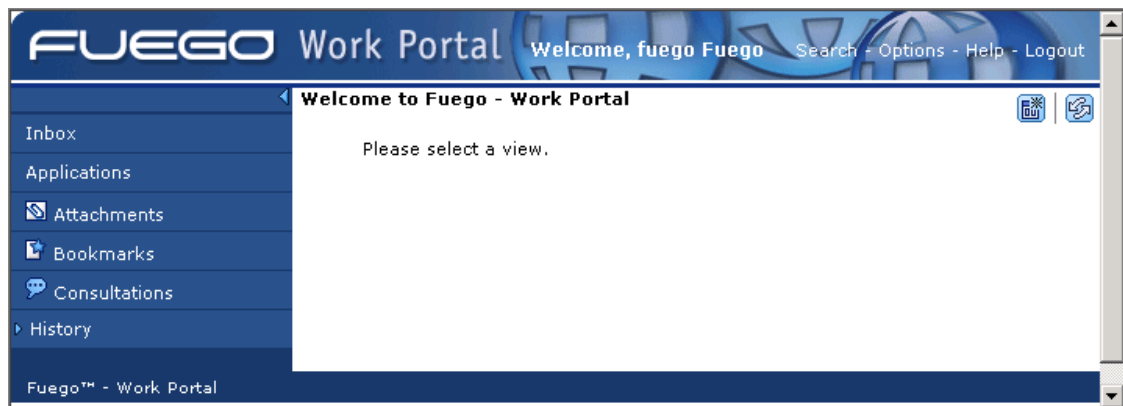


- KEY: VIEW\_WORKING\_ATTACHMENT



- Default filename: /jsp/viewWorkingAttachment.jsp
- Parameters:
  - fuego.papi.Attachment (V\_ATTACHMENT)

## welcome.jsp



- KEY: WELCOME
- Default filename: /jsp/welcome.jsp
- Parameters: NONE