Codeunit reference

Reference for common funtionality extensions

- Developing codeunits
  - Steps for creating a new codeunit
  - Different codeunit types
  - Error handling
  - Configuration / parameters
  - Variables
  - Using the provided Dev-image
- Deploying codeunits
- Formevents
  - Creating the codeunit
  - LIST level event hooks
  - ITEM level event hooks
  - FILTER event hooks
  - Event firing details
- Pagecontent
  - Description
- PagecontentPublic
  - Description
- Statusaction
  - Description
- Translator
  - Description

- ScheduledAction
- Message Modifier

Developing codeunits

Steps for creating a new codeunit

1. Create a new Java project in your favorite IDE
2. Import the p2eShared.jar in the project
3. Create a new class:
    1. The abstract parts are found in: dk.p2e.blanket.codeunit
    2. Implement all abstract methods
    3. Code the method bodies (normally "execute")
4. Compile and build
5. Deploy to webservser: [Application]\WEB-INF\lib

After first test of the codeunit a server reboot is needed for each redeployment, as there is no way of removing the loaded classes from memory.

Different codeunit types

Most likely you will need to create a Formevents that will allow specific changes for a solution, during views, updates or exports.

Most interactions wil require interaction with the following objects

- Security
- Command
- Session
- Context
- DbConnection
- EventHandler

Error handling

Exceptions are handled by themselves using the errorRegistration(Exception e) method.

In case the errors are not caught by the codeunit itself, the generic error handler takes over

1. Logs the error in the eventlog
2. Returns a standard error page to the user

Other debugging options include log4j and specialized TempusServa **Systemout.print** functions (yes, the class name is "Systemout").

```
Systemout.println("hello world");
Systemout.printSql(myStatment);
```

Configuration / parameters

Parameters are delivered through the method call, packed in a nice <String> Hashtable for easy access.

Please note that values are sent as-is, so make sure to escape values before DB operations using

```
DbConnection.escapeSql(String s);
```

Configurations can be stored using the getConfiguration methods of the command object. These values can be editedthrough the designer, depending on the Codeunit type

- System configurations: Designer > Modules > Static content
- Solution configurations: Designer > [Solution] > Advanced > Configurations

Value names will be prefixed with Class simple name - example + "."

```
com.tsnocode.MyExampleCodeunit.myParameter
```

Variables

Variables can have a scope of either

- Request
- User session
- Application

For permanent variables user server configurations.

Request variables

Request scope variables should be stored in the Hashtable **sharedObjects** in the **Command** object.

Note that the HTTP request parameters are stored in **requestParameters**.

User session variables

Acces the **sessionValues** attribute in the **Security** object

- boolean hasSessionValue(String name)
- void setSessionValue(String name, boolean value)
- int getSessionInteger(String name, int defaultValue)
- int getSessionInteger(String name)
- void setSessionInteger(String name, int value)
- String getSessionString(String name, String defaultValue)
- String getSessionString(String name)
- void setSessionString(String name, String value)

All variables her are serilizable and will survive server restarts.

Certain special user properties can also be accessed from the **Security** object

- boolean hasProperty(String name)
- String getProperty(String name)

Finally it is possible to store objects in the Hashtable **sessionObjects** in the **Security** object, but be aware that data are transient and **will not survive server restarts**.

Application variables

Application variables are persistent and accessed through the **Command** object

Their storage position depends on wether theres is a solution context or not:

- Solution present: Variables are saved to the solution **Configurations**
- None (SagID = 0): Variables are saved to the **Static content**

Access the the variables goes through get methods with default values.

- String getConfigurationValue(String name)
- String getConfigurationValue(String name, String defaultValue)

If the variable does not exist it will be created, and set to the default value (if present).

Note that it is possible to force the use of Statis content by explicitly calling

- String getSystemConfigurationValue(String name)
- String getSystemConfigurationValue(String name, String defaultValue)

Using the provided Dev-image

We provide a ready-to-go image of a virtual machine, running Debian 12, for VirtualBox.

The machine is set up with all the required software and helper scripts, to enable development of custom codeunits.

The image is setup with MariaDB, Tomcat 8, Netbeans 11.3 and a java-project that automatically deploys to the local Tomcat instance and has access to the TS-API with javadoc.

If the image is connected to a bridged network, the database and webserver are accessible from other machines (port 3306 and 8080 are open). use the command `ip a` to find the IP of the machine.

The image is running Debian 12 with a custom LXDE interface.

Getting the image running

The image is buildt for VirtualBox, so install that.

Once you have the image downloaded, start the import wizard in VirtualBox (ctrl + i) and follow it.

Now you should be able to boot the VM. The username and password for the user is noted in the comment on the VM.

The machine includes the following programs:

- Firefox, to access the local Tomcat server
- NetBeans 11.3, to develop codeunits
- DBeaver CE 24, to access the local DB

Deploying codeunits

To deploy your code to the local Tomcat instance do the following:

1. Open Netbeans

2. Build the project (right-click the project in the project-browser on the left and select build)
3. Copy the generated .jar file from `/home/developer/NetBeansProjects/[ApplicationName]/dist` to `/mnt/sda/deploy`
4. Clear the cache (open http://localhost:8080/app/service?CacheClear) or go back to the app in firefox and select Administrator -> Services -> Cache control -> Cache clear

Debugging codeunits

Attach the NetBeans debugger to the local Tomat server:

Click "debug" in the top-left menu -> click "attach debugger" -> click "OK", the default settings should work

Add a breakpoint to your code by selecting the line number and try to invoke it from the browser.

Updating the API and platform

To update the local environment to the latest version of the TS-API and TS Platform, open a terminal and execute the following commands.

```
updateAPI
ts -w app upgrade-alphaapp
```

You will be prompted to input the password for the user.

Deploying codeunits

Formevents

Custom code, attatched to an entity, getting called at different times

Creating the codeunit

1. Make sure that **p2eShared.jar** is included in the project
2. Create a new class that extends

```
com.tsnocode.codeunit.CodeunitFormevents
```

LIST level event hooks

```
public String appendListPageHead() { return ""; }
public String appendListPageFoot() { return ""; }
```

```
public void beforeSelectList() throws Exception {}
public void beforeRenderList() throws Exception {}
```

LIST execution order

1. **beforeSelectList**
2. Gather form data from database
    1. dataFilterActive > dataFilterHandler
    2. listFilterActive > listFilterHandler
3. **beforeRenderList**
4. Return list to user

ITEM level event hooks

```
public String appendItemPageHead() { return ""; }
public String appendItemPageFoot() { return ""; }
```

```
public void beforeSelectItem() throws Exception {}
public void beforeChangeItem() throws Exception {}
public void beforeUpdateItem() throws Exception {}
public void beforeRenderItem() throws Exception {}
public void afterUpdateItem() throws Exception {}

public boolean afterUpdateRedirectActive() { return false; }
public String afterUpdateRedirectContent()   { return null; }
```

ITEM execution order: viewing data

1. **beforeSelectItem**
2. Gather form data from database
   - dataFilterActive > dataFilterHandler
   - itemFilterActive > itemFilterHandler
3. **beforeRenderItem**
4. Return form to user

ITEM execution order: posting data

1. **beforeSelectItem**
2. Gather form data from database
3. **beforeChangeItem**
4. Update field values
5. **beforeUpdateItem**
6. Write changes to database
7. **afterUpdateItem**
8. if NO OTHER ACTION:
   - **afterUpdateRedirectActive**
   - if TRUE
     - **afterUpdateRedirectContent**
9. Return content to user

FILTER event hooks

Filters will help you build customized permission schemes. They are called for **both** LIST and ITEM commands.

```
@Override
protected boolean dataFilterActive() {
    return true;
}
@Override
protected void dataFilterHandler(StringBuilder sql) {
    sql.append(" AND something");
}
```

In some cases you only want the filter to trigger for **either** LIST or ITEM commands

```
itemFilterActive() {}
itemFilterHandler(StringBuilder sql) { return sql; }
listFilterActive() {}
listFilterHandler(StringBuilder sql) { return sql; }
```

Examples

The xxxFilterActive tells if the filter is active

```
boolean dataFilterActive() { return !s.isAdministor(); }
```

The xxxFilterHandler modifies the SQL used to fetch data

```
void dataFilterHandler(StringBuilder sql) { sql.append(" AND CATEGORY NOT IN (123,456,789)"); }
```

Event firing details

Event firing global

The following events are ALLWAYS fired

- beforeSelectList
- beforeSelectItem
- beforeChangeItem
- beforeUpdateItem
- afterUpdateItem

Event firing in UI (reserved for normal users)

The following events will NOT be fired during imports etc.

- appendListPageHead
- appendListPageFoot
- beforeRenderList
- appendItemPageHead
- appendItemPageFoot
- beforeRenderItem

Event firing in UI depending on user actions

The following events are SOMETIMES be fired for normal users depending on navigation

- afterUpdateRedirectActive
  - Not executed in SUBFORM mode
  - Not executed in during imports etc.
- afterUpdateRedirectContent
  - Depends on a TRUE result from afterUpdateRedirectActive()

Pagecontent

A completely custom page, that the users can access via a link

Description

PageContent is activated by supplying the name of the class in a command

Example for the codeunit MySpecialPage

```
http://..../.../main?com.acme.samples.MySpecialPage
```

Note that is is only possible invoke classes that inherits the CodeunitPagecontent class

Supported methods are

```
public String execute( Command command, Security user, Hashtable parameters );
```

The resulting string is the inner part of the page. Wrappers, menus etc. are added to the output.

Example

```
package com.acme.samples;

import com.tsnocode.codeunit.CodeunitPagecontentPublic;

public class MySpecialPage extends CodeunitPagecontentPublic {
  @Override
  public String execute() {
    return "<h1>Hello world !</h1>";
  }
}
```

PagecontentPublic

A publicly accessable custom page. Same as Pagecontent, but public

Description

PageContentPublic are special variants of Pagecontent, that are publically accessible (they do not require a login).

Note that is is only possible invoke classes that inherits the CodeunitPagecontentPublic class.

Statusaction

Will execute just about anything for a single object in a determined state

Description

Status actions codeunits will execute just about anything for a single object in a determined state.

The codeunit is bound to a Status action, and can be executed either on entry, on exit or after a given amount of time.

As with other actions the execution is logged, in order to prevent double firing of an given event.

The event methods are

```
execute(int SagID, int DataID, int StatusID );
```

Translator

Used for handling special values during integration

Description

Translators are used for handling special values during integration: A translator is handed a value, that it can transform and return.

Supported methods

```
String execute( String config, String value, Connection conn );
```

ScheduledAction

A custom service, executed on a fixed interval

Message Modifier

Allows modification of messages sent by the system, just before they are sent