What's new in Telosys 1.1.0 ?

1/ Convention over Configuration (CoC)

This is the main improvement in this version. It allows to invoke services, screen maps and screen context without any configuration if the project's conventions are respected.
So it's not necessary to update the following configuration files :

- screens.xml
- ScreenRegistry.java
- ServiceRegistry.java

This improvement allows to test immediately the generated screens, without any configuration.
Each project can have its own conventions.

The Telosys conventions are designed to facilitate scaffolding based on the "Telosys Tools" Eclipse plugin.

→ Conventions for "services"

The project's conventions for "services" can be defined in the "**telosys.properties**", by specifying the Java packages where the services classes are located (the service is supposed to be invocated by its class name).

Example 1 (for a single package for all the services) :

```
Service.package.convention = org.demo.service
```

Example 2 (for a multiple packages) :

```
Service.package.convention1 = org.demo.service.rpc
Service.package.convention2 = org.demo.service.nav
```

Remark:
the "registerServicePackage(..)" method is still usable in the ServiceRegistry class.

→ Conventions for "screen contexts"

The project's conventions for "screen contexts" can also be defined in the "**telosys.properties**".
The "screen context" conventions are based on the "screen name".
The Java package containing the "ScreenContext" is built using the following symbolic variables :

```
${SCREENNAME}     The screen name "as is"
${SCREENNAME_UC}  The screen name converted in UpperCase
${SCREENNAME_LC}  The screen name converted in LowerCase
```

The class suffixes are specified on the same line

Example 1 (single convention) :

```
ScreenContext.convention = org.demo.screen.${SCREENNAME_LC}.${SCREENNAME},
            Data,  Mgr,  Trg,  Proc
```

Example 2 (multiple conventions) :

```
ScreenContext.convention1 =
      org.demo.screen.${SCREENNAME_LC}.${SCREENNAME_UC},
      dat,  mgr,  trg,  proc
ScreenContext.convention2 =
      org.demo.screen.${SCREENNAME_LC}.${SCREENNAME_UC},
      data,  manager,  trigger,  procedure
```

Example 3 (packages by class types) :

```
ScreenContext.convention =
      org.demo.screen.context.data.${SCREENNAME_UC}data,
      org.demo.screen.context.mgr.${SCREENNAME_UC}mgr,
      org.demo.screen.context.trg.${SCREENNAME_UC}trg,
      org.demo.screen.context.proc.${SCREENNAME_UC}proc
```

→ Conventions for "screen maps"

The project's conventions for "screen maps" are defined in the "**screens.xml**" file with the tag "**screen_conventions**". This tag defines all the values to use when the framework is trying to load a "screenmap".
The classical screen definitions (with the tag "screen") are still usable.

Example :

```
<screen_conventions
      template="tpl.jsp"
      body   = "/screen/${SCREENNAME_LC}/${SCREENNAME}.jsp"
      script = "/screen/${SCREENNAME_LC}/${SCREENNAME}.js"
      css    = "/screen/${SCREENNAME_LC}/${SCREENNAME}.css"

      contextid     = "0"
      contextname   = "${SCREENNAME}"
      contextaction = "none"
/>
```

2/ Screen configuration embedded in the screen-body JSP file

In order to bring together all the configuration options in the same file, it's now possible to define everything in the screen body JSP file.
It can be done using the "**ScreenConfig**" marker (embedded in a JSP or HTML comment).

The following variables can be used between "**[ScreenConfig]**" and "**[/ScreenConfig]**":

- Predefined variables : **contextname**, **contextid**, **contextaction**, **template**
- Template element values : **$**element_name
- Template element contents : **%**element_name

Example :

```
<!--
[ScreenConfig]

// specific context : name, id, action
contextname = S003
contextid   = 2
contextaction = open

// specific template
// template = TplElements.jsp

// template elements
$title = Book management
%msg2 = /include/message.inc

[/ScreenConfig]
-->
```

For screens defined individually in "screens.xml", it's also possible to specify the context-id, context-name and context-action, using the following attributes in the "screen" tag :

- contextid
- contextname
- contextaction

Example :

```
<screen name="Country" template="tpl.jsp"
            body="/screen/country/Country.jsp"
            script="/screen/country/Country.js"
            contextid="5" contextname="Foo" contextaction="none" >
     <element    name="title" value="Country screen" />
</screen>
```

3/ Response improvements

ScreenResponse and **ServiceResponse** are now descendants of the common abstract class "GenericResponse". So they share the same methods and are more homogeneous.

The rendering errors (errors in "renderer's JSP") are now reported in the XML response flow in the "view-errors" tag, in order to notify the "client side".

The "ScreenResponse" JavaScript class has the following new methods :

- hasErrors()
- getErrors()
- getErrorsAsText()
- hasExceptions()
- getExceptionsAsText()
- hasValues()
- hasMessages()
- hasOutputErrors()
- getOutputErrors()
- getOutputErrorsAsText()
- showOutputErrors()
- reportsErrors()
- showAllErrors()

(see JavaScript framework documentation for more details)

4/ Other evolutions

Character encoding :
In order to support special characters in **services parameters**, all input and output characters are now encoded in UTF-8.

TagLib :

- Symbolic variables are now supported in the "fileupload" tag attributes

Javascript framework :
New functions :

- fwkRefreshImage
- fwkClearImage
- fwkReplacePage
- etc…

New methods for "**actions**" object :

- setParametersFromFields : each non-void field is transmitted as a request parameter
- numberOfParameters

Upload properties :

There's 2 new properties in "telosys.properties" for file upload operations :

- UploadMaxSize : the default maximum size for the upload http request size in Mega bytes
- UploadCharset : the charset to use to read the filename

Download mime-types :

The file download "mime type" is based on the "mime mapping" defined in the standard configuration file "web.xml"

Debug tools :

The last service request and response is now kept on the client-side and can be displayed using the following method :

```
ScreenServiceLastCall.show();
```

So it's now possible to show the last request/response for both action and service.

To offer this functionality, just add the following script in the screen bodies' template :

```
<script type="text/javascript">
	function debugShowLastServiceCall()
	{
		ScreenServiceLastCall.show();
	}
	function debugShowLastActionCall()
	{
		actions.showLastAction();
	}

	keyboard.setKeyAction(Keyboard.F9,  debugShowLastServiceCall,
				Keyboard.EVT_KEY_DOWN );
	keyboard.setKeyAction(Keyboard.F12, debugShowLastActionCall,
				Keyboard.EVT_KEY_DOWN );

</script>
```

F9 : shows the last service call
F12 : shows the last action call