HowToCFormsInLenya2.0

NOTE: Since version 2.0 it is possible to use cforms in combination with the usecase-fw. The following how to describes the basic steps you need to do. h1. Create a cform specific usecase handler Create an usecase like described in http://lenya.apache.org/1_4/reference/usecase-framework/index.html. Instead to use the default usecase handler use something like: {noformat} form.setAttribute("counter", new java.lang.Integer(0)); generic.doc = Packages. javax.xml.parsers.DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument(); generic.doc.appendChild(doc.createElement ("contacts")); form.save(generic.doc); saveDocument(generic.doc, generic.proxy.getParameter('sourceUri')); {noformat} The important part is @type=" cforms" in the element. This will tell lenya to use the special cform code. Everything inside the element can be used for custom flow definitions. Have a look on the usecase.js to see what is going on in detail. We are going to explain the basic concept with the and element. {noformat} /* * var generic - Generic object that can be used in all custom flow code * that is added by usecases. Right now it focus on document opertations * but the properties should be extended through user/dev feedback. * The intention of this var is to provide a generic global flow object * (like in lots of cocoon examples) that can be accessed through out the different flow stages. * Alternatively one can use the invoking java class by providing bean properties methods * (getter /setter methods) in this class and use them within the extensions (see usecase doc). */ generic={proxy:proxy,uri:null,doc:null,generic:null}; {noformat} You can use the generic variables to store values that you may need in later steps of your custom flow or in the cform. {noformat} // form definition form = new Form(viewDef); // custom flowscript if (view.getCformDefinitionBody()!=null){ scriptString= view.getCformDefinitionBody(); evalFunc = new Function ("form","generic",scriptString); evalFunc(form,generic); } {noformat} The dynamicRepeater example in cocoon needs a counter to be set in the flow script. {{{ form.setAttribute("counter", new java.lang.Integer(0)); }}} That can be done because we pass the "form" into the dynamic generated function. This way all changes that we may need in our custom flow will be applied to the form. {noformat} // form binding if (view.getCformBinding() != null){ var viewBind = "fallback://lenya/"+ view.getCformBinding(); if (cocoon.log.isDebugEnabled()) cocoon.log.debug("usecases.js::executeUsecase()::cforms in usecase " + usecaseName + ", preparing formDefinition, calling Cocoon with viewUri = [" + viewBind + "]"); // form binding form.createBinding(viewBind); // custom flowscript if (view.getCformBindingBody()!=null){ scriptString= view.getCformBindingBody(); evalFunc = new Function ("form","generic",scriptString); evalFunc(form,generic); } } {noformat} The binding for the dynamicrepeater example need a document to work. We can use the generic.doc variable to store it and pass it to the binding. This document you may want to save later on in the outro. This could be done with e.g. {noformat} form.save(generic. doc); saveDocument(generic.doc, generic.proxy.getParameter('sourceUri')); {noformat} The other interesting usage of the the generic variable can be seen here. It is possible to pass variables form your java class usecase to the flow and the cforms. {noformat} Document doc = getSourceDocument(); String sourceUri=doc.getSourceURI(); setParameter("sourceUri", sourceUri); {noformat} This way you can save the document back if it is invoked by a certain doc.