



Developer Guide

Yoshinobu Kano

November 13, 2009

A web-based version of this guide is also available on the U-Compare website [here](#).

This document is under construction but partially updated incrementally.

1 Using Third Party UIMA Components in U-Compare

This section describes how to use UIMA components in general in the U-Compare system. Basic knowledge about the Java classpath and UIMA descriptors is assumed in this section.

1.1 Common Requirements

UIMA descriptors (type system, analysis engine, collection reader, etc.) should use `import name="..."`, not `import location`, at least for the descriptors visible from U-Compare. Children in an aggregate component could have `import location` tag, but we cannot assure that it works or not.

1.2 Register Components Menu

1.2.1 Custom Classpath Setting

In the *Library* menu of the U-Compare workflow manager (the startup window), there is *Register Components* menu item. When you open this menu, a classpath setting dialog will appear. You can add your custom classpath entries, *.jar or any directory as Java normally allows, during runtime via this dialog. Due to the Java class loading mechanism, you may have to restart U-Compare to apply your changes, especially in cases some of your resources already loaded should

be masked by the new classpath entries (or vice versa). Pay attention that such a masking may cause unexpected behaviours.

1.2.2 Registering Custom Components

Then check your custom classpath entries on, which includes your component descriptors, and click *Search Component* button at the bottom of the dialog. A list of component descriptors will appear if any, in another dialog window. U-Compare validates your descriptors whether the xml syntax is valid, file dependencies are not missing, etc., and shows errors if any. Please select the check boxes of your components, click *Add to Library* button. You will see your selected components in the *Custom* category of the *Component Library* on the right of the workflow manager. You can drag to change the category which your custom components belong to, or right click to show the delete menu, etc. Please refer to the User Guide for details.

Now you can use your custom components in the drag-and-drop way together with the predefined U-Compare components. However, please note that the mixture of components with different type systems would not work properly in most cases.

Both of the classpath setting and the component library setting are automatically saved. U-Compare checks the classpath entries when startup and shows a warning dialog, if any entry or any descriptor is missing in your file system.

1.3 U-Compare Import/Export Package

U-Compare provides a special importable archive format with *.ucz* file extension. We considered to use the UIMA PEAR package, but unfortunately concluded that PEAR is not proper for our case. One of the core aims of U-Compare is to provide the highest interoperability, which allows users just to drag-and-drop to use components. On the other hand, PEAR is intended to be more generic purpose including native tools which might require manual, complex installation process, sometimes not OS independent.

1.3.1 Export Workflow, Components and Resources

You can export workflow descriptor, component descriptors, resources (**.jar* and all files under the directory of the specified classpath entries). Because it is impossible for U-Compare to detect which resources are required to run your current workflow, you should select the classpath entries by yourself.

The primary use case is to export the current workflow with necessary component descriptors and resources. Component descriptors which are predefined and provided by U-Compare are not included in the exported *.ucz* archive.

It is also possible to use this export function to create a custom component package. When you check the workflow descriptor checkbox off, the archive will

contain component descriptors included in the current workflow and specified resources if any.

You can distribute the generated .ucz archive to any U-Compare users, if the licenses of the contained resources allow. We recommend the U-Compare distributable package, but it is also happy for us to host your .ucz file in our U-Compare website. Please contact us or report the created package in the U-Compare Forum.

2 Command Line Mode without U-Compare GUIs

UCLoader is our default launcher system, where UCLoader.class is launched via command line, while UCLoaderLauncher is intended to be used for GUI based startup. Please refer to <http://u-compare.org/launch.html> for details.

UCLoader can be used as a pure command line tool with specific options. This section describes about such usages.

2.1 Common Requirements

Java 6 installed, U-Compare should be launched by UCLoader at least once. Internet connection required when you used web service components, and when you launch UCLoader for the first time.

Please download UCLoader.class from <http://u-compare.org/downloads/UCLoader.class>

2.2 UCLoader Options

Assuming that you saved UCLoader.class in the current directly, run:

```
java -cp . -XmsXXXm -XmxXXXm -Djavaws.workflow.path="path/to/yourworkflow.xml"  
UCLoader --jnlp http://u-compare.org/lib/u-compare-runworkflow.jnlp
```

“-cp” is the Java VM option to specify your classpath (in this case the current directory “.” is specified). You should include UCLoader.class in your classpath.

“-Xms” and “-Xmx” are the Java VM option to specify the amount of heap memory allocation.

You should set “-Djavaws.workflow.path” value to specify the location of your workflow descriptor file (see details below). The file should be on the classpath, and the location should be specified based on the classpath root.

This launcher runs any UIMA CPE workflow descriptor. All of the required resources should be on your classpath, but you don't have to include U-Compare predefined resources. Before launching the workflow, all of the U-Compare resources are specified on your classpath, the environment is the same and shared with the GUI mode.

2.2.1 Example Workflow

Please download the example workflow descriptor from <http://u-compare.org/downloads/StdinProteinTagging-CPE.xml> . Save this file to the directory where the UCLoader.class is stored.

Then run:

```
java -cp . -Xms400m -Xmx800m -Djavaws.workflow.path="StdinProteinTagging-CPE.xml" UCLoader --jnlp http://u-compare.org/lib/u-compare-runworkflow.jnlp
```

after initialization finished, please enter English sentences and press enter. You will get annotations processed by a sentence splitter and a protein mention tagger, in the standard output stream shown in your console.

Note that this string-then-enter input is a convenient way, the formal format is described below.

2.3 U-Compare Simple Stand-Off Format

Since this option runs any UIMA CPE workflow, it is up to your workflow what happens – your components may create GUI windows, network connections, etc.

However, for the developer’s convenience, we prepared I/O components which takes input from the standard input stream, outputs all of the generated annotations to the standard output stream. The above example workflow uses these components.

The input format is defined as:

```
[byte_length_of_rawtext]\n[rawtext]\n[annotations]\n\n
```

Example:

```
21
This is the raw text.
annotations...
```

The output format is equal to the annotations part format of the input.

Annotations part is defined as:

- annotations are lines which end with a blank line. Annotations are separated into lines. Each line represents a single annotation.
- annotation consists of fields separated by white spaces.
- Legend: begin end typename uniqueID [featureName="value"—featureName=referredUniqueID]*

e.g.

```
22 26 jp.ac.u.tokyo.is.s.www_tsujii.bio.Protein id="gene_prod3" category="ion
channel"
```

```
32 46 .Protein id="gene_prod82" category="ion channel"
```

```
00 jp.ac.u.tokyo.is.s.www_tsujii.bio.Interaction p1="gene_prod3" p2="gene_prod82"
```

Notes:

- begin and end are the offsets in integers. Both 0 for non-span annotations. Offsets are character-based counts.
- typename is the UIMA defined full-package name of the type of this annotation. start with `.(dot)` is an abbreviation for the default package (`jp.ac.u.tokyo.is.s.www_tsujii`)
- uniqueID is a concatenated string value of one or more alphabet letter(s) and a unique integer for that letter(s). Default prefix is “u”.
- featureName and its value type should be equal to the UIMA side definition of the annotation type. Literal values should be quoted by double quotations and escaped in the XML manner. Reference to the other annotation should not be quoted. Its value is a unique ID.

2.4 Editing Workflow Descriptor

The workflow descriptor is a UIMA CPE (Collection Processing Engine) descriptor, which definition is fully written in the Apache UIMA official documentations.

However, it is simple to edit the workflow to modify which components to be called. Please open the example workflow descriptor above in your text editor. You will see `<casProcessor>` xml tags, each of them corresponds to an analysis engine processor, called in the order described in this xml file. If you want to delete one of the processors, just delete the whole corresponding `<casProcessor>` xml tag block. If you want to insert new processor, copy one of the `<casProcessor>` xml tag block in a proper place, then change the name field of the `<import>` xml tag in that `<casProcessor>`. This field specifies the location of the component descriptor file, path from the classpath root concatenated by dot(`.`), without the `.xml` suffix.

If you want to use the U-Compare predefined components, it is enough just to specify this field. Please refer to the next section, using U-Compare components from your UIMA workflow, for how to get the locations of the U-Compare components.

2.5 Calling U-Compare GUI Generated Workflows

Upcoming.

3 Using U-Compare Components within Your UIMA Workflow

You can use individual U-Compare components without the U-Compare platform. If you are a UIMA developer who makes your own UIMA workflows from scratch, you might prefer this way. However, please pay attention that mixing

U-Compare components with components developed by other groups may cause type system incompatibility, produce no annotation.

In this section you are assumed to know how to create UIMA workflows.

3.1 Fetching Jars by UCLoader

Please launch UCLoader (see Launch U-Compare page in the U-Compare website for details), then all of the required library jar files are stored in your [your.user.home]/.U-Compare/jars directory. You have to put these jar files into your classpath.

3.2 Component Import Path Name

All of the U-Compare components are assumed to be used via the classpath loading, i.e. <import name>. should be used in your descriptors.

Unarchive u-compare.jar file in the [your.user.home]/.U-Compare/jars directory. A jar file is compatible with zip so you can use unzip tools to unarchive.

In the unarchived directory, /descriptors/categories.xml has the list of ready-to-use components with their import-name paths. Please note that this list might be updated without notification; update is automatic by using UCLoader.

4 Making Your Tools into U-Compare Compatible

Please refer to Section ?? about how to wrap your original tools to a UIMA component and run it in U-Compare. This is the prerequisite of this section. In this section we assume that you already have your UIMA components running in U-Compare or you know how to make UIMA components which can be run in U-Compare.

4.1 Common Requirements

4.1.1 U-Compare Type System Compatible

U-Compare compatible simply means that a component can be used in a workflow together with other U-Compare components. *U-Compare compatible* component should be a UIMA component and should use the U-Compare type system, or extended types of the U-Compare type system, for the component I/O capability, not just in the component descriptor setting but in the actual annotation instances.

4.1.2 Type System Converter

Because UIMA provides *Aggregate Analysis Engine* which holds nested child components, only the outermost component is required this type system compatibility which is visible to the users. In cases there already have UIMA components with another type system implemented, creating a *type system converter* component would be the straightforward way because there are no need to modify the original component. Then create a new aggregate analysis engine component with proper I/O capabilities of U-Compare type system compatible types, put three components in this order: the type system converter (another type system to U-Compare type system), the original component, and the type system converter (U-Compare to another). If the type system converter properly converts annotation instances, the newly created aggregate analysis component is now a *U-Compare compatible component*.

4.1.3 Collection Reader cannot be inside Aggregate Component

CollectionReader, which is another type of UIMA component like AnalysisEngine, normally reads corpus and generate CASes. Because CollectionReader cannot be in the aggregate component, type conversion should be processed inside CollectionReader if any. If you already have your own CollectionReader of another type system, some modification would be required.

4.2 U-Compare Type System

Please refer to the section of U-Compare type system for details of our type system.

4.3 U-Compare Distributable Component

The components shown in the component library of the U-Compare startup window are *U-Compare distributable component*. *U-Compare distributable component* is required to have a somewhat higher level of the interoperability than the *U-Compare compatible components*, to allow users to use that component by a simple drag-and-drop mouse motion. If your component has this level of interoperability, we can share and distribute your component via U-Compare. This subsection describes how to achieve such a level of the interoperability.

Another option is to provide your resource in the U-Compare importable package style. Please refer to the U-Compare import/export subsection, in the Using Third Party UIMA Component section above.

4.3.1 UIMA SOAP Web Service

Apache UIMA provides an easy way to deploy any UIMA analysis engine as a SOAP web service. If your implementation includes non-Java codes, please make use of this web service architecture. A collection reader cannot be deployed as

a SOAP web service. Please read the following sections about the local service implementation, in case of the collection reader.

Assuming that your component is already U-Compare compatible, first deploy your component as a SOAP UIMA web service, then you should have your UIMA soap service descriptor file. Please create an aggregate component which simply contains the web service descriptor only, with proper I/O capabilities using the U-Compare type system. This is because the soap service descriptor cannot have any information other than the service URI.

The last thing to do is making and sending us a jar file, which contains all of the required descriptors in their proper classpath location. After confirming and deciding to add your component in our component library, we will add your jar file to the U-Compare pre-defined resources and library.

4.3.2 Pure Java Component as Local Service

A pure java component is the most preferred way in U-Compare. We can automatically distribute your component via U-Compare, running in the users' local environment. The requirement is the classpath based access to the resources. Please read the next subsection for details. Current system environment is Java 6 and Apache UIMA 2.2.2. For the other libraries included in the system, please refer to [your.home.directory]/.U-Compare/jars folder after running UCLoader at least once. Please do not include the same classes/jars which are already included in the library above.

When you created your component, put all of the required resources into your jar files with proper classpath locations. You can create any number of jar files, but create a jar files of descriptors only, isolated from other resources. This is because the Java Web Start distribution is on demand, but the descriptors are always loaded when starting U-Compare. We can avoid unnecessary downloads by this isolation.

4.3.3 Loading Resources via Classpath

Because the whole U-Compare system is provided in the Java Web Start technology based manner, all of the resources should be in *.jar files which are on the classpath. This means that all of the resources should be accessed via the Java ClassLoader, i.e. accessed only by classpath based locations. Using the classloader directly would be the normal way in Java, but in the UIMA's case we have UIMA ResourceManager. You should first set `externalResource` field, which has a key-url pair, in your component descriptor. Then in your Java code,

```
UIMAContext uimaContext = getUimaContext();
URI uri = uimaContext.getResourceURI("KeyToYourResource");
InputStream in = uri.toUrl().openStream();
...
```

If you need to extract the entire jar file information,

```
UIMAContext uimaContext = getUimaContext();
```

```
URI uri = uimaContext.getResourceURI("KeyToYourResource");
JarFile jarFile = ((JarURLConnection)uri.toUrl().openConnection()).getJarFile();
...
```

In other words, if there is any use of `java.io.File`, then your component will not work in the Java Web Start based system, i.e. not U-Compare distributable.

4.3.4 Verifying Your Component

After creating your U-Compare distributable component, please verify your component as follows. First start U-Compare. Add your jar files to the classpath list of U-Compare from the *Register Component* submenu in the *Library* menu. Then search for your descriptors and add them to the library. Now you are ready to test your components as in the same environment as the predefined U-Compare components.

4.3.5 Contributing Your Component Packages

Before sending your packages, please make verification as above, and set the name and description field in your descriptor files with a user comprehensive expression. Then please send us all of the *.jar, together with a brief tool description which we can use in the tool list of our website. In your *.jar files, do not include classes and resources which are already provided by the U-Compare system as described above. Please refer to the *Pure Java Component as Local Service* section above for details.

5 U-Compare Type System

The U-Compare type system is designed to be both *shared* and to be used in *comparison and evaluation*.

The U-Compare type system descriptor file can be found as below. Launch UCLoader once, unarchive `u-compare.jar` file in the `[your.user.home]/.U-Compare/jars` directory. A jar file is compatible with zip so you can use unzip tools to unarchive. (If you need the developmental version, please unarchive `u-compare-devel.jar`.)

In the unarchived directory, `/org/u_compare/U_compareTypeSystem.xml`, is the type system definition file. Please note that the xml file should be located under `/org/u_compare/` path on your classpath setting, i.e. in the descriptor files it should be referred as `<import name="org.u_compare.U_compareTypeSystem">`.

Currently we have this single file for the type system, but we plan to divide the file into a couple of sub type system files. In this case you will find `<import>` declarations in the main `U_compareTypeSystem.xml` file, which refer to the dependent sub type systems. This is just for browsing the type system, it is enough to specify the main type system only as above for using the type system.

5.1 Shared, But Not Common, Type System

We are not planning to make our type system as a common single type system. It is apparently impossible to create a single common type system, because the concept or category which a type represents could be more or less different, even for the level of the individual person. However it is also apparent that lacking the type system compatibility largely spoils the interoperability which UIMA could provide.

Then the solution would be creating type system converters. Because creating a type system converter whenever a new type system is created/updated is not realistic, using a *shared* type system which could bridge local type systems would be the better way.

Information loss is more or less inevitable when converting type systems, but the less the better. We have designed the U-Compare sharable type system to decrease such information loss as less as possible. Firstly, by making types as hierarchical as possible, we can use the intermediate types to share the middle level category information. Secondly, UniqueLabel and its descendant types (like the Penn Treebank tagset) to make category labels unique, rather than the text string feature, assures the higher level of the uniqueness.

5.2 Type System for Comparison and Evaluation

One of the central features of U-Compare is to compare similar annotations over different components or evaluate versus gold standard data. Because we can only use the types and type system but not the instance information in the descriptor level, U-Compare detects which annotations to compare, based on the types and their type system hierarchy. It turns out to require the type system design to be used for *comparison*, i.e. hierarchical enough and organized properly to share the ancestor types.

Currently the Apache UIMA type system should be in the tree structure class, but Apache UIMA will provide the multiple inheritance as ECore compatible in future along with the UIMA specification. Then we can make more proper type system hierarchy.

Although the basic direction is clear as above, it is very difficult to express the real world concepts in such a hierarchy, as the researches of the ontology building show. Actually it would be impossible to create a single unique type system. Even how deep the type system should be is a large problem, there seem to be no clear solution exist.

5.3 Using and Extending U-Compare Types

First you need to find a proper supertype which is expressing the type of your instance, in the U-Compare type system. Then it is recommended to create your own type by extending that supertype, defined in your own type system descriptor file. It allows you to change the supertype without modifying the code so much, also allows you to add your own features. U-Compare type system

will be expanded continuously, there might be more suitable supertype appear in future.

5.4 Covered Field

Currently the U-Compare type system covers basic syntactic/semantic/document annotation types. If you are planning to contribute your component but have any type of annotations not covered by our type system, please contact us.

As for the explanation of each type, please refer to our paper (Kano, et al., 2009) presented in NAACL SETQA workshop. The paper can be downloaded from <http://www.aclweb.org/anthology/W/W09/W09-1504.pdf>, which includes diagrams of the main part of the type system and overview description for each type.

6 Combinatorial Comparison System Manual

Upcoming.

7 Pluggable Evaluation System Manual

Upcoming.