XOTcl for OpenACS developers

a tutorial presented at the
Openacs and .LRN Spring Conference 2007
in Vienna, Austria

Martin Matuška

Institute for Information Systems and New Media
Vienna University of Economics and Business Administration

April 25, 2007

Abstract

This tutorial provides an example-based introduction to XOTcl, a dynamic object-oriented extension to the scripting language Tcl and briefly explains its interaction with the web server AOLServer and the OpenACS/.LRN community framework. It is intended for audience with knowledge of Tcl scripting, AOLServer and OpenACS. A short overview of object-oriented programming basics is included.

1 Basics of object-oriented programming

Object-oriented programming (OOP) is a computer programming technique that makes use of data structures called **objects** and **classes**. This chapter gives a brief introduction to these and related terms. The Unified Modeling Language[1] (UML) from the Object Management Group is a standardized specification language for object modeling. The figures in this chapter are parts of UML 2.0 class diagrams.

1.1 Objects

Objects are abstract data types. They usually represent real entities (e.g. city, train or person). Each object contains individual information (e.g. a person has a name) and individual operations (e.g. walk, move). The integrated information can be accessed only with a operation (e.g. display name). The integrated information is stored in **object attributes** and the integrated operations form **object methods**.

An object may be connected with other objects. In OOP the connection of two or more objects is referred as **object association** (e.g. a car is associated with a driver). Two important types of object association where a superordinated object contains subordinated objects are **object aggregation** and **object composition**. Aggregated objects are connected by reference and form a one-to-many (parent-child) relation where one parent object may have n child objects and each child object may have only one parent object. Object composition adds a "lifetime responsibility" to aggregation. If a parent object is deleted, all child objects (associated with object composition) are deleted, too. An example of an aggregated object would be a copmany (parent) which contains employees (child). If the company is closed (=object is destroyed) the employees still exist as individual objects. On the other hand, a composed object may be a building (parent) that contains rooms (child). If the building is destroyed, the rooms are destroyed, too.

[1] Unified Modeling Language: http://www.uml.org/

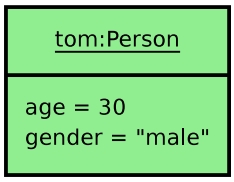

Figure 1: Object *tom* of class *Person* with attributes in UML

1.2 Classes

Many objects have common characteristics - they share common attributes and/or methods. A **class** represents a template for several objects and describes their internal structure. E.g. class *Person* might describe common attributes *name*, *gender* and *age* and common methods *walk*, *sit* and *sleep*. In OOP each object belongs to a class. An object that belongs to a specific class is called an **instance** of that class.

Classes may have common characteristics, too. For example, classes *Student* and *Worker* might use the same attributes *name*, *gender* and *age* but class *Student* has an individual attribute *school* and class *Employee* an individual attribute *salary*. We can group these common attributes into a separate class *Person*. This process is called **generalization**. The classes *Student* and *Worker* **inherit** the common attributes (and/or methods) from the class *Person*. The individual attribute *school* **specializes** the class *Student* from the superclass *Person*. Superclasses that usually have no instances (objects) and are developed with the purpose of being inherited by other clases are called **abstract classes**.

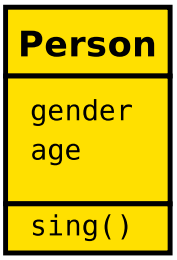

Figure 2: Class *Person* with attributes and methd in UML

1.3 Inheritance

Classes build an inheritance hierarchy. Classes placed below a class in this hierarchy are its **descendands**, classes placed above are its **ancestors**. The class that directly inherits another class is called a **direct descendant** or a **subclass** of the inherited class. The class being inherited is a **direct ancestor** or a **superclass** of the lower class.

Jacobson et al. defines four main purposes of inheritance (Jacobson et al., 1992, p. 64):

- **Reuse**
  Inheritance simplifies reuse of code. Abstract classes integrate common code and objects use code found in classes.
- **Subtyping**
  Classes can be seen as an implementation of (sub-)types. This is only possible with behaviorally compatible classes. In this case subclasses **extend** superclasses (new attributes and methods are added).
- **Specialization**
  If subclasses are not behaviorally compatible, their structure was altered. If a superclass cannot always replace its subclasses, they **replace** (parts of) the superclass.

- **Conceptual**
  Inheritance is similar to intuitive semantic, e.g. a boy is a human. This enhances the readability and understandability of source code.

In addition to simple inheritance there is a concept of **multiple inheritance**. With multiple inheritance one subclass can have multiple superclasses. There are two principial ways, how attributes and methods of more than one superclasse can be accessed. First a separate naming for inherited attributes and methods from each superclass may be used. Second, attributes and methods from the superclasses may use a solution called **overriding**. All ancestors are organized in a ordered list. The first ancestor in the list that contains the requested attribute or method overrides all others. XOTcl uses this approach.

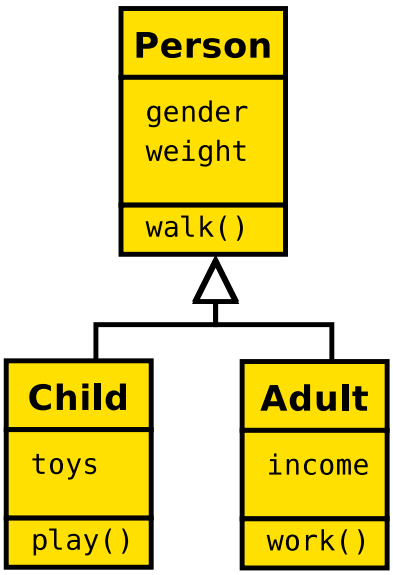

Figure 3: Class inheritance in UML

1.4 Static vs. Dynamic OOP

The most known object-oriented languages have static OOP. The probably most used of these are C++ and Java. In static OOP languages, classes and class hierarchy are statically defined and cannot be changed at runtime. Modifying the class hierarchy (e.g. replacing top-level classes) of an already developed application is possible but expensive.

Dynamic OOP languages enable applications to be customized wihtout access to source code during development and after deployment. Examples of dynamic OOP languages are Smalltalk, CLOS and XOTcl.

> "With Dynamic OOP languages, the amount of work necessary to make a change is proportional to the degree of change, not the size of the application. New objects, new classes and new behavior can be added on the fly, and unlike static OOP languages, Dynamic OOP applications do not have to be rewritten to accommodate any change." (Franz Inc., 1997)

Dynamic OOP languages ease the implementation of advanced software engineering concepts like patterns (Gamma et al., 1995) and aspects (conference AOSD, 2007). One of the solutions for implementing design patterns is using filters (Neumann and Zdun, 1999).

2 Object-oriented programming with XOTcl

From the XOTcl homepage[1]:

> Extended Object Tcl (for short: XOTcl, pronounced exotickle) is an object-oriented scripting language based on Tcl. It was originally designed for providing language support for design patterns and provides novel constructs such as filters or transitive mixin classes. The language is designed for empowering rather than constraining system developers. The basic object model is highly influenced by CLOS.

The open-source Tcl extension XOTcl (Neumann and Zdun, 2000) is a value-added replacement for OTcl (MIT Object Tcl, Wetherall and Lindblad, 1995) and does not require OTcl to compile. Figure 4 shows an overview of XOTcl features. For more information about XOTcl features, see the the XOTcl homepage and the XOTcl@Work tutorial (Neumann and Zdun, 2001).

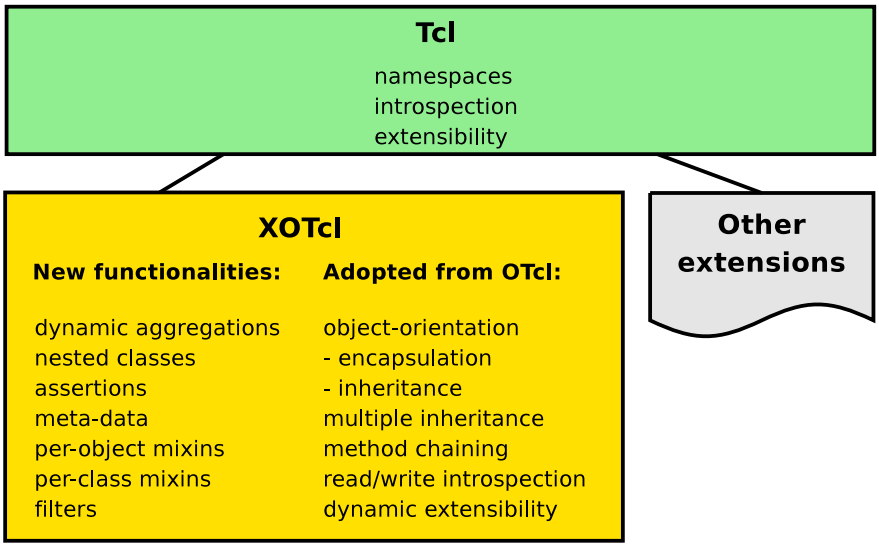

Figure 4: Overview of XOTcl features

XOTcl is included in several Tcl distributions including the industry standard Tcl distribution ActiveTcl. Table 1 lists some Tcl distributions that contain XOTcl.

Distribution	Operating system	Homepage
ActiveTcl	MS Windows, Linux, Solaris, ...	http://www.activestate.com/activetcl
WinTclTk	Microsoft Windows	http://wintcltk.sourceforge.net
Tcl/Tk Aqua	Mac OS X	http://tcltkaqua.sourceforge.net

Table 1: Tcl distributions with included XOTcl

Almost all linux distributions (RedHat, Suse, Fedora, Debian, Gentoo, etc.) include Tcl packages. XOTcl can be easily compiled and installed from sources on all these systems using GNU configure and GNU make. Debian Linux and FreeBSD include XOTcl packages. For more information please read documentation bundled with the source code release of XOTcl.

The XOTcl source code is downloadable from the XOTcl homepage. The XOTcl homepage includes a language reference and a XOTcl Tutorial in HTML and PDF formats. At the time of writing this document XOTcl is available in version 1.5.3. The source code distribution contains additional libraries. Some of these are listed in table 2.

[1] XOTcl homepage: http://www.xotcl.org

Libary	Description
xotcl::actiweb	active web object system (building web applications with active web documents)
xotcl::comm	internet protocols library (http, ftp, imap, ldap, dav, ...) and http server
xotcl::rdf	XML/RDF parser and interpreter
xotcl::store	general persistent data storage (supports SDBM, GDBM and other backends)
xotcl::xml	XML and SGML parser (supports TclXML and expat)

Table 2: Selected XOTcl additional libraries

2.1 Objects in XOTcl

To get started with XOTcl, the XOTcl package has to be loaded first:

```
% package require XOTcl
1.5.3
```

By default, XOTcl commands can be accessed only with the package namespace prefix (`::xotcl::command`). To load XOTcl commands into the global namespace, we need to issue the following command:

```
% namespace import ::xotcl::*
```

2.1.1 Creating and destroying objects

We can start programming in XOTcl and create an empty object. Each class and object in XOTcl register a new command and therefore require an unique name (identifier). The syntax for object creation is `Classname objectname`. In XOTcl all objects are instances of an always present meta-class *Object*. To create an instance of the meta-class object, we enter the following line:

```
% Object tom
::tom
```

An empty object named *tom* was created. This command is a shortcut for the `Object create` method. Using `Object create` is more secure because if the class *Object* had a method registered under the name *tom*, this method would be called instead of creating a new instance.

```
% Object create susan
::susan
```

The meta-class *Object* has several pre-defined methods. One of these is the *destroy* method used for destroying objects. To destroy the object *Susan*, we invoke:

```
% susan destroy
```

2.1.2 Object variables and procedures

The meta-class *Object* common to all objects provides several methods for object data manipulation. E.g we can **set** and **unset** object variables. Variable contents are requested with *Objectname* ***set*** *variablename*.

```
% tom set age 30
30
% tom set age
30
% tom unset age
```

Variables stored in XOTcl objects are treated like all other Tcl variables including arrays and lists. To allow manipulation of these variables, class *Object* provides methods like **append**, **array**, **lappend** and **incr** that are wrappers to the corresponding Tcl commands.

```
% tom set actions(now) "sleep"
sleep
% tom set actions(later) "cook"
cook
% tom array names actions
later now
% tom set friends peter
peter
% tom lappend friends susan
peter susan
```

The next step consists of registering methods for objects using the **proc** method. Methods are called with syntax *Objectname* ***methodname*** *arguments*. To make use of internal object variables in methods, we need to use the **instvar** method. To reference the current object, **my** or [**self**] is used.

```
% tom proc invite {friend} {
    my instvar invited
    lappend invited $friend
    puts "Friends invited: $invited"
}
% tom invite susan
Friends invited: susan
% tom invite peter
Friends invited: susan peter
```

To create child objects, the following syntax is used:

```
% Object tom::john
::jom::john
```

2.1.3 Object introspection

Class *Object* provides an introspective method **info**. With this method we can retrieve various information from the object like the list of object variables (**info vars**), object procedures (**info procs**), class membership (**info class**) etc.

```
% tom info vars
actions invited friends
% tom info procs
invite
% tom info class
::xotcl::Object
% tom info args invite
friend
% tom info children
::tom::john
```

2.2 Classes in XOTcl

XOTcl includes in addition to the meta-class *Object* a meta-class *Class*. This class provides predefined methods common to all classes in XOTcl. The class *Object* inherits class *Class*.

```
% Object info class
::xotcl::Class
```

2.2.1 Creating classes

New classes are created with the **Class** (or **Class create**) command. To create an empty class and an instance of it, we use the following commands:

```
% Class Person
::Person
% Person fred
::fred
% fred info class
::Person
```

There is a method named **instproc** to register procedures for classes. It has a similar syntax like the Objectname **proc** method.

```
% Person instproc sleep args {
    puts "[self] is sleeping."
}
% fred sleep
::fred is sleeping.
```

New methods added to classes are propagated to already existing instances as well (instance *fred* of the class *Person* may use the method **sleep**). There is a a special method named **init** that is invoked on the creation of every instance of a class. This method may be redefined or extended for each class.

```
% Person instproc init args {
    my set score 0
    puts "[self] was created with score: [[self] set score]"
    next
}
% Person jane
::jane was created with score: 0
::jane
```

The method **next** is used for chaining methods between Classes (and Objects). Calling **next** from a method invokes a identically named method (e.g **init**) of the next class in the precedence order, usually the superclass. The precedence order for **next** can be retrieved with the **info precedence** method. Figure 5 shows the UML 2.0 class diagram notation for the created class *Person* and its instances *fred* and *jane*.

```
% jane info precedence
::Person ::xotcl::Object
```

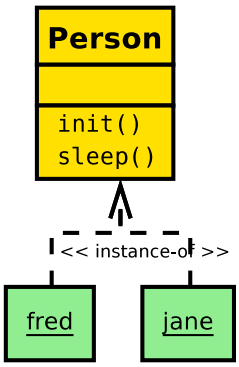

Figure 5: Class *Person* with instances *fred* and *jane*.

2.2.2 Inheritance

Classes in XOTcl may inherit another classes with the **-superclass** parameter upon class creation. A class may have more superclasses (multiple inheritance). The order of superclasses is important because the **method overriding** and the **next precedence** follow this order.

```
% Class Person
::Person
% Class Janitor -superclass Person
::Janitor
% Janitor instproc cook args {
    puts "I have no idea of cooking."
}
% Class Cook -superclass Person
::Cook
% Cook instproc cook args {
    puts "I am cooking a meal."
}
% Class Employee -superclass {Janitor Cook}
::Employee
% Employee jerry
::jerry
% jerry cook
I am cooking a meal.
```

The previous highlights the importance of superclass order in method overriding. The method **cook** was used from the very next superclass *Janitor* because it was supplied as the first argument in **-superclass**. In the next example we rewrite the cry procedure for class *Adult* by appending the **next** command at the end.

```
% Janitor instproc cook args {
    puts "I have no idea of cooking."
    next
}
% jerry cook
I have no idea of cooking.
I am cooking a meal.
```

The **cook** method was called from class *Janitor* and **next** command called this method from the next class in the precedence order: *Cook*. The next order for this example is visualised on Figure 6.

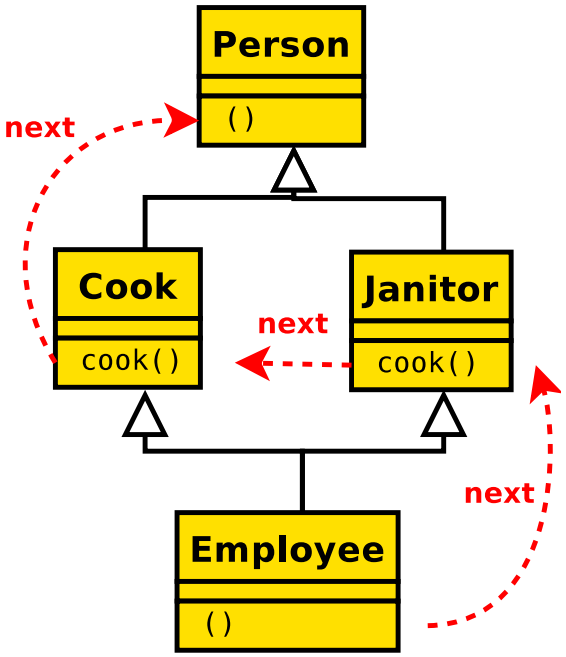

Figure 6: Example precedence order for XOTcl **next** command.

XOTcl objects may dynamically change classes and XOTcl classes may change their superclass relationships. Class *Object* provides method **class** and class *Class* provides method **superclass**.

```
% Class Male
::Male
% Class Female
::Female
% Class Child -superclass Male
::Child
% Class Adult -superclass Female
::Adult
% Child info superclass
::Male
# now we change Child's superclass from Male to Female
% Child superclass Female
% Child info superclass
::Female
# info heritage returns all ancestors in the precedence order
% Child info heritage
::Female ::xotcl::Object
% Female info subclass
::Child ::Adult
% Child Steve
::Steve
% Steve info class
::Child
# now we change Steve's class from Child to Adult
% Steve class Adult
% Steve info class
::Adult
```

2.2.3 Abstract classes

In XOTcl a class is defined abstract if at least one method of this class is abstract. An abstract method is defined with the **abstract instproc** command.

```
% Class Score
::Score
% Score abstract instproc setbase args {
  my set basescore 0
}
```

2.2.4 Class introspection

There is an introspective **info** method for all classes similar to the object's **info** method. For examples see end of section 2.2.2

All classes in XOTcl are objects, too. To check the type of a Object, methods **isobject**, **isclass** and **ismetaclass** can be used.

```
% Class City
::City
% City paris
::paris
% paris isclass
0
% paris isobject
1
```

```
% City isclass
1
% City isobject
1
```

2.3 Advanced OOP concepts

XOTcl provides special methods for implementing advanced OOP concepts like patterns (Gamma et al., 1995) and aspects (conference AOSD, 2007). These special methods operate with *mixins* and *filters*. For more information on these methods (**mixin**, **filter** for objects and **instmixin**, **instfilter** for classes) see the XOTcl Documentation[1] and the **xotcl::patterns** package set (included in source release).

3 AOLserver module

AOLserver modules extend the Tcl function set provided by AOLServer in a similar way packages add new Tcl functions. There are two types of modules - binary modules and script modules. Binary modules are written mostly in C, script modules are written in Tcl. AOLserver ships a couple of binary and tcl modules in the standard distribution. More information about AOLserver modules and an extensive module list can be found at AOLserver wiki[2]. Some binary modules used by OpenACS are listed on Table 3.

Module	Type	Description
nscache	required	Tcl interface to AOLserver's caching API
nspostgres	required	Internal PostgreSQL driver
nssha1	required	AOLserver module to perform SHA1 hashes.
nsopenssl	optional	SSL socket driver links against OpenSSL
nsldap	optional	LDAP client interface

Table 3: AOLServer binary modules used by OpenACS

The Tcl script modules are loaded from `.tcl` files located in a subdirectory (`modules/tcl`) of the AOLserver installation. They are loaded into the AOLserver **blueprint**. The blueprint can be described as a set of Tcl commands that includes all procedures from the provided Tcl module files. It is loaded for each connection thread . Examples of the Tcl script modules shipped with AOLserver are `http.tcl` (HTTP client), `sendmail.tcl` (SMTP client) or `form.tcl` (url-encoded or multi-part forms handler).

The XOTcl source distribution provides a Tcl module for AOLserver version 4.0 and higher (`xotcl.tcl`). This module loads the XOTcl package (no need for `package require XOTcl`) and initializes all necessary functions (e.g. classes and objects). XOTcl cannot be loaded into the AOLserver blueprint like the other modules do. To use Classes and Objects, they have to be initialized in correct order (e.g. meta-classes have to be defined first). To put XOTcl procedures (like Class, Ojbect, etc.) in correct order, the **xotcl::serializer** package is used.

The AOLserver XOTcl module is located in the `generic/aol-xotcl.tcl` subdirectory of the XOTcl source distribution. It is installed using the `make install-aol` command or by a manual file copy into `modules/tcl/xotcl.tcl` under the AOLserver installation directory.

[1] XoTcl Documentation: http://media.wu-wien.ac.at/doc/index.html
[2] AOLserver Wiki - Modules: http://panoptic.com/wiki/aolserver/Modules

4 OpenACS and XOTcl

OpenACS packages may take advantage of XOTcl features. There are three basic ways, how XOTcl could be used in OpenACS:

1. direct package loading - scripts load the package directly (`package require XOTcl`). This method is not recommended because of low performance and high memory requirements.
2. AOLserver module - XOTcl is pre-loaded by AOLserver
3. AOLserver module with xotcl-core package - the advanced function set of xotcl-core (section 4.1) significantly simplifies OpenACS programming.

To enable the operation of OpenACS packages running XOTcl, the AOLserver XOTcl module should be installed first. See section 3 for more information. With a loaded XOTcl module, there is no need for `package require XOTcl` and XOTcl commands may be used in all scripts. Examples of OpenACS packages that use XOTcl are:

- **XoWiki**[1] - a wiki implementation for OpenACS in XoTcl. XoWiki combines aspects of wikis (ease of page-creation) with aspects of a content management system (revisions, reusable content, multiple languages, page templates). The XoWiki package is used at the OpenACS homepage: http://www.openacs.org/xowiki/
- **XOTcl Request Monitor**[2] - request monitor for OpenACS applications. It computes performance summary information such as requests/views per seconds, average response time, number of users connected, lists currently active threads, etc.
- **Chat**[3] - a web-based chat implementation for OpenACS using different streaming techniques (under development)

4.1 xotcl-core

The **xotcl-core** package provides core functionality for OpenACS applications using XOTcl. This package is already included in the .LRN distribution, installation instructions for OpenACS are at following URL: http://openacs.org/xowiki/xotcl-core.

The xotcl-core package provides several classes including:

- **Api-browser API** - classes documenting XOTcl objects, classes and methods integrated with the api-browser of OpenACS
- **XOTcl simple Content repository API**
  Class CrClass (retrieve, modify and store objects in the content repository, supports categories)
- **Low level db abstraction API**
  Class DbPackage (OpenACS database access)
- **HTML Widget Classes** - creating HTML widgets based on tDOM
- **Thread management API** - support for AOLserver and XOTcl threads
  Class THREAD (create, initialize, destroy threads and pass commands to threads)
  Class Proxy (simplifies interaction with threads and hides that certain classes/objects are part of a thread)
- **Background file delivery** - manage background file delivery threads
  Resource-saving way of handling file uploads, uses class THREAD
- **Policies API**
  Class Policy (list/verify permissions and privileges)

[1] XoWiki homepage: http://media.wu-wien.ac.at/download/xowiki-doc/
[2] Request Monitor source: http://cvs.openacs.org/cvs/openacs-4/packages/xotcl-request-monitor/
[3] Chat source: http://cvs.openacs.org/cvs/openacs-4/packages/chat/

- **Generic chat** - classes for a web-based chat implementation

Almost all OpenACS packages that use XOTcl depend the xotcl-core package. To operate correctly, this package has to be loaded before these packages. OpenACS has no package dependency system yet[1], packages are loaded in alphabetical order. Package **acs-bootstrap**[2] ensures that xotcl-core is loaded and initialized prior to other packages.

4.2 Example code: object interface to content repository

This example[3] presents a simple way to access and manipulate items from the content repository using the class *CrClass* from the **xotcl-core** package.

First, we create an XOTcl object from an arbitrary content repository item provided that the type has a class definition (type is read from OpenACS).

```
CrItem instantiate -item_id 6413
```

The created object *6413* has all core and defined "extra" attributes of the content repository. Now we perform an operation on the object:

```
6413 append title " - Works perfectly"
```

We can save the object in a new revision

```
6413 save
```

or a new item can be created:

```
6413 save_new
```

To delete the object from database we use:

```
6413 delete
```

The destroy command would delete the XOTcl object but not the database entry.

Re-classing is possible, too:

```
CrItem instantiate -item_id 6417
6417 class ::xowiki::Page
6417 set title "en:[6417 set title]"
6417 save_new
```

4.3 Example package: xotcl-note

The example package **xotcl-note** was created by Gustaf Neumann for demonstrative purposes. It is a generic xotcl-based notes application using the object interface with object types and subtypes based on the content repository (with category support). Package uses package initialize for context in forms. The package can be downloaded from the following URL:
http://media.wu-wien.ac.at/download/xotcl-note-0.8.apm

References

Aspect-Oriented Software Development conference AOSD. Wiki. Webpage, 2007. URL http://aosd.net/wiki/index.php?title=Main_Page.

Franz Inc. Dynamic Object Oriented Programming. White Paper, 1997. URL http://www.franz.com/resources/educational_resources/white_papers/doop.lhtml.

[1] at the time of writing this document, OpenACS version 5.2.0
[2] file that loads and initializes XOTcl: packages/acs-bootstrap/bootstrap.tcl
[3] source: OpenACS Development forums: http://openacs.org/forums/message-view?message_id=346699

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1995. ISBN 0-201-63361-2.

Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering, A Use Case Driven Approach*. ACM press, Harlow, Essex, England, 1992. ISBN 0-201-54435-0.

Gustaf Neumann and Uwe Zdun. Filters as a language support for design patterns in object-oriented scripting languages. In *Proceedings of COOTS'99, 5th Conference on Object-Oriented Technologies and Systems*, San Diego, May 3-9 1999.

Gustaf Neumann and Uwe Zdun. Xotcl, an object-oriented scripting language. In *Proceedings of Tcl2k: The 7th USENIX Tcl/Tk Conference*, Austin, Texas, USA, February 2000. URL http://nm.wu-wien.ac.at/research/publications/xotcl-tclconf.pdf.

Gustaf Neumann and Uwe Zdun. Xotcl @ work. Tutorial at 2nd European Tcl User Meeting, June 8-9 2001. URL http://nm.wu-wien.ac.at/research/publications/xotcl-tutorial.pdf.

David Wetherall and Christopher J. Lindblad. Extending tcl for dynamic object-oriented programming. In *Proceedings of the Tcl/Tk Workshop 95*, Toronto, Ontario, July 1995. URL http://tns-www.lcs.mit.edu/publications/tcltk95.djw.html.