quercus: php in java

Quercus **is Caucho Technology's fast, open-source, 100% Java implementation of the PHP language. Performance is 4x mod_php and is comparable with PHP accelerator performance. Quercus uses Resin-IoC/WebBeans to integrate with Resin services.**

Introduction to Quercus

Quercus is Caucho Technology's fast, open-source, 100% Java implementation of the PHP language. Quercus is a feature of Caucho Technology's Resin Application Server and is built into Resin - there is no additional download/install. Developers using Resin can launch PHP projects without having to install the standard PHP interpreter (http://www.php.net) as Quercus takes on the role of the PHP engine.

What is Quercus

Quercus implements PHP 5 and is internationalization/localization (i18n/l10n) aware. Quercus natively supports Unicode and the new Unicode syntax of the up-and-coming PHP 6. Quercus implements a growing list of PHP extensions (i.e. APC, iconv, GD, gettext, JSON, MySQL, Oracle, PDF, Postgres, etc.). Many popular PHP applications will run as well as, if not better, than the standard PHP interpreter straight out of the box.

Resin with Quercus

Quercus is much more than just yet another PHP engine. Quercus is the first to tightly integrate the web server with a PHP engine. Quercus runs on top of Caucho Technology's Resin Application Server. As a result, PHP applications can automatically and immediately take advantage of Resin's advanced features like connection pooling, distributed sessions, load balancing, and proxy caching.

A New Java/PHP Architecture

Quercus is pioneering a new mixed Java/PHP approach to web applications and services. On Quercus, Java and PHP is tightly integrated with each other - PHP applications can choose to use Java libraries and technologies like JMS, EJB, SOA frameworks, Hibernate, and Spring. This revolutionary capability is made possible because 1) PHP code is interpreted/compiled into Java and 2) Quercus and its libraries are written entirely in Java. This lets PHP applications and Java libraries to talk directly with one another at the program level. To facilitate this new Java/PHP architecture, Quercus provides an API and interface to expose Java libraries to PHP.

Benefits of Quercus

Quercus and Quercus' PHP libraries are written entirely in Java, thereby taking the advantages of Java applications and infusing them into PHP. PHP applications running on Quercus are simply faster, easier to develop, more capable, more secure, and more scalable than any other PHP solution.

Quercus gives both Java and PHP developers a fast, safe, and powerful alternative to the standard PHP intepreter. Developers ambitious enough to use PHP in combination with Java will benefit the most from what Quercus has to offer.

Performance - simply faster

- Quercus outperforms straight mod_php by about 4x for MediaWiki and Drupal.
- PHP developers can use Java tools like profilers to get in-depth information about the PHP program performance.

Development - fast, safe, and easy

- PHP extensions written in Java are fast, safe, and relatively easy to develop compared to those written in C. Since Java is the library language, developers won't need to be paranoid about third-party libraries having C-memory problems or segvs and are freed to concentrate on solving the objectives at hand.

Capability - powerful Java technologies at the developer's fingertips

- Quercus has the best of both worlds: PHP and Java. PHP applications can take advantage of Java technologies like JMS, EJB, SOA frameworks, Hibernate, and Spring.

Security - no more pesky C memory bugs

- All Quercus extensions libraries are coded in Java. Therefore, developers do not have to worry about C pointer overruns and segmentation faults from PHP extensions anymore.

Scalability - Massive clusters of PHP

- Thanks to Resin, PHP applications can beautifully scale to as many servers as desired.
- PHP applications can now enjoy connection pooling, distributed sessions, fail-safe load balancing, and proxy caching. These benefits require no change in the PHP code.

Internationalization - 16-bit unicode

- Because Quercus is a Java implementation, it natively supports 16-bit unicode strings and functions. Quercus (in 3.1.0) supports the new PHP 6 internationalization syntax, and the older unicode conversion functions like iconv. Since 3.1.3, the new PHP6 unicode features are off by default but

they can be enabled with the PHP ini unicode.semantics=on.

Existing PHP applications on Quercus

Killer Apps: Mediawiki, Wordpress

Caucho has designated a few applications as Quercus "killer apps". For these applications, we take extra time to test that each new application version works well with Quercus. Any issues raised with the killer apps have priority over other Quercus bugs.

- http://wiki.caucho.com/Mediawiki
- http://wiki.caucho.com/Wordpress

Other applications

- http://wiki.caucho.com
- DokuWiki
- Drupal
- Gallery2
- Mantis
- Mediawiki
- Openads
- PHP-Nuke
- phpMyAdmin
- PHProjekt
- Vanilla
- Wordpress

Configuring Quercus

php.ini

Individial PHP initialization values can be set in resin-web.xml. For example, to set the settings for sending mail:

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <php-ini>
        <sendmail_from>my_email_address</sendmail_from>
        <smtp_username>my_email_username</smtp_username>
        <smtp_password>my_email_password</smtp_password>
      </php-ini>
    </init>
  </servlet-mapping>
</web-app>
```

A PHP style ini file can also be specified:

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <ini-file>WEB-INF/php.ini</ini-file>
    </init>
  </servlet-mapping>
</web-app>
```

Character Encoding

Quercus 3.1.0 supports PHP6 and has full support for Unicode. But like PHP6, Quercus 3.1.3 has its Unicode support turned off by default for compatibility with legacy PHP applications. Unicode support can be enabled with the php ini unicode.semantics.

With unicode semantics off, Quercus will intrepret bytes in the default ISO-8859-1 encoding. Quercus will behave just as PHP5 would. With it on, PHP5 applications may break and you need to be concerned with the following three encodings options: script-encoding, unicode.output_encoding, and unicode.runtime_encoding. By default, Quercus uses UTF-8 for all three.

Script encoding indicates the encoding of PHP script source files. If the source code for an application is not encoded in UTF-8, Quercus may give invalid UTF-8 conversions errors when it tries to convert bytes read to UTF-8. The solution is to tell Quercus to parse PHP scripts using the correct character set (ISO-8859-1 for most applications). For example, to tell Quercus to use ISO-8859-1, add <script-encoding> to the init tag of QuercusServlet:

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <script-encoding>ISO-8859-1</script-encoding>
    </init>
  </servlet-mapping>

</web-app>
```

If the PHP application also expects conversion from binary to string using a character encoding that is not UTF-8, then the unicode.runtime_encoding is used to specify the encoding. In PHP 6, there are two types of strings, Unicode and binary. A binary string is a string where the data is binary, the encoding is unknown, or the encoding is not Unicode (UTF-16). If you ever use a function that will likely return a binary string, then you probably need to set unicode.runtime_encoding. Quercus may convert your binary string to Unicode and then to your output encoding for output to the browser. If your runtime encoding is wrong, then you would see garbage in your

browser.

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">

  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <script-encoding>iso-8859-1</script-encoding>
      <php-ini>
        <unicode.runtime_encoding>iso-8859-1</unicode.runtime_encoding>
      </php-ini>
    </init>
  </servlet-mapping>

</web-app>
```

unicode.output_encoding is the charset used to display output to the browser. You can set it in your resin-web.xml:

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <script-encoding>iso-8859-1</script-encoding>
      <php-ini>
        <unicode.output_encoding>iso-8859-1</unicode.output_encoding>
        <unicode.runtime_encoding>iso-8859-1</unicode.runtime_encoding>
      </php-ini>
    </init>
  </servlet-mapping>
</web-app>
```

Compiling PHP Scripts for Increased Performance

Quercus will automatically compile PHP scripts into Java classes for better performance. This is available only in Resin Professional.

The default behaviour in Resin Professional is to execute the PHP script in interpreted mode, and to compile the script in the background. When the compiled version is ready, it is used instead of the interpreted version. To force compilation, use the <compile> tag within the <init> tag:

WEB-INF/resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <compile>true</compile>
```

```
    </init>
  </servlet-mapping>
</web-app>
```

Using Databases

JDBC drivers are required to use databases in Quercus. There are JDBC drivers for MySQL, Oracle, SQLite, and many other database engines. The desired JDBC driver should be downloaded into Resin's ${resin.root}/lib directory. Resin will automatically load jars in the lib directory upon startup.

DATABASE TYPE	URL FOR DOWNLOAD
MySQL	http://dev.mysql.com/downloads/connector/j
PostgreSQL	http://jdbc.postgresql.org/download.html
Oracle	http://www.oracle.com/technology/software/tech/java/sqlj_jdbc

The database support in Quercus supports robust database connection pooling since Quercus runs in Resin, a fast Java application server. All PHP database access automatically uses JDBC-pooled connections. PHP code does not need changing to take advantage of this capability.

The PHP database apis supported include PDO (portable database objects), mysql, mysql improved, postgres and oracle. Any JDBC-compliant database is available to PHP scripts using PDO.

PDO access to JNDI-configured databases

```
<php

$db = new PDO("java:comp/env/jdbc/my-database");

...
?>
```

JNDI DataSource

If a database with JNDI name jdbc/myDatabase is defined in resin.conf, (see Database Configuration), Quercus can do a JNDI lookup for the database when database functions are called. Thus, database connection parameters like user name can be omitted within PHP scripts. This allows easier maintenance and enables Java and PHP database settings to be centrally located in resin.conf.

Scripts can use the jndi name directly:

```
<?php

  // standard PHP
  //mysql_connect($host, $username, $password, $dbname);
```

```
  // using JNDI lookup
  mysql_connect("java:comp/env/jdbc/myDatabaseName");

?>
```

You can use a JNDI <database configuration in the WEB-INF/resin-web.xml to override the PHP connection code. If a <database> is provided, any mysql_connect call will return the configured database, ignoring the parameters to the mysql_connect call.

Example: overriding database configuration in resin-web.xml

```
<web-app xmlns="http://caucho.com/ns/resin">
  <database jndi-name="jdbc/mysql">
    <driver type="org.gjt.mm.mysql.Driver">
      <url>jdbc:mysql://localhost:3306/test</url>
      <user></user>
      <password></password>
    </driver>
  </database>

  <servlet-mapping url-pattern="*.php"
                   servlet-class="com.caucho.quercus.servlet.QuercusServlet">
    <init>
      <database>java:comp/env/jdbc/myDatabaseName</database>
    </init>
  </servlet-mapping>
</web-app>
```

Java/PHP integration

Using Java services from PHP: Resin-IoC/WebBeans

If you're already using Resin-IoC/WebBeans to organize your application into services, your PHP script can grab a reference to the service with the registered name. Calling java_bean with a singleton bean's name will return the singleton, and calling java_bean with a component's name will return a new instance of the component. Quercus will automatically marshal PHP calls to the bean's Java methods, so your service's entire functionality will be available to the PHP script.

Example: using a WebBeans service

```
<?php

  $house_manager = java_bean("houseManager");

  $house = $house_manager->findHouse("Gryffindor");

  foreach ($house->getPrefects() as $prefect) {
    echo $prefect . "\n";
  }

?>
```

WebBeans (JSR-299) is an inversion-of-control/dependency-injection framework specification for JavaEE 6 which is designed to organize Java services, and also integrate with scripting frameworks. WebBeans integrates tightly with the newest EJB and persistence specifications, so PHP applications using the WebBeans interface will gain the latest, cleanest integration with Java applications. Resin-IoC is Caucho's implementation of the WebBeans framework, and serves as the underlying architecture of Resin itself, as well as the Resin EJB implementation.

Working with Java classes in PHP

Instantiating objects by class name

An alternative to import is to use new Java(...) with the class name and any constructor arguments.

```
<?php

  $a = new Java("java.util.Date", 123);

  echo $a->time;

?>
```

Importing classes

Quercus supports the use of an import statement in PHP. import makes any java class available to the PHP script with it's unqualified name.

Example: creating a Java class with import

```
<?php

  import java.util.Date;

  $a = new Date(123);

  echo $a->time;
?>
```

User classes can be placed in the webapp's WEB-INF/classes directory.

Example: WEB-INF/classes/example/MyBean.java

```
package example;

public class MyBean
{
  int _value;

  public MyBean(int value)
  {
    _value = value;
  }

  public int getValue()
  {
    return _value;
  }

  public String makeMessage()
  {
    return "Hello, my value is " + _value;
  }
}
```

mybean.php

```
<?php

  import example.MyBean;

  $bean = new MyBean(123);

  var_dump($bean);
  var_dump($bean->value);
  var_dump($bean->makeMessage());
?>
```

The import keyword will also work on PHP classes but it has a different functionality than for Java classes. import will try to autoload PHP classes by including the file WEB-INF/classes/*classname*.php from the application's WEB-INF/classes directory.

Calling Java Methods

PHP syntax is used for invoking methods. PHP property syntax can be used for invoking getters and setters of Java objects.

```
<?php

  import java.util.Date;

  $a = new Date(123);
```

```
  echo $a->getTime();    # calls getTime()
  echo $a->setTime(456); # calls setTime(456)

  echo $a->time;      # calls getTime()
  $a->time = 456;     # calls setTime(456)
?>
```

Static members and methods

Static methods and members are available using PHP syntax if the Java class has been imported.

```
<?php

import java.util.Calendar;

$calendar = Calendar::getInstance();

var_dump($calendar);
?>
```

An alternative to import is to use java_class() to access static members and methods.

```
<?php

  $class = java_class("java.lang.System");

  # System.in
  $in = $class->in;

  # System.currentTimeInMillis();
  $time = $class->currentTimeInMillis();

?>
```

Java method overloading

Quercus allows overloaded Java methods to be called from within PHP code. The number of arguments is most important, followed by the argument types. Quercus will use the method whose arguments are the most easily marshaled (i.e. a PHP string easily goes into a Java String whereas a PHP array is a mismatch for a Java int).

Because the PHP language itself does not support overloading, the Quercus overloading of Java methods may not be exact. Therefore, it's best to keep overloading to a minimum. Overloading by the number of arguments will always work, but overloading by types is trickier.

Example: MyModule.java

```
import com.caucho.quercus.module.AbstractQuercusModule;

public class MyModule extends AbstractQuercusModule
{
  public static void foo(String a, boolean b)
  {
  }

  public static void foo(String a, String b)
  {
  }
}
```

example.php

```
<?php

  foo('abc', false);

?>
```

In the example above, the first Java method public static void foo(String a, boolean b) is called because it requires the least amount of type coercion.

Note Only Java methods with the same amount of arguments will be considered.

Modules: Adding PHP functions

The core PHP functions are implemented inside Quercus modules. Quercus modules are the Java equivalent of PHP modules.

All Quercus modules need to implement AbstractQuercusModule. Functions defined in your modules are callable from within PHP script by using just the function name. Function names need to be distinct in order to prevent name collisions, though Quercus does support function overloading (for Java functions only).

A typical Quercus module looks like:

WEB-INF/classes/example/HelloModule.java

```
package example;

import com.caucho.quercus.env.Env;
import com.caucho.quercus.module.AbstractQuercusModule;

public class HelloModule extends AbstractQuercusModule
{
  /**
   * @param env provides Quercus environment resources.
   * @param str
   */
  public void hello_test(Env env, String str)
  {
```

```
    // 'echos' the string
    env.println("hello " + str);
  }
}
```

example.php

```
<?php

  // PHP 5 is case-insensitive
  // just prints "hello me" to the browser.
  hello_test("me");

?>
```

For a tutorial on how to implement your own Quercus module, see the **Quercus module tutorial**.

Marshalling: PHP to Java conversions

PHP types

For every PHP type, there is a Java type that is used internally to represent the corresponding PHP value. All of the Java types extend Value.

PHP TYPE	QUERCUS CLASS
null	NullValue
string (php5)	StringBuilderValue
string (php6, binary)	BinaryBuilderValue
string (php6, unicode)	UnicodeBuilderValue
bool	BooleanValue
int	LongValue
float	DoubleValue
array	ArrayValue
object	ObjectValue
ref/var	Var

Java method arguments

In Quercus, Java methods can be called from within PHP. Java arguments for Java methods are marshaled to the correct type from the PHP parameters that were passed in.

When the Java argument type is declared to be Object, the value will be marshaled to a Java object. For example, a PHP int (LongValue) will be marshaled to an Integer. The only exceptions are PHP arrays and objects: they are passed in as-is without marshaling.

When the Java argument type is declared to be a Quercus Value, the PHP value is passed in directly without marshaling.

If the Java argument type is an object, passing in a PHP NULL will result in a null Java argument.

Java to PHP conversion

JAVA TYPE	PHP TYPE	QUERCUS TYPE
null	NULL	NullValue
boolean	bool	BooleanValue
Boolean	bool	BooleanValue
byte	int	LongValue
Byte	int	LongValue
short	int	LongValue
Short	int	LongValue
int	int	LongValue
Integer	int	LongValue
long	int	LongValue
Long	int	LongValue
float	float	DoubleValue
Float	float	DoubleValue
double	float	DoubleValue
Double	float	DoubleValue
String	string (php5)	StringBuilderValue
String	unicode (php6)	UnicodeBuilderValue
char	string (php5)	StringBuilderValue
char	unicode (php6)	UnicodeBuilderValue
Character	string (php5)	StringBuilderValue
Character	unicode (php6)	UnicodeBuilderValue
char[]	string (php5)	StringBuilderValue
char[]	unicode (php6)	UnicodeBuilderValue
byte[]	string (php5)	StringBuilderValue
byte[]	string (php6)	BinaryBuilderValue
Object[] (any other array)	array	ArrayValue
Calendar	effectively int (getTimeInMillis())	JavaValue
Date	effectively int (getTime())	JavaValue
URL	effectively string (toString())	JavaValue
Collection	array	JavaValue
List	array	JavaValue
Map	array	JavaValue
other Java Objects	object	JavaValue
Value	any	Value

Java objects like Calendar and Map are placed inside JavaValues and then returned to the PHP environment. A JavaValue is a wrapper that exposes the object's Java methods to PHP. For example, if $url is holding a Java URL object, then we can use $url->getHost() to call the URL's getHost() method.

Some Java objects may have an effective PHP value. Take for instance, Date. A Date object is, for practical purposes, a PHP int with it's value pegged to Date.getTime().

Collection, List, and Map behave just like PHP arrays. Suppose $map holds a Java HashMap, then it's certainly valid to do $map["foo"] = "bar". However, there are some limitations that are dependent on the underlying Java type. For example, $list[-1] = 5 will not be possible for a Java List because List indexes start at 0.

HttpServletRequest and HttpSession

QuercusServlet automatically creates the $request variable for PHP scripts. It contains the javax.servlet.http.HttpServletRequest object.

PHP sessions are not shared with servlet sessions. The $request variable can be used to obtain the servlet session if required.

$request->getSession(true) HttpSession

```
<?php
  $session = $request->getSession(true);

  $foo = $session->getAttribute("foo");
?>
```

PHP Module highlights

Standard modules

Quercus implements the standard PHP libraries (arrays, strings, date, regexp, etc). It also supports extension libraries like zip and zlib for compression, mcrypt for encryption, mail (implemented with JavaMail), and bcmath for large numbers.

APC (object caching)

For PHP object caching, Quercus implements the APC module. PHP applications can use the APC functions to save PHP objects without resorting to serialization and database persistence. Because Quercus runs in Resin, a Java web server, the saved objects are quickly available to any thread running PHP. In other words, unlike Apache which makes sharing across different PHP processes difficult, Quercus can just store a singleton cache of the APC-cached objects.

Because Quercus compiles PHP to Java code, PHP scripts get the opcode caching of APC for free. At this time, performance of Quercus is roughtly comparable with performance of mod_php with APC, i.e. it is significantly faster (3-5 times) than mod_php running by itself.

Image support ('gd')

Quercus provides the image module, so users can use image manipulation functions like watermarking and thumbnail generation in any PHP script on Quercus. .jpg, .png, and .gif files are currently supported. Java users will also find these libraries convenient.

PDF generation (PDFlib api)

PDF generation in Quercus follows the PDFlib API. Since the Quercus PDF implementation is a new implementation in Java, no special downloads are needed to use PDF.

AJAX (JSON)

Quercus also includes the JSON module for encoding and decoding AJAX-style requests. The JSON modules is an excellent example of the benefits of writing modules in Java. Because Java supports garbage collection and protects from pointer overruns, the JSON module implementation is straightforward and reliable, without having to worry about all the possible memory problem in a C library.

Gettext (localization)

Quercus supports the gettext API and .po and .mo files. gettext is a portable API for localization, i.e. translation of program messages. In the future, the Quercus gettext implementation will support Java message bundles so Java applications using PHP can use standard Java localization techniques.

ResinModule

jndi_lookup

Retrives an object from JNDI. jndi_lookup is useful in a SOA (Service Oriented Architecture) system to locate a Java service.

```
<?php

$conn = jndi_lookup("java:comp/env/jms/jms-connection-factory");
$queue = jndi_lookup("java:comp/env/jms/test-queue");

...
?>
```

mbean_explode

Explodes a JMX ObjectName into an array.

mbean_explode

```
<?php
```

```
var_dump(mbean_explode("resin:type=WebApp,name=/foo,Host=bar.com"));

?>
```

```
array(4) {
  [":domain:"]=>
  string(5) "resin"
  ["Host"]=>
  string(7) "bar.com"
  ["name"]=>
  string(4) "/foo"
  ["type"]=>
  string(6) "WebApp"
}
```

mbean_implode

Creates a JMX ObjectName from an array.

mbean_implode

```
<?php

$a = array(":domain:"=>"resin", "type" => "ThreadPool");

var_dump(mbean_implode($a));

?>
```

```
resin:type=ThreadPool
```

MBeanServer

An object representing a JMX MBeanServer.

```
<?php

$mbeanServer = new MBeanServer();

$threadPool = $mbeanServer->lookup("resin:type=ThreadPool");

echo "thread-max: " . $threadPool->threadMax;
```

lookup

Returns a proxy to the mbean matching the given name.

```
<?php
```

```
$mbeanServer = new MBeanServer();

$threadPool = $mbeanServer->lookup("resin:type=ThreadPool");
```

query

Returns mbean proxies matching the name pattern.

```
<?php

$mbeanServer = new MBeanServer();

foreach ($webApp in $mbeanServer->query("resin:type=WebApp,*")) {
  echo $webApp->name . "<br>\n";
}
```

resin_debug

Write debugging information to the log. The log is at INFO level.

resin_thread_dump

```
<?php

$a = array("a", "b");

resin_debug("ARRAY: $a[0]");

?>
```

resin_thread_dump

Produce a thread_dump to the logs. The log is at INFO level.

resin_thread_dump

```
<?php

$a = array("a"=>"b");

resin_thread_dump();

?>
```

resin_call_stack

Returns an array containing the current PHP function call stack.

resin_call_stack

```
<?php

function foo()
{
  bar();
}

function bar()
{
  var_dump(resin_call_stack());
}

foo();

?>
```

resin_var_dump

Produce a var_dump to the logs. The log is at INFO level.

resin_var_dump

```
<?php

$a = array("a"=>"b");

resin_var_dump($a);

?>
```

resin_version

Returns the version of Resin running Quercus.

```
<?php

var_dump(resin_version());

?>
```

xa_begin

Starts a distributed transaction. All database connections will automatically participate in the transaction. Returns TRUE for success, FALSE for failure.

```
<?php

xa_begin();
```

```
...

xa_commit();
?>
```

xa_commit

Commits a distributed transaction. All database connections will automatically participate in the transaction. Returns TRUE for success, FALSE for failure.

```
<?php

xa_begin();

...

xa_commit();
?>
```

xa_rollback

Rolls back a distributed transaction. All database connections will automatically participate in the transaction. Returns TRUE for success, FALSE for failure.

```
<?php

xa_begin();

...

xa_rollback();
?>
```

xa_rollback_only

Marks the current distributed transaction as rollback only. Subsequent attempts to commit the transaction will fail with a warning. Returns TRUE for success, FALSE for failure.

See Also

- Quercus home page
- Caucho Forums, including a Quercus forum
- Caucho Bug Tracker
- Caucho Mailing Lists

Technology.