# ENOVIA Live Collaboration™

V6R2009x

Programming Guide

# Copyright and Trademark Information

Dassault Systèmes Enovia Corp.
900 Chelmsford Street
Lowell, MA 01851

Telephone 978.442.2500
Email: enovia.info@3ds.com

http://www.3ds.com

## Additional Components

This product also includes additional components copyrighted by other third parties. The sections that follow provide license and copyright notices of these software components.

### Apache

### Apache Ant

Copyright © 1994-2001 World Wide Web Consortium, <a href="http://www.w3.org/">World Wide Web Consortium</a>, (<a href= "http://www.lcs.mit.edu/">Massachusetts Institute of Technology</a>, <a href="http://www.inria.fr/">Institut National de Recherche en Informatique et en Automatique</a>, <a href= "http://www.keio.ac.jp/">Keio University</a>). All Rights Reserved.

http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.

Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, a short notice of the following form (hypertext is preferred, text is permitted) should be used within the body of any redistributed or derivative code:

"Copyright © 1999-2004 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/"

Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

This license came from: http://www.megginson.com/SAX/copying.html. However please note future versions of SAX may be covered under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at http://www.saxproject.org/ for more up-to-date releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for XML), and release all of the SAX 2.0 source code, compiled code, and documentation contained in this distribution into the Public Domain. SAX comes with NO WARRANTY or guarantee of fitness for any purpose.

David Megginson, david@megginson.com

## Apache Axis

=========================================================================

NOTICE file corresponding to section 4(d) of the Apache License, Version 2.0, in this case for the Apache Axis distribution.

=========================================================================

This product includes software developed by The Apache Software Foundation (http://www.apache.org/).

## Apache Tomcat

[under Apache License, Version 2.0 above]

## Apache Servlet-API

[under Apache License, Version 2.0 above]

## FTP

Copyright (c) 1983, 1985, 1989, 1993, 1994

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
   This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAYOUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 1997-1999 The Stanford SRP Authentication Project

All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL STANFORD BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER

Copyright 1990 by the Massachusetts Institute of Technology.

All Rights Reserved.

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

## Getline

Copyright (C) 1991, 1992, 1993 by Chris Thewalt (thewalt@ce.berkeley.edu)

Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notices appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

## GifEncoder

GifEncoder - write out an image as a GIF

Transparency handling and variable bit size courtesy of Jack Palevich.

Copyright (C)1996,1998 by Jef Poskanzer <jef@acme.com>. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## ImageEncoder

ImageEncoder - abstract class for writing out an image

Copyright (C) 1996 by Jef Poskanzer <jef@acme.com>. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## JavaMail

Sun Microsystems, Inc.

Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. LICENSE TO USE. Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.

2. RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

3. LIMITED WARRANTY. Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.

4. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

## Jakarta POI

[under Apache License, Version 2.0 above]

## JDK

The following section contains a list of the files and directories that may optionally be omitted from redistributions of the JDK. All files not in these lists of optional files must be included in redistributions of the JDK.

Optional Files and Directories

The following files may be optionally excluded from redistributions. These files are located in the jdk1.5.0_<version> directory, where <version> is the update version number. Solaris and Linux filenames and separators are shown. Windows executables have the ".exe" suffix. Corresponding files with _g in name can also be excluded.

jre/lib/charsets.jar

Character conversion classes

jre/lib/ext/

sunjce_provider.jar - the SunJCE provider for Java Cryptography APIs

localedata.jar - contains many of the resources needed for non US English locales

ldapsec.jar - contains security features supported by the LDAP service provider

dnsns.jar - for the InetAddress wrapper of JNDI DNS provider

bin/rmid and jre/bin/rmid

Java RMI Activation System Daemon

bin/rmiregistry and jre/bin/rmiregistry

Java Remote Object Registry

bin/tnameserv and jre/bin/tnameserv

Java IDL Name Server

bin/keytool and jre/bin/keytool

Key and Certificate Management Tool

bin/kinit and jre/bin/kinit

Used to obtain and cache Kerberos ticket-granting tickets

bin/klist and jre/bin/klist

Kerberos display entries in credentials cache and keytab

bin/ktab and jre/bin/ktab

Kerberos key table manager

bin/policytool and jre/bin/policytool

Policy File Creation and Management Tool

bin/orbd and jre/bin/orbd

Object Request Broker Daemon

bin/servertool and jre/bin/servertool

Java IDL Server Tool

bin/javaws, jre/bin/javaws, jre/lib/javaws/ and jre/lib/javaws.jar

Java Web Start

src.zip

Archive of source files

Redistributable JDK Files

The limited set of files from the JDK listed below may be included in vendor redistributions of the J2SE Runtime Environment. They cannot be redistributed separately, and must accompany a JRE distribution. All paths are relative to the top-level directory of the JDK.

jre/lib/cmm/PYCC.pf

Color profile. This file is required only if one wishes to convert between the PYCC color space and another color space.

All .ttf font files in the jre/lib/fonts directory.

Note that the LucidaSansRegular.ttf font is already contained in the J2SE Runtime Environment, so there is no need to bring that file over from the JDK.

jre/lib/audio/soundbank.gm

This MIDI soundbank is present in the JDK, but it has been removed from the J2SE Runtime Environment in order to reduce the size of the Runtime Environment's download bundle. However, a soundbank file is necessary for MIDI playback, and therefore the JDK's soundbank.gm file may be included in redistributions of the Runtime Environment at the vendor's discretion. Several versions of enhanced MIDI soundbanks are available from the Java Sound web site: http://java.sun.com/products/java-media/sound/. These alternative soundbanks may be included in redistributions of the J2SE Runtime Environment.

The javac bytecode compiler, consisting of the following files:

bin/javac [Solaris(TM) Operating System and Linux]

bin/sparcv9/javac [Solaris Operating System (SPARC(R) Platform Edition)]

bin/amd64/javac [Solaris Operating System (AMD)]

bin/javac.exe [Microsoft Windows]

lib/tools.jar [All platforms]

The Annotation Processing Tool, consisting of the following files:

bin/apt [Solaris(TM) Operating System and Linux]

bin/sparcv9/apt [Solaris Operating System (SPARC(R) Platform Edition)]

bin/amd64/apt [Solaris Operating System (AMD)]

bin/apt.exe [Microsoft Windows]

jre\bin\server\

On Microsoft Windows platforms, the JDK includes both the Java HotSpot Server VM and Java HotSpot Client VM. However, the J2SE Runtime Environment for Microsoft Windows platforms includes only the Java HotSpot Client VM. Those wishing to use the Java HotSpot Server VM with the J2SE Runtime Environment may copy the JDK's jre\bin\server folder to a bin\server directory in the J2SE Runtime Environment. Software vendors may redistribute the Java HotSpot Server VM with their redistributions of the J2SE Runtime Environment.

Unlimited Strength Java Cryptography Extension

Due to import control restrictions for some countries, the Java Cryptography Extension (JCE) policy files shipped with the J2SE Development Kit and the J2SE Runtime Environment allow strong but limited cryptography to be used. These files are located at

<java-home>/lib/security/local_policy.jar

<java-home>/lib/security/US_export_policy.jar

where <java-home> is the jre directory of the JDK or the top-level directory of the J2SE Runtime Environment.

An unlimited strength version of these files indicating no restrictions on cryptographic strengths is available on the JDK web site for those living in eligible countries. Those living in eligible countries may download the unlimited strength version and replace the strong cryptography jar files with the unlimited strength files.

jconsole

jconsole.jar

jconsole may be redistributed outside the JDK but only with Sun's JRE.

Endorsed Standards Override Mechanism

An endorsed standard is a Java API defined through a standards process other than the Java Community ProcessSM (JCPSM). Because endorsed standards are defined outside the JCP, it is anticipated that such standards will be revised between releases of the Java 2 Platform. In order to take advantage of new revisions to endorsed standards, developers and software vendors may use the Endorsed Standards Override Mechanism to provide newer versions of an endorsed standard than those included in the Java 2 Platform as released by Sun Microsystems.

For more information on the Endorsed Standards Override Mechanism, including the list of platform packages that it may be used to override, see

http://java.sun.com/j2se/1.5.0/docs/guide/standards/

Classes in the packages listed on that web page may be replaced only by classes implementing a more recent version of the API as defined by the appropriate standards body.

In addition to the packages listed in the document at the above URL, which are part of the Java 2 Platform Standard Edition (J2SETM) specification, redistributors of Sun's J2SE Reference Implementation are allowed to override classes whose sole purpose is to implement the functionality provided by public APIs defined in these Endorsed Standards packages. Redistributors may also override classes in the org.w3c.dom.* packages, or other classes whose sole purpose is to implement these APIs.

The cacerts Certificates File

Root CA certificates may be added to or removed from the J2SE certificate file located at <java-home>/lib/security/cacerts. For more information, see The cacerts Certificates File section in the keytool documentation.

Web Pages

For additional information, refer to these Sun Microsystems pages on the World Wide Web:

http://java.sun.com/

The Java Software web site, with the latest information on Java technology, product information, news, and features.

http://java.sun.com/docs

Java Platform Documentation provides access to white papers, the Java Tutorial and other documents.

http://developer.java.sun.com

Developer Services web site. (Free registration required.) Additional technical information, news, and features; user forums; support information, and much more.

http://java.sun.com/products/

Java Technology Products & API

--------------------------------------------------------------------------------

## JDOM

## Krypto

## OpenLDAP

## OpenSSL

## Oracle

## SSLUtils

## Sun RPC

## Tcl

## Xalan

[under Apache License, Version 2.0 above]

## Xerces

[under Apache License, Version 2.0 above]

## Xerces2

[under Apache License, Version 2.0 above]

# Table of Contents

1

# Java Program Objects

## Introduction to JPOs

This chapter describes how to develop Java Program Objects (JPOs) that adhere to "best practices" and facilitate migration to future releases of ENOVIA products. It attempts to:

- Explain the details of JPOs, and their proper role in Applications.
- Provide guidelines to ensure JPO consistency across all Applications.
- Describe techniques for developing, debugging, and invoking JPOs.

The introduction of the Java Program Object (JPO) is a dramatic move from Tcl into the world of Java. JPOs allow developers to write Java programs on top of the ENOVIA Live Collaboration Studio Customization Toolkit programming interface and have those programs invoked on the fly. When running inside the ENOVIA Live Collaboration Kernel, the programs share the same context and transaction as the thread from which they were launched. In fact, the Java programs are running inside the same Java Virtual Machine (JVM) as the kernel. JPOs are not supported on platforms not offering a JVM.

While Tcl continues to be supported and does offer a scripting approach which can make sense in some cases, the Java language brings several advantages over Tcl:

- compiled, and therefore greater run-time performance
- object-oriented
- thread safe

JPOs are just another type of Program Object that have code written in the Java language. Just about anywhere a Program Object can be used, a JPO can be used. (To launch a JPO from the command bar or as a method on a Business Type, a Tcl wrapper needs to be written. More on this in the *Invoking Methods* section.)

Java code must be contained in a class. This class will make up the code of a JPO. However, simply translating Tcl Program Objects into Java is not the goal. This would lead to lots of small classes, each containing a single method. Also, many Tcl Programs would gain little in performance and actually be difficult to convert if they rely heavily on macros.

The very object-oriented nature of Java lends itself to encapsulating multiple methods into a single class. The goal is to encapsulate common code into a single class. A good test that code is grouped properly is to verify that the code is tightly cohesive (logically belongs together) and loosely coupled (not overly dependent on code living elsewhere). This encapsulation will greatly reduce the number of program objects in the database and simplify the maintenance of business logic.

## A note about scalability and performance

The issue of scalability comes down to the specific schema and functionality of the implementation. An implementation may be able to support 4,000 concurrent users on a certain hardware/software configuration, but on that same configuration they might not be able to support another custom application with 500 concurrent users. Use of Web Navigator mean that the schema, the triggers, etc. are all custom and will scale at a different rate than any other ENOVIA Live Collaboration implementation.

The approach to take is to find out what in the specific application uses a lot of resource and re-architect it to be either more efficient, or offload it to a background process on another server. If 500 users are utilizing 100% of the CPU, then either more CPU is needed or the application needs to be more scalable. Inefficient use of APIs, complex triggers that need to be executed frequently, and inefficient data modeling are common sources of scalability issues. Database tuning including adding additional indices, is an effective technique to improve performance and scalability simultaneously. Code and design review specifically with an eye toward scalability is the recommended approach.

## Specifying the Classpath

Whenever working with the Java language, establishing the proper CLASSPATH is important for both compiling and running the program. At runtime, the application server needs to know about all necessary .jar files in its implementation of a classpath. The installation programs for the ENOVIA Live Collaboration server configure a variable called MX_CLASSPATH that is added to the environment's CLASSPATH to compute it at runtime. For compiling, add MX_CLASSPATH to your environment, ensuring that it includes references to all necessary .jar file. Refer to MX_CLASSPATH in the *ENOVIA Live Collaboration Installation Guide* for details.

*The first step in debugging any Java program is to check that the CLASSPATH used by the server includes all necessary .jar files.*

# Java Program Objects

A Java Program Object (JPO) is just another type of Program object that contains code written in the Java language. Generally, anywhere a Program object can be used, a JPO can be used.

JPOs have been created to satisfy a need for the ability to run Java programs natively inside the kernel, without creating a separate process and with a true object-oriented integration 'feel' as opposed to working in MQL. JPOs allow developers to write Java programs on top of the ENOVIA Live Collaboration Studio Customization Toolkit programming interface and have those programs invoked dynamically.

When running inside the ENOVIA Live Collaboration Kernel, the programs share the same context and transaction as the thread from which they were launched. In fact, the Java programs are running inside the same Java Virtual Machine (JVM) as the kernel.

Java Program Objects are also tightly integrated with the existing scripting facilities of ENOVIA Live Collaboration. Developers can seamlessly combine MQL, Tcl, and Java to implement their business model.

Java code must be contained in a class. In general, a single class will make up the code of a JPO. However, simply translating Tcl program objects into Java is not the goal. This would lead to many small classes, each containing a single method. The very object-oriented nature of Java lends itself to encapsulating multiple methods into a single class. The goal is to encapsulate common code into a single class.

A JPO can be written to provide any logical set of utilities. For example, there might be a JPO that provides access to Administration objects that are not available in the Studio Customization Toolkit. But for the most part, a JPO will be associated with a particular Business Type. The name of the JPO should contain the name of the Business Type (but this is certainly not required).

It is the responsibility of the JPO programmer to manually create the JPO and write all of the methods inside the JPO. Keep in mind that JPO code should be considered server-side Java; no Java GUI components should ever be constructed in a JPO.

## Separating Interface From Implementation

The Java code in a JPO should be thought of as server-side (back-end) implementation code. A JPO can be written to provide any logical set of utilities. For example, there might be a JPO that provides access to Administration objects that are not available in the Studio Customization Toolkit. But for the most part, a JPO is associated with a particular Business Type. The name of the JPO should contain the name of the Business Type (but this is certainly not required). Access to a JPO should be done through an interface bean. More on this interface layer will be given in a later section.

Keep in mind that JPO code should be considered server-side Java. This means that certainly no Java GUI components should ever be constructed in a JPO. Otherwise, the full richness of Java is at the disposal of the JPO programmer. Details of the methods to write and their coding and naming conventions follow.

## Applications

JPOs are ideally suited for use inside applications. The use of JPOs in applications have the combined effect of moving Java code off the JSPs (away from the View and into the

Model, if one thinks of the Model-View-Control design) and executing it on the appropriate application server (back-end) tier for optimum performance. JavaBeans can be used on the Web server (middle) tier by JSPs as the interface to the JPOs (more on this later).

The Framework (BPS) and the applications that sit on top of the Framework ship with JPOs defined for most Business Types in the schema. Their style adheres to the guidelines found in this guide. These JPOs will serve as examples to work from when writing your own JPO. It is also hoped that extensions to the Framework and applications will utilize inheritance from these JPOs. Each JPO is shipped as a combination of a base class and a derived class. All “out of the box” (OOTB) business logic is placed in the base class. The derived class is available for extensions to the business logic. Since the system always refers to the derived class, custom extensions are picked up without a need for changing OOTB logic. For example, a Business Type named Project would be represented by the 2 JPOs named emxProjectBase and emxProject (which extends emxProjectBase). Any custom business logic should be placed in emxProject JPO.This form of extensions to an application is less susceptible to problems associated with installing a new version of the application.

To further exploit the power of inheritance, the Framework ships with 2 JPOs that are to serve as the base class for all type JPOs (business object and relationship) in the schema. The JPO emxDomainObject should the be base class for each JPO representing a business object type. The JPO emxDomainRelationship should be the base class for each JPO representing a relationship type.

## Simple Example

Before going into the details of how best to write JPOs, it makes sense to present a simple example. Note, this example does not represent a JPO that meets the standards presented later.

Run the Business Modeler component of the ENOVIA Studio Modeling Platform as a Business Administrator and place the following code into a new Program Object named “Hello World”. Mark the Program Object as type “Java.”

```
import matrix.db.*;
import java.io.*;

public class ${CLASSNAME}
{
  public  ${CLASSNAME}  () {

  }
  public ${CLASSNAME} (Context context, String[] args) throws
Exception
  {
  }
public int mxMain(Context context, String []args)  throws
Exception
 {
  BufferedWriter writer = new BufferedWriter(new
MatrixWriter(context));
  writer.write("Hello World!\n");
  writer.flush();
  return 0;
  }
}
```

There are several ways to now invoke this new JPO (see *Invoking Methods*). In this example, an instance of the class “Hello World”* is instantiated using the constructor (Context, string[]). The Context object is constructed by the kernel to allow the program to share the transactional context of the launching thread. If desired, the constructor can be implemented to connect to an ENOVIA Live Collaboration Kernel elsewhere on the network. In this case, however, the program will run on a different thread, and inside its own transaction boundary.

[*Actually, the resulting class name will be modified to adhere to the Java language rules for naming classes. This process is referred to as *name mangling* and is discussed in later sections.]

### MatrixWriter

The class matrix.db.MatrixWriter is available to JPOs for sending output to the console of the thread from which they were launched. For instance, if invoked from a Tcl program, the output would be captured and returned as the results from the Tcl/MQL command.

```
tcl;
set results [mql exec prog “Hello World”]
# the next line should display “Hello World!”
```

```
puts $results
```

The MatrixWriter class is useful for seamlessly replacing existing Tcl programs with JPOs.

## MatrixLogWriter

The class matrix.db.MatrixLogWriter is available to JPOs for sending output to the log file found in the *logs* directory of the ENOVIA Collaboration Live Collaboration server home. This can be useful for debugging, as well as any needs for general purpose logging. For example, if one were to replace MatrixWriter with MatrixLogWriter in the “Hello World” example, the string “Hello World!” would appear in the log file.

## Name Macros

There are two problems with using schema object names for the JPO class. The first problem is that ENOVIA Live Collaboration schema object names are allowed to contain spaces and other non-alpha characters that would cause the Java compiler to complain. The second problem is dynamic changes to the schema object names would not automatically be made to the code within JPOs.

Special macros are used to map the schema object names to legal Java-compliant names (a process known as *name mangling*). These macros are appropriately modified automatically when a schema object name is changed. There are two types of special macros.

The first macro is used to refer to the JPO name in the Java class definition. In this case the macro is

```
${CLASSNAME}
```

A similar macro is also used to refer to other JPO classes inside the code of a JPO, for example when extending a base class or calling a method in another JPO. In these cases the macro is

```
${CLASS:jpo_name}
```

where `jpo_name` is the name of the JPO holding the class of interest. Use of these parameterized class names allows both dynamic loading of classes after the source has been changed without restarting the system, and automatic generation of dependencies between multiple Java Program Objects.

For example, if there is JPO named “emxRADProject” and its corresponding Business Type RADProject derives from Project (which has a corresponding JPO named “emxProject”), then the Java code defining the class for RADProject would look something like:

```
public class ${CLASSNAME} extends ${CLASS:emxProject}
{
:
}
```

During compilation (and extraction), the macro `${CLASSNAME}` is mangled into a name that includes the JPO name “emxRADProject” (plus a suffix of “_mxJPO” followed by a hashed encoding) and the macro `${CLASS:emxProject}` is mangled into the exact same name as it was for `${CLASSNAME}` appearing in the emxProject JPO. In other words, the system does all the appropriate name mangling and the JPO programmer need only be concerned with the higher-level macros.

For example, the “Hello World” JPO ends up with a class name something like `Hello_World_mxJPOtAXMTwEAAAAFAAAA` (the hashed encoding at the end will vary).

## Coding Conventions

This section describes necessary use of commenting and how some features of the Java language are to be used (including those features which are off-limits).

### Javadoc

The class itself, and each method in the class, must be documented using javadoc comments. Each method will be preceded by javadoc comments that describe its purpose, input and output, exceptions thrown, where used, and the release it was introduced (and where appropriate, the release it was deprecated).

It is mandated that comments be meaningful and not just stating the obvious. The comments should clearly state the role of a class and how to use it in practice. Similarly for methods. The resulting document from the javadoc comments should provide enough detail so that a programmer knows how to use a class and its methods appropriately.

For example, if a JPO named emxRADProject is associated with a Business Type RADProject which derives from Project, the class would be preceded by the following javadoc comments:

```
/**
 * This class represents the Business Type RADProject which
derives from
 * Project. A RADProject is a specialized Project associated
with Rapid
 * Application Development… <lots more text goes here>
 * @since AEF 9.0.0
 */
public class ${CLASSPATH} extends ${CLASS:emxProject}
{
:
}
```

A method belonging to this class might look like:

```
/**
 * Get the default file format for this RADProject. Default
formats are used
 * during file checkin when no explicit format is given.
 *
 * @param context the Matrix <code>Context</code> object
 * @param args not used by this method
 * @return the default format for this RADProject
 * throws exception if the operation fails
 * @since AEF 9.0.0
 */
public String getDefaultFormat(Context context, String [] args)
throws Exception
{
:
}
```

## Mandatory Methods

Each JPO must have certain methods defined. The following subsections list each of the methods. In general, all methods take a first argument of type Context.

### Constructors

Each JPO should contain a public constructor that takes a Context object and a String array. The String array holds any arguments that need to be passed to the constructor, such as an objectid.

```
/**
 * Create a new RADProject object from a given id
 *
 * @param context the Matrix <code>Context</code>object
 * @param args hols the following input arguments:
   *   0 - Strign holding “objectId”
 * @return a RADProject
 * throws exception if the operation fails
 * @since AEF 9.0.0
 */
public ${CLASSNAME} (Context context, String[] args) throws
Exception
{
:
}
```

### Main Entry Point

Each JPO must contain a public method named mxMain that takes a Context object and a String array, and returns an int. This method is called by default when no method name is explicitly specified.

```
/**
 * Main entry point
 *
 * @param context the Matrix <code>Context</code>object
 * @param args array of input argument strings
 * @return an integer status code (0 = success)
 * throws exception if the operation fails
 * @since AEF 9.0.0
 */
public int mxMain (Context context, String[] args) throws
Exception
{
:
}
```

## Triggers

For each trigger in the Business Type there can be a corresponding public method containing the business logic. These methods return an integer value used to control the handling of events (such as blocking or overriding events).

Regardless of the naming convention you choose, you must always register the JPO method with the trigger event using Business Modeler or the Framework Trigger Manager. To associate a JPO method as a trigger program, use the JPO program name as the trigger program name and give as input arguments:

```
-method METHOD_NAME -construct ${OBJECTID}
```

Where `METHOD_NAME` is the name of the method in the JPO.

Follow this with any additional arguments you want to pass into the JPO method.

## Invoking Methods

This section presents details on how to invoke JPO methods.

All JPO methods invoked from MQL or the Studio Customization Toolkit must adhere to two signatures. These signatures are:

```
int (Context, String [])
```

and

```
Object (Context, String [])
```

Where any derivation of Object can be substituted for "Object". Any methods not called from MQL or the Studio Customization Toolkit can have any signature. So in general, some methods will be written to the signatures given above and then these methods will process the incoming argument list and call on other methods using signatures that are specific to the task being accomplished.

### Transaction Scope

JPO constructed objects live until the current transaction is ended. This means that a JPO constructor need only be called once inside a transaction. Then other methods of the JPO can be executed off the cached object.

## Returning Integers from a JPO

JPOs should not use int or Integer to return a value, since ENOVIA Live Collaboration interprets both of these data types as a status. To return a number in a JPO, define the method with a String return type. In the JPO, pack the number in the String. Then in the calling program (i.e. jsp) convert this string back to a number.

JPOs can be used to calculate values and might be called during a business object select operation. JPOs are commonly used this way in the applications. For example:

print bus TNR select program[prog_name] -method [method_name]

will call the the method 'method_name' in the JPO 'prog_name'. If the return value is something other than an int or Integer, the system will call the toString method on that object and assign its value to an RPE variable named 'prog_name'. If the method returns an int or Integer then the JPO must set the RPE variable itself. The value stored in the RPE variable is then returned from the select.

## Studio Customization Toolkit

Access to JPOs through the Studio Customization Toolkit consists of two invoke methods on a new JPO class. One form of the invoke method returns an int (useful for returning exit codes).

```
int ret = JPO.invoke(Context context, String className, String[]
initargs,
                                   String methodName, String[]
methodargs);
```

The other form returns a Java Object.

```
Object ret = JPO.invoke(Context context, String className,
String[] initargs,
                                        String methodName,
String[] methodargs,
                                     java.lang.Class retType);
```

An example of an object method in the “emxProject” JPO would be:

```
import matrix.db.*;
public class ${CLASSNAME} {
  public ${CLASSNAME} (Context context, String []args) {
}

public BusinessObjectList query(Context context, String []args)
{
Query query = new Query(args[0]);
    return query.evaluate(context);
  }
}
```

called by the following in a .jsp:

```
String[] init = new String[] {};
String[] args = new String[] { ".finder" };
BusinessObjectList list =
(BusinessObjectList)JPO.invoke(context, "Project",
                           init, "query", args,
BusinessObjectList.class);
```

*Note the use of casting, which is necessary because the system only knows to return an object instance. In the example above you see that BusinessObjectList is consistently used for declaring the object to hold the return object, for casting, and for the sixth argument that defines the return type.*

Working with Java Objects is very powerful as the previous example showed. But what about passing Objects into a JPO method? The JPO class has two methods that correspond to serializing an Object into a String, and de-serializing a String back into an Object. The method names are `packArgs` and `unpackArgs`, respectively. Since JPOs only accept strings as arguments, you must use the `packArgs/unpackArgs` methods to convert Java objects to/from strings when using JPOs.

These methods actually work with a String array of size two, that holds the class in the first slot and the serialized object in the second slot. This allows users of the String array to do proper type casting. One can build up several Objects in a String array by calling `packArgs()` several times using the proper indexing into the String array. However, it is better to place all arguments in a single compound object (like a hashmap) and then pack just the single compound object into the args array. Keep in mind that any Object needing to be passed as an argument in this way must implement the java.io.Serializable interface.

---

*You can only call* `packargs` *on objects that are serialized. In releases before 9.5.3.0, many of the Studio Customization Toolkit classes were not. Most classes are now serialized and so may be passed as arguments to JPOs.*

---

The `packArgs` method has the following signature:

```
public static String[] packArgs(Object in) throws Exception
```

The `unpackArgs` method has the following signature:

```
public static Object unpackArgs(String[] in) throws Exception
```

The following packing logic is placed in a bean that is called from a JSP.

```
    /**
     * Sends an icon mail message to the specified users.
     *
     * @param context the Matrix <code>Context</code> object
     * @param toList the to list of users to notify
     * @param ccList the cc list of users to notify
     * @param bccList the bcc list of users to notify
     * @param subject the notification subject
     * @param message the notification message
     * @param objectIdList the list of objects to send with the 
notification
     * @throws FrameworkException if the operation fails
     * @since AEF 9.5.0.0
     * @grade 0
     */
     public static void sendMessage(Context context,
                                    StringList toList,
                                    StringList ccList,
                                    StringList bccList,
                                    String subject,
                                    String message,
                                    StringList objectIdList)
        throws FrameworkException
    {
        try
        {
            ContextUtil.pushContext(context);

            // Create the arguments for the notification.
            Map note = new HashMap();
            note.put("toList", toList);
            note.put("ccList", ccList);
            note.put("bccList", bccList);
            note.put("subject", subject);
```

```
                note.put("message", message);
                note.put("objectIdList", objectIdList);

                // Pack arguments into string array.
                String[] args = JPO.packArgs(note);

                // Call the jpo to send the message.
                JPO.invoke(context, "emxMailUtil", null,
"sendMessage", args);
            }
            catch (Exception e)
            {
                throw (new FrameworkException(e));
            }
            finally
            {
                ContextUtil.popContext(context);
            }
        }
```

The unpacking logic is then found in the JPO method where it is performed prior to passing the arguments on to the worker method (this design pattern is very useful for hiding the details of packing and unpacking arguments).

```
        /**
         * Sends an icon mail notification to the specified users.
         *
         * @param context the Matrix <code>Context</code> object
         * @param args contains a Map with the following entries:
         *        toList - the list of users to notify
         *        ccList - the list of users to cc
         *        bccList - the list of users to bcc
         *        subject - the notification subject
         *        message - the notification message
         *        objectIdList - the ids of objects to send with the
notification
         * @returns nothing
         * @throws Exception if the operation fails
         * @since AEF 9.5.0.0
         */
        public static int sendMessage(Context context, String[]
args)
            throws Exception
        {
            if (args == null || args.length < 1)
            {
                throw (new IllegalArgumentException());
            }
            Map map = (Map) JPO.unpackArgs(args);

            sendMessage(context,
                        (StringList) map.get("toList"),
                              (StringList) map.get("ccList"),
                              (StringList) map.get("bccList"),
                              (String) map.get("subject"),
                              (String) map.get("message"),
                              (StringList) map.get("objectIdList"));

        return 0;
    }
```

## Runtime Program Environment

The Runtime Program Environment (RPE) is accessible to JPO methods using the Studio Customization Toolkit MQLCommand object. The various get and set environment commands can be passed and the results received back. The RPE is not dependable in the thin client architecture of ENOVIA products. It is recommended that the RPE be avoided in this environment.

## Shortcomings

It will be necessary to continue to write individual Tcl Program objects for each method of a Business Type. However, these individual Tcl Program objects should just be wrappers that call the appropriate JPO method. Likewise, Tcl Program wrappers should be used on the command bar to call on JPO methods.

These Tcl wrappers can utilize macros which are then passed to the JPO method as arguments. This is necessary since JPOs are compiled and therefore lose any use of macro substitution. So the Tcl wrapper gets the appropriate macro values at runtime and passes them along as arguments to the JPO method.

## Interface Layer

To better insulate JSPs from the use of JPOs, an interface layer must be provided. These light-weight wrappers hide Studio Customization Toolkit details from the JSP programmer and provide a more natural interface to the business logic contained in JPOs (considered the implementation layer). These wrappers are also able to hold state across multiple pages of an application and reduce the number of calls to JPOs executing on the backend application server.

There are several approaches to wrapping JPOs: JavaBeans, custom tags, simple Java classes, or JSP includes. ENOVIA recommends the use of JavaBeans as the best solution.

### JavaBeans

JavaBeans are a nice way to encapsulate business logic that might otherwise appear on a JSP. Our ultimate goal is to keep the bean interface thin and push the business logic all the way back to the application server tier in the form of a JPO. The idea is to have the lightweight bean (living on the Web tier) call on the JPO to do all the work and then hold the resulting information (state) to be served up as needed to JSPs. JavaBeans are simply Java classes that support introspection. They are easy to use within a JSP. The `jsp:useBean` associates a JavaBean with the JSP and ensures that the object is available for the scope specified in the tag. The bound object can be referenced using the associated id from one or more JSPs (depending on the scope). The tag syntax is as follows:

```
<jsp:useBean id=”name” scope=”page|request|session|application”
beandetails />
```

Where *beandetails* is one of:

```
class="className"
class="className" type="typeName"
beanName="beanName" type="typeName"
type="typeName"
```

Returning to our “Hello World” example in the early part of this document, if we assume we have written a HelloWorld bean to act as the interface to our JPO, we might find the following usage on a JSP:

```
<jsp:useBean id=”helloBean” scope=”session” class=”HelloWorld”
/>
<html>
<body>
<%
  helloBean.hello();
%>
</body>
</html>
```

This shows a HelloWorld bean being defined and given an id of “helloBean”. The `hello()` method simply calls on our JPO to generate the “Hello World!” text.

```
public class HelloWorld implements Serializable
{
  public HelloWorld ()
  {
  }

  public void hello()
  {
  String[] init = new String[] {};
  String[] args = new String[] {};
  // establish a context … <details not shown>
  int status = JPO.invoke(context, "Hello World”, init,
“mxMain", args);
  }
}
```

## Getting User/ Password Information

The context.getUser() and context.getPassword()  methods always give the current user and password, or empty strings if there is none. For all validate password triggers (check, override, or action), the username, password, and vault are provided as the first, second and third arguments to the trigger program, JPO or Tcl.

## Round-trip Engineering of JPOs

The goal is to support true *round-trip* engineering. This would mean that JPO code could freely move in and out of ENOVIA Live Collaboration from an external IDE. The benefits of such an environment are obvious and include: better editing, cross-referencing, visual modeling, and debugging.

### Compiling

ENOVIA Live Collaboration is set up so that invoking JPOs will cause an automatic compile if necessary, using Java directly.

Two ini variables are related to compiling JPOs:

MX_JAVAC_FLAGS allows you to pass extra flags to the compiler.

MX_JAVA_DEBUG, when set to `true`, compiles JPOs with debugging information available and sets up MQL so that a debugger program can be attached to it.

To force compilation before invoking a JPO method, use the MQL command `compile program`. This is useful for bulk compiling and testing for compile errors after an iterative change to the JPO source.

```
compile program PATTERN [force] [update] [COMPILER_FLAGS];
```

When a JPO is compiled or executed, any other JPOs which are called by that JPO or call that JPO must be available in their most recent version. The compile command includes an update option which will update the requested JPO’s dependencies on other JPOs that may have been added, deleted, or modified. Use the following command to ensure all JPOs have up-to-date dependencies:

```
<mql> compile prog * force update;
```

The class file resulting from the compile command is placed into the ENOVIA Live Collaboration database. A special class loader finds the class file in the database when referenced.

When a password trigger uses a JPO, the program must be compiled explicitly before it can be used, since there is no active context before a login transaction is committed. Business Administrators must compile the program when initially written and whenever modifications are made. Include the force option when you need to recompile a JPO. In order to recompile the JPO(s) used by password trigger, the Business Administrator can set `MX_PASSWORD_TRIGGERS = false` in matrix.ini and log in through MQL.

#### Deprecation warnings when compiling JPOs

With Version 10, `MX_JAVAC_OPTIONS` is set to `-nowarn`, which suppresses deprecation warnings when compiling JPOs.

If you want to see deprecation warnings, for example, when debugging:

- For automatic compiles, you can set `MX_JAVAC_OPTIONS` to `-deprecation`.
- For compiles using the command line, include the `-deprecation` flag and precede the compile command with `verbose on`, for example:

```
verbose on
compile prog NAME -deprecation
```

## Editing

From the Business Modeler, an external editor of choice can be launched from the Program object dialog. The matrix.ini variable MX_EDITOR is used to label the button, and the variable MX_EDITOR_PATH is used to specify the path to the external editor. Once specified, the dialog will include an additional button used to launch the editor. All changes made to the code will automatically be read back into the system. As a simple example, on Windows the settings might be:

```
MX_EDITOR=notepad
MX_EDITOR_PATH=c:\winnt\system32\notepad.exe
```

Another approach is to extract the Java code out of the ENOVIA Live Collaboration database and into a file directory. This can be done using the `extract program` MQL command. The resulting code will be ready for compilation (name mangling will already have been performed) and further development.

Some may choose to do all, or most of their editing in an offline IDE. The only requirement is that JPO names must follow the Java language naming convention. While editing outside the ENOVIA Live Collaboration system, the names of JPO classes can be referenced in a semi-mangled form. This form consists of the JPO name followed by the string "_mxJPO". For example, if editing code that needs to instantiate an object defined by the JPO "emxProject", the following would be used:

```
emxProject_mxJPO proj = new emxProject_mxJPO();
```

JPOs in the ENOVIA Live Collaboration can be populated with external files using the `insert program` MQL command. This command converts mangled names back into the appropriate special macro form. The previous example would be changed to:

```
${CLASS:emxProject} proj = new ${CLASS:emxProject}();
```

## Extracting

Extracting the Java source in the form of a file out to a directory is useful for working in an IDE. While in the IDE a user can edit, compile, and debug code. The `extract program` MQL command processes any special macros and generates a file containing the Java source of the JPO. If no source directory is given, the system uses ENOVIA_INSTALL/studio/java/custom (which is added to `MX_CLASSPATH` automatically by the install program).

```
extract program PATTERN [source DIRECTORY] [demangle]
```

*In order to use the extract feature, the JPO name must follow the Java language naming convention (i.e., no spaces, special characters, etc.). Only alphanumeric printable characters as well as '.' and '_' are allowed in class names in the JPO.*

The demangle option causes the file and class names to be normalized during the extract. For example, an extract of the HelloWorld JPO by default would create a file named HelloWorld_mxJPOtAXMTwEAAAFAAAA.java containing a class named HelloWorld_mxJPOtAXMTwEAAAFAAAA. When using the demangle option, the extracted file will be named HelloWorld_mxJPO.java containing a class named HelloWorld_mxJPO. This ability is needed by some IDEs that do not deal well with the hash value added to the file and class names.

Program object names may include a package specification, such as `enovia.java.custom.testjpo`. In this example, `enovia.java.custom` is the package, and `testjpo` is the name of the class. In this case, the code of the JPO does not need to include a package statement; it is added to the code if necessary during compilation or extraction. In addition, when programs are extracted, the source is placed in an appropriate subdirectory based on the name of the JPO; for example, `ENOVIA_INSTALL/studio/java/custom/testjpo.java`.

## Inserting

After testing and modifying Java source in an IDE, it is necessary to insert the code back into JPOs in the ENOVIA Live Collaboration database. The `insert program` MQL command regenerates special macros in the Java source as it is placed back into a JPO (reverse name-mangling - see *Name Macros*). If the JPO does not exist, the `insert` command creates it automatically.

```
insert program FILENAME | DIRECTORY;
```

For example:

```
insert program ENOVIA_INSTALL/studio/java/custom/
testjpo_mxJPO.java
```

OR

```
insert program ENOVIA_INSTALL/studio/java/custom/
```

The later will insert all the .java files in the specified directory.

When programs are inserted, the package statement is stripped from the source code, and the package name is prepended to the name of the class to create the program object; in the above example, it will be named `ENOVIA_INSTALL.studio.java.custom.testjpo`.

*In order to use the insert feature, the JPO source must follow the Java language naming convention for class code (i.e., no spaces, special characters, etc.) and also include "`_mxJPO`". All characters including and after "`_mxJPO`" are removed during insertion (refer to Name Macros.)*

### Extracting and Inserting JPOs

Macro processing is used by all program objects (JPO as well as Tcl) before they are run to allow value replacement for a macro in a program. In the resulting code the number of backslashes is halved when it finds 2 in a row.

The extract command calls the macro processing code, so the files it produces can be executed by Java outside of ENOVIA Live Collaboration. This means that the insert command also needs to take macro processing into account. Therefore, (since version 10.5.1) the insert command doubles up any backslashes in JPO code going into the database. For example:

1. If a JPO is written with a command that has 3 backslashes, when it is executed within ENOVIA Live Collaboration it goes to Java with 2 backslashes.
2. If the same program is extracted, the command now includes 2 backslashes.
3. If this same program is then inserted into ENOVIA Live Collaboration, it gets 4 backslashes (which provides the same code to Java as if it had 3 backslashes.)

## Debugging

JPOs can be debugged using an external IDE. As stated earlier, you must set the .ini variable MX_JAVA_DEBUG to “true”, and then run the MQL console and force compile the JPOs to be debugged. You can then attach to the JVM, started on behalf of the MQL console, on port 11111 from any number of IDEs (such as NetBeans or Eclipse).

To debug in an application server environment, you must determine the technique specific to each type of application server on how to setup the JVM for debugging. The you must run in the RIP mode version of Live Collaboration server so that the JPOs run in the same JVM as the application server. This makes it easier to debug your Java business logic that might be spread across JSPs, Beans, and JPOs, since only 1 JVM is running.

## Java and exec Commands

The `java` command has been added and the `exec prog` command has been enhanced for JPOs. The syntax for these commands is as follows:

```
java CLASS_NAME [-method METHOD_NAME] [ARGS] [-construct
ARG];
```

```
exec program PROGRAM_NAME [-method METHOD_NAME] [ARGS]
[-construct ARG];
```

where ARGS is zero or more space-delimited strings and ARG is a single string. If the method returns an int, then this is assumed to be the exit code from the method (which has meaning to the ENOVIA Live Collaboration system when used as a trigger program, wizard utility program, etc.). If the method returns an Object, then the exit code is set to zero and the object is serialized and passed back in the program's output stream. Obviously, this serialized object is of no use in the MQL environment, but is very useful in the Studio Customization Toolkit environment (see next section).

The `java` command takes any class name and finds that class both inside a JPO and outside the ENOVIA Live Collaboration database. The class name must be in its fully mangled form.

The classname selectable can be used on JPO programs to print the classname specified.

The `exec program` command can be used to avoid having to know the mangled class name. However, the class has to live inside a JPO (no search is made outside the ENOVIA Live Collaboration database).

The `-construct` clause is used to pass arguments to the constructor. (**Note**: if more than one argument needs to be passed to the constructor, the `-construct` clause can be repeated as many times as necessary on a single command). This may be necessary to establish an object's identity. For example, if a JPO named "emxProject" held business logic for managing business objects of type "Project", then this JPO would undoubtedly be derived from the Framework emxDomainObject JPO (which derives from the Studio Customization Toolkit BusinessObject class).

```
import matrix.db.*;
public class ${CLASSNAME} extends ${CLASS:emxDomainObject} {

  public ${CLASSNAME}(Context context, String[] args) throws
Exception
  {
    super(args[0]);
  }

  public int ComputeEndDate(Context context, String[] args)
throws Exception
  {
     // do work here to compute this Project's end date…
```

```
        return 0;
      }
    }
```

With this assumption, to compute the end date of an existing Project, the following command would be executed inside a Tcl method on the schema type Project:

```
exec prog Project –method ComputeEndDate –construct ${OBJECTID}
```

A minor addition was also made to launching methods on a business object so that the term “method” is not overloaded when using a JPO method. The new syntax allows either the keyword “method” or “class” to be used. For a JPO it reads better to use the keyword “class”:

```
exec bus T N R class Project –method ComputeEndDate –construct
${OBJECTID}
```

***Note****: As stated earlier, once constructed, an object stays in existence until the transaction is completed. This means subsequent method calls need not supply the* `-construct` *clause.*

2

# Program Object Coding Strategies

## Handling Variables

Because the operating system is unaware of the business objects stored in ENOVIA Live Collaboration, a method is needed to pass information from ENOVIA Live Collaboration to the command line in order to make the commands within a program's code useful.

In early versions of ENOVIA Live Collaboration, an internal structure called a macro was used to pass values as variables between ENOVIA Live Collaboration and the operating system. In Version 6, the Runtime Program Environment (RPE) was added to allow passing arguments into and out of Tcl programs. With the introduction of Java programs, arguments can be passed in directly through the calling sequence, and returned information is in the return argument; the Java language does everything needed, and so there is less of a need for the RPE. While macros and the RPE continue to be supported, consider the following usage of variables in programs (excluding wizards) and the best way to handle them:

1. *ENOVIA Live Collaboration needs to pass information to a program.*

   Programs should be configured with input arguments outside of the program code (either in their program object definition, or as part of their invocation (using exec prog PROG_NAME ARG1 ARG2...in a wrapper or calling program or in MQL interactive mode). Input arguments are text substituted by ENOVIA Live Collaboration at run time before being passed to the program; no additional requests to the RPE are needed.

Tcl programs can include macros in their code to perform macro substitution, but better performance is generally experienced with input arguments. This is because the text substitution makes one pass through the script, and then the program execution makes another pass.

2. *ENOVIA Live Collaboration needs information passed back from a program.*

   Exit codes (integers) must be returned for trigger programs. (Refer to *Generating an Exit Code* for details.) Other programs are expected to return strings, such as those used for attribute ranges, or in expressions in access filters, table columns, cues, tips, etc. JPOs used for these purposes can be defined to return a string and so there is no need to do a “set env”, which, again means better performance.

3. *Programs need to pass information to each other.*

   When they are directly associated, that is, if one program calls another, the top level program should get all data needed from input arguments (or get env) and should pass arguments to all called programs. This is true even with Tcl (RPE should not be relied upon).

   Programs that are not directly associated, such as a check trigger and an action trigger) must resort to the RPE, where program1 calls 'set env' and program2 calls 'get env'.

## Wizards

Wizards are a very special case, since ENOVIA Live Collaboration  and the supporting programs are passing lots of data back and forth. When JPOs are used for wizard components, variables are passed using input arguments. When other types of programs are used for components of a wizard, the RPE is used. When a JPO needs to return data to ENOVIA Live Collaboration, it is returned as a string value from the program. Refer to programming reference section of the *MQL Guide* for details.

## Transaction Management

A *transaction* is defined as *a unit of work* which needs to succeed or fail as a whole. Transaction come in two varieties, readonly and update. Readonly transactions exist primarily as means to improve performance whereas update transaction exist primarily as a means of maintaining database consistency.

Readonly transactions should be used whenever combining more than two Studio Customization Toolkit or mql operations. The presence of the active transaction allows the ENOVIA Live Collaboration core to cache information about the objects being retrieved. For instance, a transaction boundary allows the core to keep pertinent information about the policies, states, access rules, group/role membership, etc. in memory. If multiple operations are being performed on the same object, or objects using the same policy, then the transaction will improve performance by avoiding re-evaluation of the same polices, states, etc. for each operation.

Update transactions should be used whenever changes to multiple objects must be kept in sync. For instance, when adding objects to a structure, it may be necessary to create multiple relationships as a single unit of work. This task should be performed inside an update transaction to ensure that either all the relationships are created or none of them are. This ensures the database remains in a consistent state.

Update transactions also have the performance benefit associated with readonly transactions. That is, ENOVIA Live Collaboration will cache information about the structure of objects that are encountered during the transaction. Update transactions add some additional overhead because when writing to the database, management of the internal cache is much more expensive, but still far better than making requests to the database for each operation.

Update transactions introduce the possibility of *deadlock*. Deadlock occurs when two processes (or threads) are competing for the same resources. For instance, process #1 tries to update object A, then update object B. At the same time, process #2 tries to update object B, and then update object A. One of these two processes will encounter a deadlock situation and their transaction will fail. To avoid deadlock, when ever possible, update objects in a predictable order. For example, when updating a structure, always update top-down. When updating a list of business objects, sort them by object id prior to update. The key is to be consistent in all parts of the application.

A program may have only one transaction open at a time. An attempt to start a new transaction when a transaction is already active will result in an error.

It is absolutely essential that whenever a program starts a transaction it takes the necessary steps to ensure that the transaction is either committed or aborted within the scope of the program. In Java and Tcl, this is generally accomplished by placing a 'catch' block around code that uses transactions. All exceptions must be caught, not just Studio Customization Toolkit or ENOVIA Live Collaboration related errors. Failure to catch all exceptions will leave the transaction active in the event of an unrelated programming failure (such as array out of bounds, cast exception, etc.). The consequence of leaving transactions active is that the transaction will hold on to vital system resources (oracle connections, cursors, memory) essentially “forever”. If many active transactions are left stranded by the application, the system will eventually run out of resources and fail.

## Studio Customization Toolkit methods for Transactions

The `isTransactionActive` method returns true if a transaction is active, false if not. However, note that there is a distinction between an active transaction and one that is able to be committed. If an error has occurred since the transaction was started, it may still be active, but not "commited". You can use `getTransactionType` to provide more specific information, as explained in the javadocs:

```
      /**
       * Determine the current transaction state.
       * <p>
       * Transactions can be started as 'read' or 'update' transactions.
       * They will remain active until committed or aborted.
       * If an error occurs, they will remain active, but will be marked as
       * 'aborting' to indicate that they cannot be committed.
       * <p>
       *
       * @return current transaction state (See Context Field Summary). One of:
       * Context.TRANSACTION_TYPE_INACTIVE: No transaction is active.
       * Context.TRANSACTION_TYPE_READ: Read transaction is active.
       * Context.TRANSACTION_TYPE_UPDATE: Update transaction is active.
       * Context.TRANSACTION_TYPE_ABORTING: Transaction is aborting - an error has
occurred that would block commit.
       * @since ADK 9.0.3.1.0.1
       */
      public int getTransactionType()
```

In addition the context class method, `commitWithCheck` provides the following functionality:

```
/**
* Checks if the current transaction is commitable, and commits/aborts accordingly.
* The transaction is commitable if it is active and there have been no Matrix system
errors
* since it was started (including errors that were caught).
* In all cases, the transaction state will be inactive after this call.
*
* @return true if transaction was committable (active, no errors since start), false if
no * transaction started or transaction is not committable.
* @exception MatrixException if database access error.
*/
public boolean commitWithCheck() throws MatrixException
```

## Optimizing Programs

This section contains standards and suggestions to help for improving the performance and/or maintenance of custom code.

### Using Limits

Using limits effectively can definitely improve performance. Refer to the *ENOVIA Live Collaboration Installation Guide* for details about the limit environment variables.

MX_BOS_EXPAND_LIMIT is used by the following methods:

```
BusinessObject.expand
BusinessObject.expandTable
BusinessObject.expandIntoStructuredSet
BusinessObject.expandIntoStructuredSetSort
BusinessObject.expandSelect
BusinessObject.getToRelationship
BusinessObject.getFromRelationship
BusinessObject.getAllRelationship
Table.expandAndEvaluate
Structure.loadStructure
Structure.select
Structure.subset
```

If the limit is reached, a warning is passed back to the caller.

MX_BOS_FIND_LIMIT is not used by the following calls, because limits are not stored in the database:

```
create(Context context)
update(Context context)
```

However, you can initialize the limit to that set in MX_BOS_FIND_LIMIT for an Studio Customization Toolkit Query object after reading other fields from db with:

```
open(Context context)
```

To set a limit for a Query object, in either a named or temporary query, you can use:

```
setObjectLimit(short limit)
```

The following will use the limit set in the Studio Customization Toolkit Query object, and if not set there, will use MX_BOS_FIND_LIMIT:

```
  evaluate(Context context)
  evaluate(Context context, Visuals visuals)
  evaluateIntoSet(Context context, String setName)
  select(Context context,StringList selectStatements)
  selectTmp(Context context,StringList selectStatements)
  evaluateIntoSet(Context context, String setName)
  evaluateIntoNamedSet(Context context, String setName, boolean
overwrite)
  evaluateIntoNamedSet(Context context,
  String setName,
  boolean overwrite,
  String sortBySelect,
  int direction,
  int dataType,
  int returnCount,
  Visuals visuals)
```

## History logging

If your schema uses one central object that has many objects constantly connected and disconnected from it, you should consider turning propagate connect/disconnect off on one or both sides of the relationships involved. Performance and dead lock errors can be avoided when history records are not logged, thereby reducing the need for locks in the database. Examples of these types of objects are MBOMs or a central person in the processing of routes. Refer to the *Business Modeler* chapter on relationships and the "Working with Metadata" chapter in the *MQL Guide* for details.

If you don't need history to be logged at all, turn off history for particular transactions or for the system entirely. Refer to "Extending an Application" in the *MQL Guide* for details on history.

## Sorted Sets

Query and expand operations in ENOVIA Live Collaboration have always worked from a temporary object list that is built in memory. This technique is fine for moderately-sized results, say 10,000 objects or less. But for queries returning upwards of 100,000 or 1,000,000 objects, superfluous amounts of available memory would be needed to complete the operation, or the operation could lead to server stability issues.

Programmer's can reduce memory usage on the ENOVIA Live Collaboration server by querying and expanding objects into sets. Sets are ideal for holding large results because:

- They can be constructed with a very small footprint at an acceptable cost in time and storage.
- They are easily indexed, and can now be made to page very efficiently (see *Pagination*, below).
- They offer a stateless programming interface (as opposed to a query-in-progress).

### Sorting sets

By default, ENOVIA Live Collaboration presents business objects in sets sorted alphabetically by Type then Name, then revision. You can disable this alphabetic sort, or specify other "Basic" properties by which objects in sets should be sorted, by using the following environment variable:

> **MX_SET_SORT_KEY** sets any number of basic select items (type, name, revision, owner, locker, originated, modified, current, and policy) by which objects in sets will be presented. By default the value is `type|name|revision`. You can specify ascending or descending order by prefixing the select with + or - (ascending (+) is the default.) To disable sorting, you can set this variable to a single '|' character. Disabling sorting improves response time of loading sets.

In addition to this global sorting, you can explicitly sort a set by any list of select values in MQL using the following syntax:

```
sort set SET_NAME [into SET_NAME] [select VALUE1 VALUE2...];
```

As with the sorting environment variable, you can specify ascending or descending order by prefixing the select with + or - (ascending (+) is the default.) For instance:

```
sort set x select +name -owner;
```

This command will perform a sort on both `name` and `owner` fields. Names will be sorted in ascending order, then owners will be sorted in descending order.

The Studio Customization Toolkit has the Set.sort() function that can accept multiple sort keys. Refer to the *Studio Customization Toolkit Reference Guide* (JavaDocs) for details.

When programming applications that will access ENOVIA Live Collaboration through the ENOVIA Live Collaboration server. The best performance will be realized if you:

- save all queries and expands into sets
- use the Studio Customization Toolkit or MQL to sort the set explicitly
- Optionally, disable dynamic sorting with MX_SET_SORT_KEY in the ENOVIA Live Collaboration server. The choice of whether dynamic or static sorting is used is left to the application developer. Static sorting will incur a longer time to display the first page of a result, but subsequent pages will be extremely fast. Dynamic sorting will display the first page faster, but delivery of subsequent pages will suffer. It may be desirable to introduce a new paradigm into the apps that models a "query result" object which is in fact a sorted set. Queries can be sent to a job queue to be executed and sorted and the user can be notified when they are ready. The query result can then be opened and scrolled through very efficiently.

## Pagination

Whether a set is sorted or not, it can always be paged through. In MQL, you can print a subset of a set using:

```
print set SET_NAME [start START_RANGE] [end END_RANGE];
```

The parameter START_RANGE is inclusive and zero-based. The parameter END_RANGE is exclusive and zero-based. To scroll through an entire set, you would use commands that resemble:

```
print set SET_NAME start 0 end 100;
print set SET_NAME start 100 end 200;
print set SET_NAME start 200 end 300;
```

Methods for performing these operations are also available in the Studio Customization Toolkit (Set.subset, Set.subsetSelect). Refer to the *Studio Customization Toolkit Reference Guide (*JavaDocs) for details.

In the use of pagination, you can improve the application's performance by using a page multiplier. That is, an application should request the information for some number of pages in one chunk, for example, 10 pages. Then as the user scrolls through pages 1-10, no server traffic is involved. Then, when page 11 is hit, the next 10 pages are fetched. The reason for this is the data for a small number of pages (10) can be retrieved almost as efficiently as for a single page. Making 10 small requests is much more expensive that 1 larger one.

*Show access is not checked with set command. For more information, refer to "Controlling Access" in the MQL Guide.*

## Access Business Objects and Relationships by Vault or OID

All queries for business objects should use the `vault` keyword if possible. Otherwise, the core must execute an SQL statement for all vaults in the database.

Similarly, if the business object's database identifier (OID) is known, it should be used whenever possible when executing a 'bus' MQL command. If only the type, name, and revision are known, the `in vault` clause should be used to improve performance. The RPE contains variables for both business object and relationship OIDs (OBJECTID and CONNECTID).

For example, this statement:

```
expand bus Part P1 A from relationship EBOM;
```

can be rewritten as:

```
expand bus $objId from relationship EBOM dump terse;
```

This eliminates the need for the system to search every vault before the expand is performed.

"Expand bus Part P1 A ..." starts by searching every vault for Part P1 A until it finds it.

```
select * from lxBO_5f39822c where
    lxType=:va and lxName=:vb and lxRev=:vc
           :va=1737757489
           :vb=P1
           :vc=A
select * from lxBO_678a4321 where
    lxType=:va and lxName=:vb and lxRev=:vc
           :va=1737757489
           :vb=P1
           :vc=A
select * from lxBO_876cdea1 where
    lxType=:va and lxName=:vb and lxRev=:vc
           :va=1737757489
           :vb=P1
           :vc=A
```

This search may be repeated in every vault until the object is found.

"Expand bus $objId ..." skips these preliminary queries altogether. The `terse` clause on the end saves one or more instances of the following:

```
select t.* from lxBO_0a70d4cb t where t.lxOid in (select lxToId
from lxRO_0a70d4cb where lxFromLat=:va and lxFromId=:vb and
lxToLat=:vc)
        :va=175166667
        :vb=1984423433
        :vc=175166667
```

This query is used to get the Type Name Revision of the related objects. It is unneeded with the `terse` clause. The related object's OID is free. This is especially true if you are going to do further processing on the related object. (For example, a further expand or an update.) The same initial query savings from above apply every time you use an OID instead of TNR.

## Selecting Relationship and Business Object Data

Avoid getting business object selectable items one at a time. Code similar to the following should never exist:

```
///////////////////////////////////////////////////////////////
// Do not have code like below
///////////////////////////////////////////////////////////////

      // Query using MQLCommand

            MQLCommand mqlCmd = new MQLCommand();

            mqlCmd.executeCommand(ctx, "temp query bus t2 t2* 0
dump \n");

            String rslt = mqlCmd.getResult().trim();
           StringTokenizer st1 = new StringTokenizer(rslt,"\n");
            while (st1.hasMoreElements()) {
               String resultBo = st1.nextToken().trim();
               System.out.println("Result bo ="+resultBo);
               MQLCommand subMqlCmd = new MQLCommand();
               String mqlStr = "print bus "+resultBo+ " select
attribute[int-u]  attribute[color-u];";
               subMqlCmd.executeCommand(ctx, mqlStr);
               String subResult = subMqlCmd.getResult().trim();
               System.out.println("Sub result ="+subResult);
            }
```

Instead, multiple selectables should be specified in one command. Multiple selectable items can be specified for any of the following commands:

- print bus
- expand bus
- temp query
- print set

For example:

```
           MQLCommand mqlCmd = new MQLCommand();

           mqlCmd.executeCommand(ctx, "temp query bus t2 t2* 0
select attribute[int-u].value attribute[color-u].value
from[r3].attribute[rstring-u].value dump |");

          String result = mqlCmd.getResult().trim();
          StringTokenizer st = new StringTokenizer(result,"|");
           while (st.hasMoreTokens()) {
               System.out.println("Result token
="+st.nextToken());
            }
```

The coding approach shown above could be made more efficient by using the following syntax:

```
          // Query using ADK calls
          StringList busSelect = new StringList(7);
            busSelect.addElement("id");
            busSelect.addElement("type");
            busSelect.addElement("name");
            busSelect.addElement("revision");
            busSelect.addElement("attribute[int-u].value");
            busSelect.addElement("attribute[color-u].value");

busSelect.addElement("from[r3].attribute[rstring-u].value");
            // Prepare temp query
            Query query = new Query();
            query.setBusinessObjectType("t2");
            query.setBusinessObjectName("t2*");
            query.setBusinessObjectRevision("0");
            query.setOwnerPattern("*");
            query.setVaultPattern("unit1");
            query.setWhereExpression("");
            BusinessObjectWithSelectList bwsList =
                new BusinessObjectWithSelectList();

            // Do the query
            bwsList = query.selectTmp(ctx, busSelect);

            // Iterate through the objects returned and get
            // data.
            for (int idx=0; idx<bwsList.size(); idx++)
            {
             BusinessObjectWithSelect bws =
(BusinessObjectWithSelect) bwsList.elementAt(idx);
             String sType = (String)bws.getSelectData("type");
             System.out.println("Type ="+sType);
             String sName = (String)bws.getSelectData("name");
             System.out.println("Name ="+sName);
             String sRevision =
(String)bws.getSelectData("revision");
             System.out.println("Revision ="+sRevision);
             // Get businessobject attribute values
             String attrOneVal =
(String)bws.getSelectData("attribute[int-u].value");
             System.out.println("AttrOneVal ="+attrOneVal);
             String attrTwoVal =
(String)bws.getSelectData("attribute[color-u].value");
             System.out.println("AttrTwoVal ="+attrTwoVal);
             // Get relationship attribute values
             String attrRelVal =
(String)bws.getSelectData("from[r3].attribute[rstring-u].value"
);
             System.out.println("AttrRelVal ="+attrRelVal);

              }
```

When an MQL selectable item is used to retrieve information about a relationship, the "relationship" item should not be used if a direction is known. Instead, "to" or "from" should be used. This reduces the number of database tables that need to be searched. For example, instead of:

```
relationship\[BOM\].attribute\[Qty\]
```

Use:

```
from\[BOM\].attribute\[Qty\]
```

## Referencing Schema Object Names

Avoid hard coding names of schema objects. Instead, use a strategy to look up the names of schema objects such as that found in the Application Exchange Framework or the Application Library.

## Query Performance

The key consideration for optimizing query performance is to use as many SQL convertible fields as possible. This limits the number of business objects returned to the client in order to perform the non-SQL convertible evaluation. It's important to limit the use of wildcards for type, name, and revision, and to specify a search vault.

Never use a temporary query to print information about a business object if the type, name, and revision of the business object are already known. For instance, the following code fragment should never exist:

```
///////////////////////////////////////////////////////////////
// Do not have code like below
///////////////////////////////////////////////////////////////

             MQLCommand mqlCmd = new MQLCommand();

             mqlCmd.executeCommand(ctx, "get env TYPE");
             String type = mqlCmd.getResult().trim();
             mqlCmd.executeCommand(ctx, "get env NAME");
             String name = mqlCmd.getResult().trim();
             mqlCmd.executeCommand(ctx, "get env REVISION");
             String revision = mqlCmd.getResult().trim();
            String mqlString = "temp query bus "+type+" "+name+"
"+rev+"select current dump"
             MqlCmd.executeCommand(ctx, mqlString);
```

Use a `print bus` command instead. Ideally, this command would use the business object ID instead of the type, name, and revision as discussed in *Access Business Objects and Relationships by Vault or OID*.

One reason that a programmer may be tempted to use a query to print business object data is because it will not cause an error if the type, name, and revision does not exist. Even in these cases, there are better ways to write the code:

```
///////////////////////////////////////////////////////////////
// Instead do the following:
///////////////////////////////////////////////////////////////

        try
```

```
            {

                MQLCommand mqlCmd = new MQLCommand();

                 mqlCmd.executeCommand(ctx, "get env TYPE");
                 String type = mqlCmd.getResult().trim();
                 mqlCmd.executeCommand(ctx, "get env NAME");
                 String name = mqlCmd.getResult().trim();
                 mqlCmd.executeCommand(ctx, "get env REVISION");
                 String revision = mqlCmd.getResult().trim();
                 String mqlStr = "print bus "+type+" "+name+"
"+revision+" select exists dump |";
                mqlCmd.executeCommand(ctx,mqlStr);
                String result = mqlCmd.getResult().trim();
               if (result.equalsIgnoreCase("TRUE") ){
                  String selectStr = "print bus "+type+" "+name+"
"+revision+" select current dump |";
                   mqlCmd.executeCommand(ctx,selectStr);
                   result = mqlCmd.getResult().trim();
                   System.out.println("REsult ="+result);
              }
        }

            catch (Exception ex)
            {
                 throw ex;
            }
```

## Program Object Select Expressions

`Select` expressions are used to obtain information about business or administrative objects, which can then be saved as a variable and used in a program.

Each select expression has a default behavior to minimize the amount of typing required for an answer. In the case of simple data types (string, integer, etc.), the default is always to return the value. Objects have different behavior in each case. The system attempts to produce output for each select clause input, even if the object does not have a value for it. In this case, an empty field is output.

## Using Transaction Boundaries

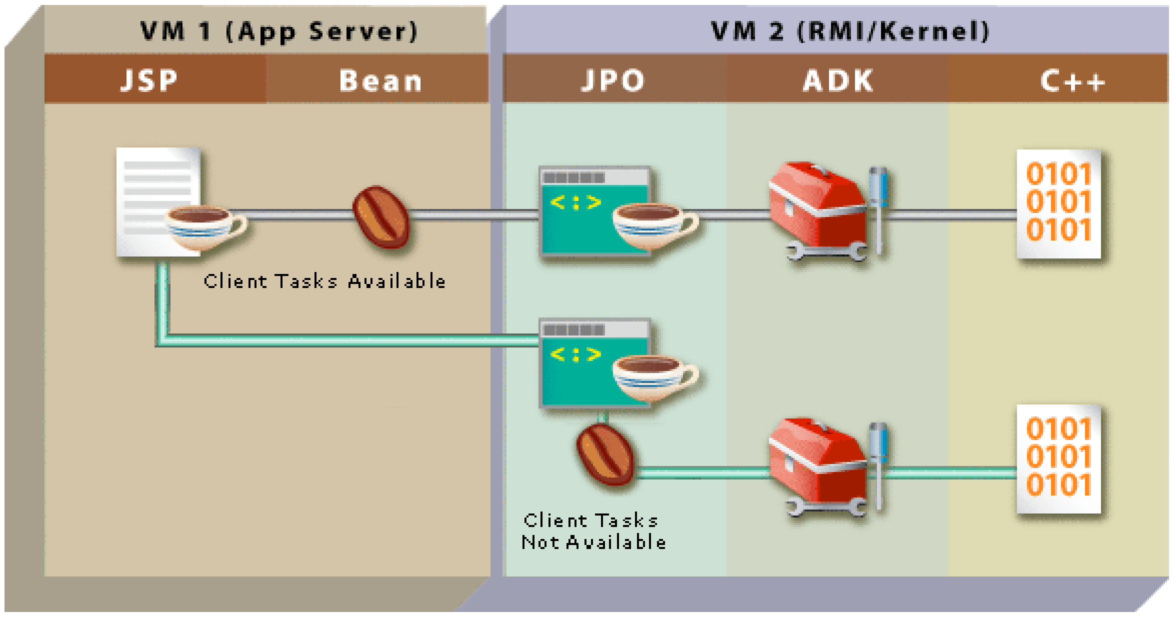
Transaction boundaries (mql `start trans` and `commit/abort trans` commands) are primarily used to roll back processing if an error occurs. If many commands are required to successfully complete a logical operation, then transaction boundaries are used. Note that in the case of event triggers, unless they are external or a deferred action, the ENOVIA Live Collaboration or MQL interface layer automatically applies transaction boundaries, and explicitly setting them in the code causes an error.

Using transaction boundaries for methods or wizard code can provide some performance benefits even if the commands inside the boundaries are not a single logical operation. When no transaction is in effect, instance data (business objects, relationships, attributes, etc.) must be reloaded from the database for each MQL command. Transactions can be used to keep data live in the cache between commands, which improves performance.

## Client Task Delivery

JSP’s and Java beans running in the Application Server are considered clients to the C++ ENOVIA Live Collaboration kernel. Also, in ENOVIA Live Collaboration non-RIP (ENOVIA Live Collaboration server) installations, the client Application Server and the ENOVIA Live Collaboration kernel are, in fact, running in entirely separate Java virtual machines. In light of this, the Live Collaboration kernel is designed to guarantee that client tasks generated by nested calls to Studio Customization Toolkit/JPO/trigger programs will be returned to the calling client, and not to other intermediate layers that run within the server. This design requirement also applies in a RIP configuration, even though the client and server happen to be running in the same Java process.

Understanding this boundary between client and server is what tells you where you can (and should) check for client tasks, such as notices, that may have been produced in a call to the kernel. Consider the diagram below where a JSP calls a bean method that further makes an Studio Customization Toolkit call to call into a JPO or where it calls a JPO that calls a bean that calls into the Studio Customization Toolkit:



A few basic points:

- JSPs always run in the Application Server client.
- JPOs always run in the kernel server.
- Bean methods may be invoked from JSPs, and run in the Application Server client, or called by JPOs, and run in the kernel server.
- Studio Customization Toolkit calls made from JSPs or bean methods are the interfaces that crossover from Application Server client to kernel server. (This is where client tasks are returned)
- Studio Customization Toolkit calls made from JPOs or bean methods running in the kernel are also run in the kernel and do not cross the client/server boundary. (So when run from the kernel (thick client), client tasks are returned here).

## Error handling

Various error codes can be returned when executing programs within an ENOVIA Live Collaboration implementation. The way a program is invoked affects the possible returned values and errors. The following are the various ways of invoking programs in ENOVIA Live Collaboration from a calling program.

1. A very common programming practice when a Tcl program calls other Tcl or JPO programs is the following:

   ```
   set errCode [catch {eval mql execute program testtcl2} retString
   ```

2. The Studio Customization Toolkit method executeCommand() can be called to run programs. A typical code segment used is below:

   ```
   MQLCommand mql = new MQLCommand;
         Boolean returnVal = mql.executeCommand(context,"execute
   program testtcl");
          String retString = mql.getResult();
          String errString =  mql.getError();
   ```

3. The Studio Customization Toolkit method JPO.invoke() can be called to run programs. A typical code segment using this is below:

   ```
   (String) returnVal = (String)JPO.invoke(context,
            "testjpo2",
            null,
            "testmethod"
            args,
            String.class);
   ```

4. JPO classes can be called in directly from other JPO's as follows:

   ```
   returnVal= ${CLASS:JPONAME}.METHODNAME();
   ```

The programmer must be aware of all the possible return values from these invocations, and provide logic to handle all applicable cases.

MQL/Tcl programs can return information to the caller in various ways:

- Using the "return" statement:
  - return 0
  - return 1
  - return RETSTRING
- Using the "exit" statement.
  - exit 0
  - exit 1
- Using the "return -code 1 RETSTRING" statement
  - return -code 1 RETSTRING
  - return -code 0 RETSTRING
- Producing an error
  - return –code 1 returns like an error
  - Tcl syntax errors, undefined variables
  - Uncaught mql errors: set id [mql print bus Does Not Exist select id]

Java programs can return information to the caller in a number of ways:

- return int
- return String
- return void
- throw Exception

The table below describes what values are returned to the caller in each of the above cases:

| Caller code | Returned Values |
|---|---|
| `set errCode [catch {eval mql execute program testprog} retString` | **errCode** is set to 0. errCode could be set to 1 if:<br>• Tcl testprog has "exit 1"<br>• Tcl testprog has return –code 1<br>• there is a Tcl syntax error or other uncaught error.<br>• JPO testprog returns 1 as int or Integer<br>• JPO testprog throws exception.<br>**RetString** is set to a string passed back in the return statement of the called program. It is set to "" in the following scenarios:<br>• For all JPOs.<br>• In case of "return –code 1 R" in a Tcl program, retString is appended by the string "exec program failed". So retString looks like :<br>`R+ 'Error: #1900068: exec program failed'` |
| `MQLCommand mql = new MQLCommand;`<br>`Boolean returnVal = mql.executeCommand(context,"execute program testprog");`<br>`String retString = mql.getResult();`<br>`String errString = mql.getError();` | **ReturnVal** is true. ReturnVal is set to false if:<br>• Java testprog throws exception<br>• Tcl testprog has "exit 1" –<br>• Tcl testprog has "return –code 1 STRING"<br>**retString** is set to a string value from the called program, set by:<br>• Tcl testprog has "return R"<br>**ErrString** is empty , except in the following cases:<br>• JPO throws an exception. ErrString is set to the exception string in this case.<br>• If a called Tcl program has "return –code 1 R", then errString is set to:<br>`R + 'Error: #1900068: exec program failed'`<br>executeCommand never throws an exception unless there are connectivity problems with the kernel. |
| `returnVal = JPO.invoke(context, "testjpo2", null, "testmethod", args, String.class);` | **returnVal** is set to the value returned from the invoked JPO<br>In case of the called JPO throwing an exception, returnVal is set to ""<br>JPO.invoke() unlike MQLCommand.executeCommand() will throw an exception if a call to the kernel causes an error. |
| `returnVal= ${CLASS:testjpo2}.testmethod();` | **returnVal** is always set to the return value from the invoked JPO.<br>In case of the called JPO throwing an exception, returnVal is set to "" |

It should be noted that when Tcl programs have an "exit 1" in the code or when JPO programs are defined to return int or Integer and have a "return 1" in their code, these returns are interpreted as "exitCodes" when running triggers or wizards in ENOVIA Live Collaboration.

## Program Exit vs. Program Failure

After ENOVIA Live Collaboration executes a program, it will first determine if the program executed successfully or failed. If it failed, an exception is raised, which generally aborts the transaction and displays an error. If it executed successfully, in some cases ENOVIA Live Collaboration will then look at the exit code returned by the program. The table below shows the eight cases where the exit code of a program *is* evaluated, and what a non-zero exit code signifies. The effect of failures is also listed.

| Program Type | Non-zero Exit Code causes... | Failure causes... |
|---|---|---|
| state action | the promotion of the business object to be blocked and “Can’t promote T N R” message to be displayed. | the promote transaction to be aborted, and an error to be displayed. |
| state check | the promotion of the business object to be blocked and “Not all requirements satisfied” message to be displayed. | the promote transaction to be aborted, and an error to be displayed. |
| trigger check | the event to be blocked and an error displayed. | the event transaction to be aborted, and an error to be displayed. |
| trigger override | the event to be skipped, but the transaction committed. | the event transaction to be aborted. |
| wizard validate | the wizard to remain on the current frame. | the Wizard to fail. |
| wizard epilogue | the wizard to remain on the current frame. | the Wizard to fail. |
| wizard prologue | the wizard to skip the current frame. | the Wizard to fail. |
| wizard button | the wizard to redisplay the current frame. | the Wizard to fail. |
| attribute range | an error to be displayed, because the entered value is not acceptable. The exit code is only evaluated for this type of program when EVENT = “attribute check.” Refer to *Programs for Attribute Ranges* for more information. | the values provided by the program to be unavailable for selection or validation. The only values that can be written to the attribute are those that are within other defined ranges. |

Note that for both state checks and trigger checks, the outcome of a failure is the same as when the exit code is non-zero. So for these two cases, programmers could accomplish the same thing either by designing the program to fail or by returning a non-zero exit code.

In general, exit code is *not* checked when ENOVIA Live Collaboration is returned to the idle state after a program is executed. This occurs in the following circumstances:

- Programs executed as Action Triggers
- Programs executed as methods
- Programs executed from a toolbar
- Wizard body code
- When any program fails

This is not to say that a method, for example, that contains nested programs cannot pass exit codes between those routines.

*Note that lifecycle actions are not on this list. Exit code IS evaluated on lifecycle actions, where a non-zero exit code will abort the promote transaction.*

## Generating an Exit Code

In general, exit codes are evaluated by ENOVIA Live Collaboration only when the program is executed within a wizard or trigger (or lifecycle check or action), where the flow of control keeps moving (that is, ENOVIA Live Collaboration still has more to do, and is not idle). Exit codes are generated in a variety of ways, depending on the platform and programming language being used.

### To return an exit code

- From an internal (Java) program, the JPO method should return an "int". The int value is taken as the return code. If the JPO method returns an object, the object must be an integer and this object's value will be the return code.
- From an internal (MQL) program in MQL mode, use the `quit` command. The `quit` command passes a value back though the use of an integer argument (for example, `quit 0`).
- From an internal (MQL) program in Tcl mode, use the `exit` command. The `exit` command passes a value back though the use of an integer argument (for example, `exit 0`).
- From an external program, use the normal exit code mechanism supported on each platform and programming language.

*Note that the Tcl command "*`return 1`*" is NOT evaluated by ENOVIA Live Collaboration as an exit code.*

### Exit Inside of an Eval Statement

An exit inside of an eval statement is processed differently than you might expect.

ENOVIA Live Collaboration replaces the real Tcl exit command with its own in order to prevent an exit command from shutting down MQL, ENOVIA Live Collaboration, ENOVIA Live Collaboration Kernel, etc. ENOVIA Live Collaboration sets a flag, which it checks before passing a new statement to Tcl. However, all that ENOVIA Live Collaboration sees of the statement in the following example is `eval {...}`, just one big eval statement. ENOVIA Live Collaboration must exit before it sees that the flag was set.

So the following code continues to execute after the `exit` command:

```
tcl;
eval {
    exit
    puts "This should never print."
}
puts "Tcl will exit before running this."
The following code works as would be expected.
tcl;
eval {
    exit
    return
    puts "This should never print."
}
puts "tcl will exit before running this."
```

```
The built in return forces it to stop executing the eval
statement, which then notices that the exit flag was set and
closes Tcl.
```

## Program Failure

Programs will fail in certain circumstances. Syntax errors are obvious causes of program failure. System errors caused by MQL that makes assumptions (such as the existence of a business object or relationship), also fail, aborting the enclosing transaction whether or not it is caught and handled by the Tcl "catch" command. Therefore, always avoid using MQL commands which could generate system errors. For example:

```
////////////////////////////////////////////////////////////////
// Do not have code like this. Doing it like below does not
// check any errors from the mql command executed.
////////////////////////////////////////////////////////////////

        try
        {
            MQLCommand mqlCmd = new MQLCommand();
            mqlCmd.executeCommand(ctx,"print rel doesnoexit ");
         }

        catch (Exception ex)
        {

            System.out.println("Relationship does not exist" +
ex.toString());
        }

////////////////////////////////////////////////////////////////
// Code it like below. Check the error from the command
// executed. The code below will hit get an error from the
//getError() call if the relationship does not exist.
////////////////////////////////////////////////////////////////

            try {
            MQLCommand mqlCmd = new MQLCommand();
            mqlCmd.executeCommand(ctx,"print rel doesnotexist
");
            String error = mqlCmd.getError();
            if (error.length() !=0)
                System.out.println("Error ="+error);
            String result = mqlCmd.getResult();
            if (result.length()==0)
                System.out.println("Rel does not exist");
            }

            catch (Exception ex)
            {
              throw ex;
```

```
                        }


                  /////////////////////////////////////////////////////////////////
                  // Another way to code it would be Check the result returned
                  // from the command executed. The code below will have no errors
                  // returned if the relationship does not exist.
                  /////////////////////////////////////////////////////////////////

                        try {
                        MQLCommand mqlCmd = new MQLCommand();
                        mqlCmd.executeCommand(ctx,"list rel doesnotexist");
                        String error = mqlCmd.getError();
                        if (error.length() !=0)
                            System.out.println("Error ="+error);
                        String result = mqlCmd.getResult();
                        if (result.length()==0)
                            System.out.println("Rel does not exist");
                        }

                        catch (Exception ex)
                        {
                          throw ex;
                        }
```

## Error Handing classes

The error handling classes of ENOVIA Live Collaboration, Studio Customization Toolkit, MatrixException and ErrorMessage, provide the following:

- MatrixException.getMessages returns an accurate vector of server-side messages, rather than “last message only”.
- Each of the ErrorMessages returned in a MatrixException can provide an unambiguous error code

In addition all methods that return an “int” also return an MQLERROR (-1) when an error is encountered during execution. For methods that do not return a status, an error status is set internally that can be retrieved by calling `mqlCheckError` that returns the status of any errors that occurred during execution. For example:

```
mqlResumeThread();

// Check to see if the above call errored by doing the following:

int error = mqlCheckError();
if (error == MQLERROR)
   {
      return ERROR;
   }
else {
  // Proceed normally.
 }
```

# Programming for the Web Navigator

When writing programs that may be used with the Web version of Matrix Navigator, there are considerations that need to be addressed. This section provides some tips to keep in mind if programs will be run on the Web.

## Disable Multiple Wizards

To prevent multiple wizards from running concurrently in the same Web Navigator session, you need to set the following parameter in MatrixAppletXML.jsps:

```
props.put("DisableMultipleWizards", "true");
```

When the DisableMultipleWizards parameter is set to "true", only one wizard can execute at a time. If the user attempts to execute a second wizard, the following message displays:

```
Multiple concurrent Wizards have been disabled.
```

If set to "false" or if the parameter is not present, multiple wizards can be run concurrently.

## File Checkin and Checkout

Two clauses of the `checkin` and `checkout` commands are designed for use on the Web only. The `client` and `server` clauses allow programmers to specify where files are to be located when their programs are executed from a Web client (either downloaded or run on the ENOVIA Live Collaboration server). These clauses are used to override the defaults and are ignored when executed on the Studio Modeling Platform client.

By default, `checkout` commands executed from the Web land files on the Web client machine. The `server` clause allows the programmer to alternatively specify the ENOVIA Live Collaboration server for the operation. For example, the following statement in a Tcl program object that is run from the Web client will land the file text.txt on the client:

```
mql checkout bus Assembly Wheel 0 format ascii file
\tmp\text.txt;
```

while the following will land it on the server:

```
mql checkout bus Assembly Wheel 0 format ascii server file
\tmp\text.txt
```

The `client` clause used with the `checkin` command provides the same alternatives. The default file location for checkin from the Web is the ENOVIA Live Collaboration server, so a programmer may specify `client` to have the file copied from the Web client machine instead. For example:

```
mql checkin bus Assembly Wheel 0 format ascii file
\tmp\text.txt;
```

would look for the file text.txt on the server, while the following would look for it on the client:

```
mql checkin bus Assembly Wheel 0 format ascii client file
\tmp\text.txt;
```

In order to be consistent, the `client` or `server` clause may be specified with both checkin and checkout. For example:

```
mql checkout bus Assembly Wheel 0 format ascii client file
\tmp\text.txt;
```

```
mql checkin bus Assembly Wheel 0 format ascii server file
\tmp\text.txt;
```

will yield the same results as:

```
mql checkout bus Assembly Wheel 0 format ascii file
\tmp\text.txt;
```

```
mql checkin bus Assembly Wheel 0 format ascii file
\tmp\text.txt;
```

## Dialog Scripting on the Web

Most `application` commands, used to display dialogs, are supported on the ENOVIA Live Collaboration server. Therefore, users of Matrix Web Navigator and other applications that use the ENOVIA Live Collaboration server to access an ENOVIA Live Collaboration database can execute these types of programs. For additional information, see *Dialog Scripting Language*.

*All parameters passed to an `application` command must be quoted, as shown in the command list.*

*OBJECTID is not yet supported by the `application` command. Objects must be specified by Type, Name, and Revision.*

The following `application` scripting commands are NOT yet supported on the Matrix Web Navigator, or by the Studio Customization Toolkit classes:

- appl edit bus "TYPE" "NAME" "REVISION" (all forms of command)
- appl find
- appl modal DIALOGID
- appl print bus "TYPE" "NAME" "REVISION" (all forms of command)
- appl wait "DIALOGID"
- dialog "DIALOGID" forms of `appl details` and `appl icons` commands
- appl wizard WIZARDNAME

If a program that contains these commands is launched from a Matrix Web Navigator session, an error is displayed.

## Dealing With User Context

The context settings provided by the user at login time define what types of accesses that user has to ENOVIA Live Collaboration. Programs, triggers and wizards may be available to users who do not have appropriate access privileges to run them, so context may be changed within the program and then changed back to the original user's context.

Two MQL commands are available that are useful in scripts that require a temporary change to the session context. In addition, you can define a program to have a user, who's access is used by any context that executes the program.

### Push Context Command

The `push context` command changes the context to the specified person and places the current context onto a stack so that it can be recovered by a `pop context` command. `Push context` can also be issued with no additional clauses, in which case the current context would be placed on the stack, but the current context would be unchanged. The command syntax is:

```
push context [user NAME] [pass PASSWORD] [vault VAULT];
```

### Pop Context Command

The `pop context` command takes the last context from the stack and makes it current. The command syntax is:

```
pop context;
```

This feature is highly useful in scripts where it is necessary to change the current context temporarily. Context is set back to original user after the interim program that changed context ends.

For example, a trigger program may need to switch context to perform some action which is not allowed under the current user context. It must then return to the original context to prohibit invalid access or ownership for subsequent actions.

### Initialization File Context Variable

Alternatively, there is an initialization file variable that controls whether context is restored or not at the conclusion of a program. When the `MX_RESTORE_CONTEXT` variable is set to true (the default), the original user's context will be restored after the program/wizard/trigger completes. If set to false, any context changes made by a program/wizard/trigger will remain changed after the program completes.

## Executing a Program

To execute a defined program, use the following statement:

```
exec program PROGRAM_NAME [-method METHOD_NAME] [ARGS] [{-construct ARG}];
```

`PROGRAM_NAME` is the name of the program to execute.

The other parameters are related to Java Program Objects (JPOs):

`ARGS` is zero or more space-delimited strings and ARG is a single string. If the method returns an int, then this is assumed to be the exit code from the method (which has meaning to the ENOVIA Live Collaboration system when used as a trigger program, wizard utility program, etc.). If the method returns an Object, then the exit code is set to zero and the object is serialized and passed back in the program's output stream. Obviously, this serialized object is of no use in the MQL environment, but is very useful in the Studio Customization Toolkit environment.

The `exec program` command can be used to avoid needing to know the JPO mangled class name. However, the class must live inside a JPO (no search is made outside the ENOVIA Live Collaboration database).

The `-construct` clause is used to pass arguments to the constructor (Note: if more than one argument needs to be passed to the constructor, the `-construct` clause can be repeated as many times as necessary on a single command). This may be necessary to establish an object's identity. For example, if one had a JPO named "Project" that held business logic for managing business objects of type "Project," then this JPO would undoubtedly be derived from the Studio Customization Toolkit BusinessObject class.

For details on programming JPOs, see *Java Program Objects*.

### Command Line Execution

ENOVIA Live Collaboration provides the functionality to run a program without displaying the entire application user interface. This may be useful for users who need to enter information (create objects), but do not need to access ENOVIA Live Collaboration for anything else. On Windows platforms, a shortcut could be added to the user's desktop by the System Administrator. This shortcut could run an ENOVIA Live Collaboration program without displaying the main dialog. For example:

```
matrix.exe -mql -c "execute program PROGNAME;"
```

You can execute any valid MQL command(s) in this manner. However, note that if additional command line options are used, do not separate the -mql and -c arguments.

## ENOVIA Live Collaboration Command Line Options

The following options can be appended to the ENOVIA Live Collaboration command when it is started from the system command line or Windows desktop icon.

| **Option** | **Meaning** |
|---|---|
| `-wizard NAME` | Directly launch business wizard. |
| `-mql` | In general, allows a program to use GUI elements of ENOVIA Live Collaboration (through the *Application Command*) while running what is mostly an MQL/Tcl program. May be used without -c for use with mqlio and mqlbos programs. |
| `-c “COMMAND[ COMMAND];”` | To be used with the -mql option to execute COMMAND(s). Must keep -mql and -c arguments together if other command line options are used. |
| `-v` | Verbose option for command line interface. |
| `-k` | No abort on error for command line interface. |
| `-p` | Use piped command line interface. |
| `-browseby VALUE` | Browse objects by `icons`, or `details`. |
| `-navigateby VALUE` | Navigate objects by `star`, `indented` or `details`. |
| `-filterbar` | Toggle filter bar on. |
| `-revisionwarn` | Warn when object has higher revision. |
| `-filepath PATH` | Default directory for checkin/checkout. |
| `-checkoutlock` | Lock files on checkout. |
| `-checkoutoverwrite` | Overwrite existing files on checkout. |
| `-checkoutconfirm` | Confirm file replace on checkout. |
| `-checkinappend` | Append files on checkin. |
| `-checkinunlock` | Unlock files on checkin. |
| `-checkindelete` | Delete files on checkin. |
| `-checkinconfirm` | Confirm file replace on checkin |
| `-laf VALUE` | Change the ‘look and feel’ of the ENOVIA Live Collaboration graphical user interface (GUI). Possible values are `windows, motif, openlook.` This can be run on any platform to have the GUI mimic that look and feel. |
| *The following options can be used on the command line for the Matrix Navigator or Business or System application.* | |
| `-bootfile FILE` | Set bootstrap file name. |
| `-open VALUE` | Open objects for `view` or `edit`. |
| `-viewby VALUE` | View objects by `icon`, `image` or `name`. |

## Program With Arguments

The keywords `method` and `program` have been added as select keywords for business objects. They can be used wherever select keywords for business objects can be used, so they can be used in both `print bus` and in expressions evaluated against business objects. The syntax is:

```
program [PROGRAM_NAME ARG1 ARG2 …]
```

*Or:*

```
method [METHOD_NAME ARG1 ARG2 …]
```

where `PROGRAM_NAME` is the name of a program

where `METHOD_NAME` is the name of a method

Zero or more arguments can be included as indicated. The total number of characters in brackets is limited to 127.

The `program` keyword allows non-method programs to be run.

- They can use the same macros as a method.
- The arguments are space-delimited.
- The program/method name must be quoted if it contains spaces.
- Arguments containing spaces must be quoted or enclosed in { }.
- If the program exits with a non-zero exit code, the select clause returns an empty string as a value.

The select clause returns a string whose value is specified by the program. The program specifies it by placing the string in the GLOBAL RPE variable whose name equals the program name. This is consistent with the use of a program to load a single valued text widget, except for having to use `global`.

Note that as a select clause, this capability can be used in any of the following ways:

- `print bus T N R select owner current program[NAME] attribute[ATT] dump`
- to load Tcl variables via: `set sVar [mql print bus ....]`
- as part of a where clause in a Query or Cue
- as the expression to be provided in a Tip
- as a column definition

*The use of programs as column headers should be done only in special circumstances, since it will significantly slow down table display because the program has to be run for* each *row.*

## Example:

For the program ‘Program 27’ (code shown below), the command:

```
print bus Bug 202740 0 select program['Program 27' a b c]
```

generates output as follows

```
Run by the owner of Bug 202740 0:
    business object Bug 202740 0
        program['Program 27' a b c] = You own this object
Run by a non-owner :
    business object Bug 202740 0
        program['Program 27' a b c] = You do NOT own this object
Run by a guest:
    business object Bug 202740 0
        program['Program 27' a b c] = You are guest!!
```

The following is the code for ‘Program 27’:

```
tcl;
#
#  A local procedure to allow turning debugging puts on or off
#  by setting/unsetting the rpe variable MXDEBUGFLAG
proc DEBUG { str } {
    if {[mql get env global MXDEBUGFLAG] > 0} {puts $str}
}
eval {

    # Get the current user, and type,name,rev
    set sUser [mql get env USER]
    set sType [mql get env TYPE]
    set sName [mql get env NAME]
    set sRev  [mql get env REVISION]

    # Get the program name into which the return string will be
written.
    set progname [mql get env 0]
    set exitCode 0

    # Verify that arguments are passed in
    for { set indx 1 } { $indx < 10 } { incr indx } {
        set var$indx [mql get env $indx]
        DEBUG "var$indx=[mql get env $indx]"
    }

    # Require at least one argument
    if { $var1 == "" } {
        set exitCode 1
    }

    # Get owner
    if { $exitCode == 0 } {
        set exitCode [catch {mql print bus $sType $sName $sRev \
                select owner dump} sOwner]
    }
    # Set a return value.
    if { $exitCode == 0 } {
        if { $sUser == "guest" } {
            set retval "You are guest!!"
        } elseif { $sOwner == $sUser } {
            set retval "You own this object"
        } else {
            set retval "You do NOT own this object"
        }

        # Write the return value to the GLOBAL program RPE
variable
        DEBUG "Return in progname: $retval"
        mql set env global $progname $retval
    }

    DEBUG "ExitCode: $exitCode"
    exit $exitCode
}
```

## Programs for Attribute Ranges

As described in the attributes chapters of the *Business Modeler Guide* and the “Working with Metadata” chapter in the *MQL Guide*, ranges can be assigned to attributes that explicitly define one or more legal values. In addition, a program may be assigned as a range for a string attribute. This program object is used to produce a different set of legal string values based on the state or settings of the specific object or relationship in which the attribute is used. It is important to keep in mind that these programs are executed under two sets of circumstances:

- When the attribute range values are provided for selection (when the attribute list box presents valid value choices);
- When the validity of a value is checked (when the **Modify** button is pressed in the attributes dialog, or the MQL `check attribute` command executed).

Code must be included to handle both circumstances.

### Example

The following is an example of a program that can be used to define ranges:

```
add program nameRange
mql
  code 'tcl;
eval {
  set event [mql get env EVENT]
  set names {Larry Curly Moe}
  if { $event == "attribute choices"} {
    set output [mql get env 0]
    mql set env global $output $names
  } else {
    set value [mql get env ATTRVALUE]
    if {[lsearch -exact $names $value] == -1} {
      exit 1
    } else {
      exit 0
    }
  }
}'
;
```

## Programs for Personal Use

The example below shows a program that could be used to define the content of a table column or the query of a cue.

```
/*
 **
 ** docTableProgram: example java program for use in table or
cue
 **
 ** This program returns the number of objects of type "docType"
connected to the current
 ** object by the relationship "docRel"
 **
 ** Note that the macro OBJECTID must be explicitly specified
when the program is
 ** configured in a table/cue definition so it can be loaded
into the 'args' array
 ** program is executed.
 **
 ** To configure this as a table column:
 ** add table docTable column label Rels
 **            businessobject program[docTableProgram -method
mxMain ${OBJECTID}]
 **
 ** To configer this as a cue:
 ** add cue docCue appliesto businessobject
 **                color red
 **               where 'program[docTableProgram -method mxMain
${OBJECTID}] > 0';
 **
 */
import matrix.db.*;
import matrix.util.*;

public class ${CLASSNAME}
{


    public ${CLASSNAME}(Context ctx,String[] args)
    {
    }
    public String mxMain(Context ctx,String[] args)  throws
Exception
    {

        // Initialize number of connected objects
        int retval = 0;

        try
        {
            // Declarations
            String objId = new String();
            MQLCommand mql = new MQLCommand();

            // Get and check arguments
            if (args.length > 0)
                objId = args[0];
            if (objId == null || objId.equals("")) {
                mql.executeCommand(ctx, "notice 'No objId for
docTableProgram'");
            }

            // Construct and open the businessobject
            BusinessObject obj = new BusinessObject(objId);
            obj.open(ctx);

            // Get the connected objects
            String relType = "docRel";
            String objType = "docType";
            StringList objSelect = new StringList();
            StringList relSelect = new StringList();
            ExpansionWithSelect exp = obj.expandSelect(ctx,
                                                        relType,
// relationship pattern
                                                        objType,
// type pattern
                                                        objSelect,
// selects on objectes
                                                        relSelect,
// selects on rels
                                                        true,
// getTo direction
                                                        true,
// getFrom direction
                                                       (short)1);
// levels

            // And return the count of related objects
            RelationshipWithSelectList relList =
exp.getRelationships();
            retval = relList.size();

            // Close the object
            obj.close(ctx);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
        return String.valueOf(retval);
        }
    }
```

# Studio Customization Toolkit Tracing

Two sets of interfaces are provided that can be used in your Java application to output tracing messages of any sort to the ENOVIA Live Collaboration trace files. These messages will be output at runtime only if the corresponding trace TYPE has been turned on. The tracing interfaces available to Java programmers are:

- Using the methods of MatrixLogWriter, which may be most appropriate in complex Java applications where the MatrixLogWriter object can be instantiated and maintained to control tracing across a set of interacting contexts/sessions.
- Using the methods of the current Context object. These procedural methods may be most appropriate for use in implementations which do not have a central controlling application, such as JSPs and the Java program objects.

## Considerations

MatrixLogWriter was implemented as a class that might be used to conveniently insert user-defined tracing messages via calls to MatrixLogWriter.write() method, but it has two disadvantages:

- the default constructor for MatrixLogWriter uses a default tracing type (LOGWRITER) and attemptsto direct the output to a default log file (ematrix.log).
- it was implemented with the requirement that it be explicitly turned on/off programmatically.

If you want to leave debugging statements in your implementation code that can be turned on when necessary for diagnosis, it is preferable to insert the trace-message calls in your code with your own specific trace type via:

```
trace type MYTRACE text "Start checkpoint: Temp query re=last";
OR
Context.printTrace("MYTRACE","Start checkpoint: Temp query
re=last");
```

These can then be turned on/off interactively in a running server just like any other trace type, with one of the following:

```
trace type MYTRACE on/filename ....
trace type MYTRACE off
```

## MatrixLogWriter methods

With the object-oriented MatrixLogWriter interface, you turn tracing on by instantiating a MatrixLogWriter object with one of two constructors:

```
public MatrixLogWriter(Context context)

Turn tracing on for all sessions with type=LOGWRITER, filename=Matrix.log

@param Context context: the current active context
@return none

public MatrixLogWriter(Context context, String filename,
 String type, boolean allFlag)
```

```
Turn tracing of specified type on/off for this/all sessions

@param Context context: the current active context
@param String filename: name of file for tracing output
@param String type: type of tracing desired (all caps: MQL, SQL, MINE)
@param boolean onFlag: true to turn trace on, false to turn it off
@param boolean allFlag: true for all sessions; false for this session only
@exception MatrixException when context cannot be established
```

Once it is instantiated, the trace file is open, and writing to it consists of simply calling the write method. The timestamp and label (for the type) will be formatted and prepended to the message.

```
public void write(String message) throws java.io.IOException
public void close() throws java.io.IOException
```

*It is important to remember to call the* `close` *function before destroying the MatrixLogWriter object or allowing it to be destroyed by going out of scope. Failure to do so will result in the process holding onto the open file descriptor, so the file cannot be deleted or renamed until the process (your server) terminates.*

## Context methods

Studio Customization Toolkit interfaces are also provided that correspond to the MQL commands in the preceding section. These interfaces are provided as methods on the current Context object:

```
public void setTrace(String filename,
                     String type,
                     boolean onFlag,
                     boolean allFlag) throws MatrixException

@param String filename: name of file for tracing output
@param String type: type of tracing desired (MQL, SQL,.., MYTRACE)
@param boolean onFlag: true turns trace on, false turns it off
@param boolean allFlag: true traces all sessions;
                        false traces this session only

@exception MatrixException when context cannot be established
@return none
```

As with the MatrixLogWriter interface, it is important to remember to turn all traces off when you have collected the information you require, especially when you are running your traces against a production server. Tracing consumes resources, and unclosed file descriptors are among them.

```
public void printTrace(String type,
                       String message) throws MatrixException

@param String type: type of tracing desired (MQL, SQL,.., MYTRACE)
@param String message: message to be printed to trace file

@exception MatrixException when context cannot be established
@return none
```

## Tracing and Debugging

User-defined tracing should be considered at the time that you are developing your implementation. Keep in mind the following:

- Because tracing can be turned off, you should consider whether you should leave a few judiciously-placed calls to printTrace for production level diagnosis and performance checks. They will be inactive until or unless you explicitly turn on tracing of the corresponding type. The emphasis here is on "few" and "judiciously-placed," since even inactive calls have to do a some checking to discover they are inactive. For example, putting calls to write "entering program" and "exiting program" for every method in a complex client application would be frivolous, since each represents an additional call to the server where the tracing flags are maintained. However, the inclusion of a few tracing calls before and/or after some complex operations can streamline downstream troubleshooting considerably.
- ENOVIA Live Collaboration merges multiple trace types into a single file, and may do so for multiple (all) sessions or threads. The inclusion of explicit TYPE tags in each label allows you to use OS tools to extract records for a single trace type into its own file. Similarly, the inclusion of the thread identifier in each label allows you to extract/segregate the activities of all types for a single session or thread.
- You should also carefully consider the placement of performance traces, since the mere fact of turning them on affects the very thing they are intended to measure. For performance analysis, the fixed format lends itself well to parsing the log files and measuring time spend in certain programs, etc.

## Examples

### Enabling with MQL

The following example shows how to turn tracing on for SQL, MQL and a user-defined tracing type 'MYTRACE', followed by the resulting text file myExample.log:

```
MQL<5>trace type SQL filename myExample.log;
MQL<6>trace type MQL on;
MQL<7>trace type MYTRACE on;
MQL<8>execute program testTrace;
>>>>>>>>>>> myExample.log <<<<<<<<<<<<<<<<<

2001349-22:12:27.740 SQL  t@284 select * from mxVer6 where mxOid=:va
2001349-22:12:27.740 SQL  t@284    :va=1
2001349-22:12:27.740 SQL  t@284 -->select
2001349-22:12:27.740 MQL  t@284 Session: mx3c1d5d9a16234334
2001349-22:12:27.740 SQL  t@284 select * from mxVer6 where mxOid=:va
2001349-22:12:27.740 SQL  t@284    :va=1
2001349-22:12:27.740 SQL  t@284 -->select
2001349-22:12:27.740 MQL  t@284 Program: testTrace
2001349-22:12:27.740 MQL  t@284    args:
2001349-22:12:27.740 MQL  t@284 allocating non-pool TCL interpreter
2001349-22:12:27.940 MQL  t@284 Begin TCL Session: mx3c1d5d9a16234334

2001349-22:12:27.970 MQL  t@284 mql: trace type MYTRACE text Start checkpoint: Temp
query rev=last
2001349-22:12:27.970 MYTR t@284 Start checkpoint: Temp query rev=last
2001349-22:12:27.970 MQL  t@284 mql time 0.000
```

```
2001349-22:12:27.970 MQL  t@284 mql: temp query bus * * * where 'revision == last'
2001349-22:12:27.970 SQL  t@284 select * from mxVer6 where mxOid=:va
2001349-22:12:27.970 SQL  t@284    :va=1
2001349-22:12:27.970 SQL  t@284 -->select
2001349-22:12:27.970 SQL  t@284 select * from lxBO_4a7a4a03
2001349-22:12:27.970 SQL  t@284 -->select
…
2001349-22:12:27.980 SQL  t@284 -->select
2001349-22:12:28.060 SQL  t@284 select * from lxBO_9faf941c
2001349-22:12:28.080 SQL  t@284 -->select
…
2001349-22:12:28.160 SQL  t@284 -->select
2001349-22:12:28.521 SQL  t@284 select * from lxBO_77298049
2001349-22:12:28.521 SQL  t@284 -->select
2001349-22:12:28.561 MQL  t@284 mql time 0.591
2001349-22:12:28.571 MQL  t@284 mql: trace type MYTRACE text End checkpoint: After temp
query
2001349-22:12:28.571 MYTR t@284 End checkpoint: After temp query
2001349-22:12:28.571 MQL  t@284 mql time 0.000
2001349-22:12:28.571 MQL  t@284 TCL Results:
2001349-22:12:28.571 MQL  t@284 END TCL time: 0.631 (0.591 in MQL)

2001349-22:12:28.571 MQL  t@284 End Program:
```

### Tracing with Tcl

The mql commands used in the following Tcl program also serve as an example for how the trace commands could be used in an interactive mql session or emxRunMQL.jsp to debug some implementation programs (Java or Tcl) or troubleshoot some behavior in a server setting.

```
#
# Purpose:  Example of tracing control from a tcl program
#

tcl;
eval {

    # Check to see if any tracing is currently on
    set lPrint [split [mql print trace] \n]
    set sEQ "="
    set sAllTraceTypes [string trim [lindex [split [lindex $lPrint 1] $sEQ] 1]]
    set sAllTraceDest  [string trim [lindex [split [lindex $lPrint 3] $sEQ] 1]]

    # Output message that some tracing is being turned off
    if {$sAllTraceTypes != ""} {
        puts "Turning off $sAllTraceTypes trace $sAllTraceDest"
    }

    # Turn all tracing off
    mql trace off
    # Turn on mql and verbose tracing to a file
    mql trace type mql,verbose filename mql-verbose.log

    # This executable will place
    # Mql/verbose trace messages into mql-verbose.log until some program/user
    # turns this tracing off via a program or manual key-in.
    catch {mql print context}
    catch {mql exec prog j}
    catch {mql print trace}

    # Turn the tracing off
    mql trace off
```

### Tracing with Context.setTrace(), Context.printTrace()

Similarly, this example shows how to use Context::setTrace and Context::printTrace. It shows the relationship of the single-session and all-session targets:

```
import matrix.db.*;
import matrix.util.*;

public class ${CLASSNAME}
{


    public ${CLASSNAME}(Context ctx,String[] args)
    {
    }
    public int mxMain(Context ctx,String[] args)
    {

        if (args.length < 1 || !(args[0].equals("all") || args[0].equals("thread"))) {
            System.out.println("\nNeed argument: all or thread\n");
            return 1;
        }
        try
        {
            // capture choice - all thread tracing or single thread tracing
            boolean allflag = args[0].equals("all");

            // Check to see if any tracing is currently on
            MQLCommand mql = new MQLCommand();
            mql.executeCommand(ctx, "print trace");
            System.out.println("Print trace:\n" + mql.getResult());

            // Turn tracing off
            ctx.setTrace("","*",false,true);
            System.out.println("\nTurn trace off");
            mql.executeCommand(ctx, "print trace");
```

```
                System.out.println("Result: " + mql.getResult());

                // Turn on tracing to a file
                // If allflag == true
                // (including this thread) will place mql/verbose trace messages
                // into mql-verbose.log until some program/user turns it off
                if (allflag)
                    ctx.setTrace("mql-verbose.log", "mql,verbose", true, allflag);
                else
                    ctx.setTrace("TRUE", "mql,verbose", true, allflag);

                mql.executeCommand(ctx, "print context");
                System.out.println("\nprint context:\n" + mql.getResult());
                mql.executeCommand(ctx, "exec prog j");
                System.out.println("\nexec prog j:\n" + mql.getResult());
                mql.executeCommand(ctx, "print trace");
                System.out.println("\nprint trace:\n" + mql.getResult());

                // Turn the tracing off
                ctx.setTrace("mql-verbose.log", "mql,verbose", false, allflag);

                System.out.println("\nTurn all-thread trace off");
                mql.executeCommand(ctx, "print trace");
                System.out.println("Result: " + mql.getResult());
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
            return 0;
        }
    }
```

## Studio Customization Toolkit Session Monitoring

Sessions that are connected to ENOVIA Live Collaboration via the Studio Customization Toolkit can be monitored to report information such as user names, cache sizes, object IDs, and the like. These statistics are useful for debugging Studio Customization Toolkit processes and monitoring memory usage.

The following features are available for monitoring Studio Customization Toolkit sessions:

- `MX_VERBOSE_TRACE` variable
- `monitor context` MQL command (ENOVIA Live Collaboration server only)
- `monitor memory` MQL command

## Verbose Tracing

MX_VERBOSE_TRACE when set to true can show key Studio Customization Toolkit call parameters in verbose tracing. On Windows, you add this variable to the ematrix.ini file; on UNIX, add and export it as an environment variable to rmireg.sh.

When you set MX_VERBOSE_TRACE to TRUE or matrix.log (your log file name), tracing output includes additional Studio Customization Toolkit call parameters, such as:

- The user logging in, which is identified in entries including allocExternalContext.bosInterface and reset.bosContent.
- Entries listing “input params” and “output params”, which contain parameter information such as user, vault, object name, language, command, etc.
- “stateful dispatch” messages, which are followed by a report of the session name.
- The open.bosBusinessObject call, which also reports the input object-id and output TNR.
- The open.bosQuery call “input params” message, which also reports query details.
- The evaluate.bosQuery call “output params” message, which reports the number of objects returned.

The above information can be used to identify specific Studio Customization Toolkit calls that cause problems within the core.

The example below reports information about input params and the session ID:

```
13:59:12.977 VERB t@1536 stateful dispatch for reset.bosContext
13:59:12.977 VERB t@1536   input params:
(session=EhTsDAjwjvlZyTMcLf2HhVYYwkDH1E0WPultfTZU9gVl9g8OJh12!-327757469!1155650348789
), user=Test Everything, passwd=, lattice=
13:59:13.289 VERB t@1536   output params: returnVal=Test Everything
13:59:13.289 VERB t@1536 dispatch complete
```

The example below reports a business object being opened, including input and output params along with the object ID of the business object being opened:

```
01:27:06.359 VERB t@2484 stateless dispatch for openTNRV.bosBusinessObject
01:27:06.359 VERB t@2484 allocate context for session
DG9SzH1NDJJphkGvlySnrSb3Qxc8dvRtLzklJnfThGJG2xyRGVGn!744420327!1128562002281:mx1128562
0262346500949
01:27:06.359 VERB t@2484   input params: name=MCADInteg-LocalConfig, type=02950905,
rev=1, vault=
01:27:06.359 VERB t@2484   output params: returnVal objectid=50203.13182.29436.6347
01:27:06.359 VERB t@2484 dispatch complete
```

The following example of a query evaluation reports input parameters that describe the scope of the query, and an output parameter that indicates the number of objects returned for the query. (If the timestamps indicate the process took a long time, a large number of objects returned could explain the long timespan.)

```
01:27:23.296 VERB t@2484 stateless dispatch for evaluate.bosQuery
01:27:23.296 VERB t@2484 allocate context for session
PUFpXp1kjpqvcIIxhXAFhQqZdNlR89L2BVFbZjNn2l3qufgw28kt|-2636696131106167120/167839130/6/
7001/7001/7002/7002/7001/-1
01:27:23.296 VERB t@2484   input params: name=, query type=Part,
name=*, revision=*, lattice=*, owner=*, where=, limit=1001, expandTypes=true
01:27:23.296 VERB t@2484   output params: returnVal objects=176
```

```
01:27:23.296 VERB t@2484 dispatch complete
```

Verbose tracing output can include the caller information appended to the context session id generated by Framework.getFrameContext(). Following is an example illustrating this-note that the caller information **`emxteamsearchcontentresult.java`** is appended to the string.

```
13:46:13.390 VERB t@1776 stateless dispatch for evaluateSelect.bosQuery
13:46:13.390 VERB t@1776 allocate context for session
B1yyHxgoYeuyPHpaAPAH2qLIBGplw42XKR5B228B2Oki2UJNLuKH!-682390697!167842110!7001!7002!11
02426802906:mx1102427172562633055:(__emxteamsearchcontentresult.java:3628)
13:46:13.390 VERB t@1776   input params: name=, query type=Document, name=*,
revision=*, lattice=eService Production, owner=*, where=(revision == last) &&
(("to[Vaulted Objects].from.id" !~~ "zz")) && (current.access[read] == TRUE), limit=0,
expandTypes=true
, objectSelect length=2
13:46:13.437 VERB t@1776   output params:
```

## Monitoring Context Objects

The `monitor context` MQL command is used to count and list currently registered bosContext objects, which map to Studio Customization Toolkit Context objects and give the core statistics:

```
monitor context [SESSION-ID] [set|!set] [terse];
```

`SESSION-ID` is used to limit the display of session information to the context object for the specified session ID. If not used, all sessions are reported.

The `[set|!set]` option limits the display of session information to contexts that are marked as “set” or “not set” respectively. (The Studio Customization Toolkit `context.shutdown()` method will mark a context “not set.”) Use this option only when the session ID is not specified.

`terse` displays only cumulative statistics and ENOVIA Live Collaboration statistics, not individual sessions.

The `monitor context` command can display the following information per session:

- number of contexts
- idle vs. active session
- total cached bytes
- session ID
- context set/logged in status
- username
- timestamp for and name of last Studio Customization Toolkit call and thread on which it executed
- transaction status (for example: active, mode, savepoints, etc.)
- estimate of context-specific cache size in use, if possible, for each active session

The command displays the following environment information for all sessions:

- Total number of sessions
- Total session cache size
- Pooled session cache size

Following is sample output for the `monitor context` command, taken using an Studio Customization Toolkit program that mimics the MQL functionality. Note that the string named after the session name `(current)` indicates the context corresponding to the current user session.

```
mql>monitor context
Pooled session cache: 0 bytes

4 context objects

Session PUF93l21k9lAjAH0hy0O2WOZCYOhggu2dvWd0owsfT9DDHzH1I5P|
-263669613110616712
0/167839130/6/7001/7001/7002/7002/7001/-1
    User:   'Test Everything' logged in
    Vault:  'eService Sample'
    Last:   t@2208, select.bosBusinessObject
    Last recorded cache size: 0
    idle:   14 minutes 52 seconds
    0 active sessions

Session mx1027692483622676970837 (current)
    User:   'creator' logged in
    Vault:  'ADMINISTRATION'
    Last:   t@2104, executeCmd.bosMQLCommand
    Last recorded cache size: 0
    Idle: 0 seconds
    1 active session
      session 0
        transaction active,readonly,wait
        0 cached entries, 0 bytes

Session mx10277030735012117155733
    User:   'creator' logged in
    Vault:  'ADMINISTRATION'
    Last:   t@1940, executeCmd.bosMQLCommand
    Last recorded cache size: 505579
    idle:   5 seconds
    2 active sessions
      session 0
        transaction active,update,wait
          savepoint save1
        3 cached entries, 3192 bytes
      session 1
        transaction active,update,wait
        3 cached entries, 502387 bytes

Session mx1027703338082-1990050644
    User:   'creator' logged out
    Vault:  'ADMINISTRATION'
    Last:   t@2284, executeCmd.bosMQLCommand
    Last recorded cache size: 0
    idle:   11 minutes 39 seconds
    0 active sessions

Total cache size: 505579 bytes
```

*Only users with System Administrator privileges can execute the* `monitor context` *command.*

## Notes

- If a context is currently executing at the time another context issues the `monitor context` command, output for the active session will resemble the following:

```
Session PUGpAOiBm3OcnMK5Bk27QcQYjcm2iq2UwMQzxh9KGjTTddp52xkr|-263669613110616712
0/167839130/6/7001/7001/7002/7002/7001/-1
    User:   'Test Everything' logged in
    Vault:  'eService Sample'
    Last:   t@1956, executeCmd.bosMQLCommand
    Active: 2 seconds
    Last recorded cache size: 0 (update requested; reissue monitor context command)
*** Cannot report session stats - session is active ***
```

  The above sample shows the “Active” time for the session, and a warning message states “…session is active.”

- In the interest of thread safety, monitor context processing reports only what is safe to report. Access to cache and transaction information during monitor context processing is done in a thread-safe fashion so it does not impact system performance and stability of other sessions and the system as a whole.

# 3

# Introduction to the Studio

## Overview

The ENOVIA Live Collaboration Studio Customization Toolkit provides documentation, source code, sample files, and wrapper code that may be used as a reference for custom programming. You can use the classes available in the applet or servlet jar files that are installed with the core software (eMatrixAppletXML.jar and eMatrixServletRMI.jar). When you install the Studio Customization Toolkit, you get the javadoc for every class that is included in all of these .jar files, as well as source code that can be used as examples, and in fact are used as examples in this guide. For information about installing the Studio Customization Toolkit, see the *ENOVIA Live Collaboration Installation Guide*.

When you develop and compile against one of the servlet jar files, you can produce applications that use Java servlets, JPOs and JavaServer Pages (JSPs) to communicate with the ENOVIA Live Collaboration database. The ENOVIA products (ENOVIA Supplier Central, etc.) are good examples of these types of applications that use the Studio Customization Toolkit. To build such an application, you first install the version of the ENOVIA Live Collaboration server you will run the programs on, and compile your code against the server's jar file. For example, if you are running the ENOVIA Live Collaboration server, you need to compile against eMatrixServletRMI.jar.

If you want to build any other kind of client application you would use the classes in eMatrixAppletXML.jar. Additionally, if you are using the ENOVIA Live Collaboration server without an application server and want to build client-only applications (an application with no servlets or JSPs), you can use eMatrixServletRMI.jar. In the case of the ENOVIA Live Collaboration server, this is sometimes referred to as "talking directly to Live Collaboration server." You can compile this kind of code against either the eMatrixServletRMI.jar, or eMatrixAppletXML.jar. (A servlet-enabled Web or application server is required to run servlets and JSPs.)

C++ classes (eMatrixMQL) are also included when you install the ENOVIA Live Collaboration Studio Customization Toolkit. The eMatrixMQL.h file provides the C++ programmer with the same functionality as the Java Client package provides to the Java programmer. Refer to *C++ Studio Customization Toolkit* in Chapter 8 for additional information.

## ENOVIA Live Collaboration Java Class Packages

ENOVIA Live Collaboration .jar files include a set of Java class packages, which can use to build applications. The .jar files include some other packages that are used at runtime but are not used to build and that should not be called directly. These packages are not listed here and are not included in the Javadocs. The packages you can use to build applications are described below.

### ENOVIA Live Collaboration Java class packages common to eMatrixServlet and eMatrixApplet jar files

- **db Package**
  The db package consists of Java classes that implement and access ENOVIA Live Collaboration database objects. Database objects can be categorized as:
    - Administrative objects, which include attribute, form, format, group, page, person, policy, program, relationship, report, resource, role, and type objects.
    - Personal visual objects, which include query, set, cue, view, filter, toolset, and table objects.
    - Instance objects, which include business objects and relationships.
- **Resource Package**
  The Resource package contains classes representing .gif images used in the ENOVIA Live Collaboration application.
- **VUI Package**
  The VUI package provides Java binding for user interface components.
- **Util Package**
  The Util package provides Java binding for utility functions.
- **Resource Package**
  The Resource Package contains classes representing `.gif` images used in the ENOVIA Live Collaboration application. The full source code for this Package is provided as part of the ENOVIA Studio Customization Toolkit.

### eMatrixServletXXX.jar packages

- **Servlet Package**

The Servlet package contains the ENOVIA MatrixOne servlets: BusinessObjectServlet, BusinessTypeServlet, FileCheckinServlet, FileCheckoutServlet, FileUploadServlet, Framework, FrameworkServlet, LoginServlet, LogoutServet, MatrixExchangeServlet, MatrixXMLServlet, ProgramServlet, ResourceServlet, WizardServlet. These servlets are described in Chapter 6, *Servlets* and examples of JSPs that use the servlets are shown throughout this guide.

### eMatrixAppletXML Packages

- **Client Package**

  The Client Package provides Java binding of client functions that communicate with the ENOVIA Live Collaboration server.
- **Common Package**

  The Common Package consists of Java classes that implement common dialogs such as choosers. The full source code for this Package (including Class code and Method code) is provided as part of the Studio Customization Toolkit.
- **ENOVIA Live Collaboration Package**

  The ENOVIA Live Collaboration Package consists of Java classes that implement application dialogs. The full source code for this package is provided as part of the Studio Customization Toolkit.

## Inventory of Provided Source Code

The Studio Customization Toolkit provides source code for the following packages:

- **The Resource Package**

  Code is provided for .gif images that are included in various elements of the ENOVIA Live Collaboration user interface, such as images that make up buttons and icons and images that go on toolbar buttons.
- **The VUI Package**

  Code is provided that implements Java binding for generic user interface components, including viewers, buttons, menus, icons, and toolbars.
- **The Util Package**

  Code is provided for various utility functions, such as error message and problem reporting, searching, listing, and so on.
- **The Common Package**

  Code is provided that implements common dialogs such as choosers and pattern windows: vault chooser, user chooser, type chooser, select pattern windows, and so on.
- **The Matrix Navigator Package**

  Code is provided that implements the Matrix Web Navigator. This applet can start the main application dialogs such as the flat browser, star browser, indented browser, state browser, basics, attributes, session context, find, formats, open set, and so on.
- **Applet Package**

  Code samples are included for eight applets. Each one implements an eMatrixwindow within an HTML page, with the ENOVIA Live Collaboration interface hidden from the user.

  For each applet, the classes, Java code, and an HTML page are provided. Applets are included to implement the following:

- Attribute Dialog
- Basics Dialog
- FlatQuery
- FlatSet
- Formats Dialog
- GetAttributes
- Indented Browser
- SimpleFind
- Star Browser
- State Browser
- StatesPopup

Source code for Matrix Web Navigator is also included as a sample. Parts of its code are used throughout this guide as examples to demonstrate the key features of these packages and how you can use them to build applications.

- **MQL Package**

  This applet displays an MQL screen in a Web page. Any valid MQL command may be executed, including those that update the database, such as create, approve or promote. When commands that involve external resources are executed, those resources exist on the ENOVIA Live Collaboration server, not the client where invoked. For example, the MQL command:

  ```
  print bus Assembly "MTC 12345" A select description
  current dump "|" output 12345.txt
  ```

  will create the text file on the ENOVIA Live Collaboration server host machine in the Live Collaboration server's directory, not on the machine running the applet. Similarly, external program files must reside on this host in this directory if they are to be executed from the MQL applet.

## Java Packages

A package is a group, or collection, of related Java classes, similar to a C++ library. A package is a convenient way of collecting methods, objects, or components, and bundling them together.

The packages included in the eMatrixAppletXML are:

- the Client package – a collection of client methods
- the Resource package – a collection of `.gif` image classes
- the DB package – a collection of database objects
- the VUI package – a grouping of user interface components
- the Util package – a collection of utility methods
- the Common package – a grouping of common dialogs
- the ENOVIA Live Collaboration package – a grouping of application dialogs

In general, the Java class packages should be set up as follows:

```
company/classes/product/package/*.class
```

where:

| Path Element | Definition |
|---|---|
| `company` | The company name. |
| `/classes` | Indicates that the directories below this contain class files. |
| `/product` | The application product name. |
| `/package` | These subdirectories are named for the class packages and they contain the class files. |
| `/*.class` | The various class files. |

The Matrix classes are provided in the following structure:

```
matrix/classes/matrix/client/*.class
matrix/classes/matrix/resource/*.class
matrix/classes/matrix/db/*.class
matrix/classes/matrix/vui/*.class
matrix/classes/matrix/util/*.class
matrix/classes/matrix/common/*.class
matrix/classes/matrix/matrix/*.class
```

You should customize the structure to indicate your company and product, and any packages that you create. For example:

```
acme/classes/widget/client/*.class
acme/classes/widget/db/*.class
acme/classes/widget/vui/*.class
acme/classes/widget/util/*.class
acme/classes/widget/common/*.class
acme/classes/widget/matrix/*.class
acme/classes/widget/acmepackage1/*.class
```

You should create a similar structure for your Java source code. Replace the *classes* subdirectory with *source* so that the directory structure for your Java source code will be familiar. The source Java files contain the methods in the classes used for the implementation of the classes. For example:

```
acme/source/widget/client/*.java
acme/source/widget/db/*.java
acme/source/widget/vui/*.java
acme/source/widget/util/*.java
acme/source/widget/common/*.java
acme/source/widget/matrix/*.java
acme/source/widget/acmepackage1/*.java
```

## Compiling Java Classes

When you compile using your Java class packages, you must specify the exact order in which the compiler is to search for class files. This is typically done through the environment variable `CLASSPATH`. Alternatively, if you are working in a development environment like Symantec Cafe, `CLASSPATH` may be modified in the project file settings.

`CLASSPATH` is an environment variable that tells the compiler the order in which to search for packages and classes. ENOVIA Live Collaboration classes are contained in the file `ENOVIA_INSTALL/studio/java/classes/eMatrixAppletDownloadXML.jar`, which can be used to compile against. For example, use the following `CLASSPATH` statement:

```
CLASSPATH = ENOVIA_INSTALL/studio/java/classes/
eMatrixAppletXML.jar
```

If multiple .jar files exist, use `CLASSPATH` to specify the search order. For example, if there are two files called *widget.class* (one in the acme.jar file and one in the eMatrixAppletDownloadXML.jar file), and the one in acme.jar is the one you want to use, you must indicate this in the `CLASSPATH` statement. To search the acme.jar file first, specify the *`acme`* path before the *matrix* path in the `CLASSPATH` statement:

```
CLASSPATH = /acme/classes/acme.jar;/ENOVIA_INSTALL/studio/java/
classes/eMatrixAppletXML.jar;...
```

The compiler will search the *acme* path first and locate the *widget.class* file in the acme.jar file. The *widget.class* in eMatrixAppletDownloadXML.zip will not be used.

## Programming with or without the Business Process Services

If you have any ENOVIA products—such as ENOVIA Supplier Central, ENOVIA Program Central, ENOVIA Variant Configuration Central, etc.— then you have the framework because it is required for the applications. It's also possible to have the framework only and build custom applications to work with the framework.

The examples used within this guide do not rely on the use of the Application Exchange Framework.

### Advantages of Using the Framework

In most cases, the framework makes the job of creating custom JSPs much easier. The framework provides:

- a common schema for applications, including common data models, business rules, and processes
- a standard user interface that includes menus and graphics
- mechanisms that control which menus and options are displayed for a user based on the user's assigned roles
- mechanisms for looking up objects even if the object names have changed, for managing multiple trigger events, and for automatically naming objects as they are created
- templates for JSP pages that perform common functions needed for all framework JSPs, such as presenting the login page if the user is not logged in, importing Java packages, and displaying the standard user interface
- utility trigger programs that are commonly needed, such as a program that enters the name of an object's originator in an Originator attribute and programs that check for required connections, files, or states
- wizards that let you add new suites, applications, and tasks, assign access to tasks, and merge existing schema with the framework schema
- sample data based on use cases; helpful for training and testing
- an installation program that prevents problems with object name collisions, tracks the version of objects so only updated objects are installed, and records details to a log file
- many ways to customize the framework to fit your business rules and applications

For details about all these features, see the *ENOVIA Live Collaboration Application Development Guide* and *ENOVIA Business Process Services Administrator's Guide* .

### If You Have the Framework

If you have the framework, you can use this guide to help you write custom JSPs and servlets using the ENOVIA Live Collaboration servlet Studio Customization Toolkit. However, keep in mind that the framework supplies a common interface and uses specific conventions. To make sure your JSPs perform the common functions needed for every JSP in the framework, you should use the template JSPs that are installed with the Application Exchange Framework. Also, to make a new suite, application, or task appear in the Business application menus, you'll need to register them and assign access to roles. For instructions on how to program JSPs for use with the framework, see the *Application Exchange Framework Programming Guide*.

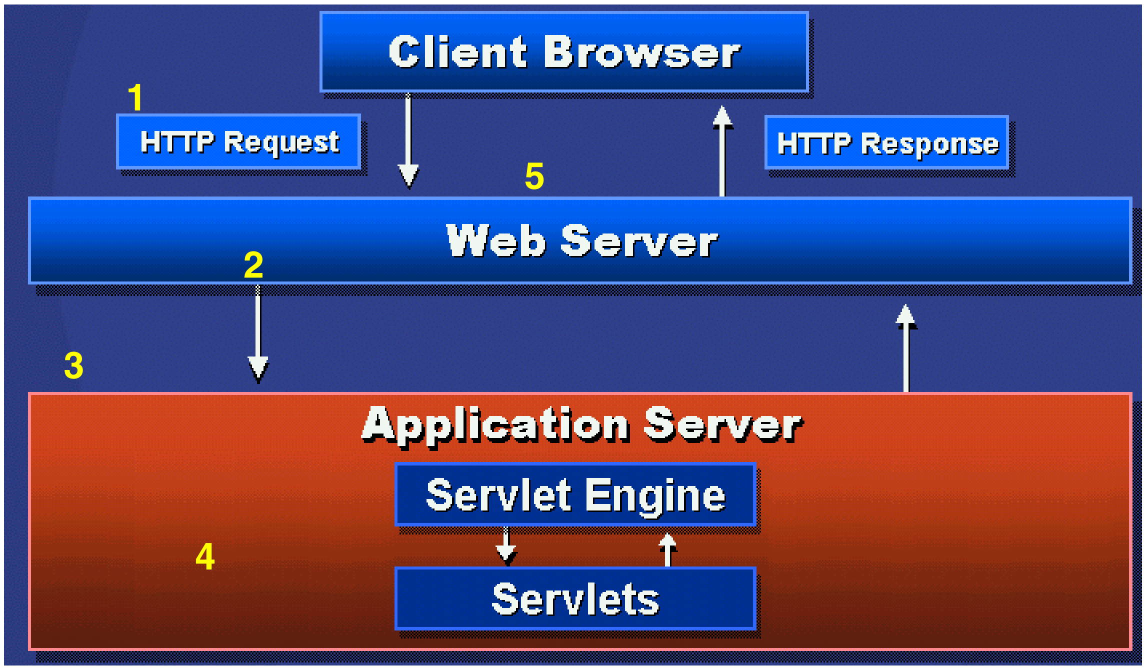
## Overview of Servlets

Similar to the way Java applets extend the functions of a browser, Java *servlets* extend the functions of Web/application servers. Servlets run on and are managed by servers. They are precompiled Java programs that receive HTTP requests from a browser. Servlets then build and send back an HTTP response to the browser. Generally, the response produces HTML screens with dynamic data. For example, the data could be updated based on user input and based on information from the ENOVIA Live Collaboration database.

The ENOVIA Live Collaboration Servlets are installed as part of the ENOVIA Live Collaboration server (eMatrixServletRMI) and directly call the Studio Customization Toolkit, as is the case with the other parts of the server. Servlets must be registered with the Web/application server.

### Servlet Processing

The following graphic illustrates the processing that occurs when a servlet is requested from a browser. Each numbered point is described below. (The servlet engine could be part of the Web server and not use an application server).

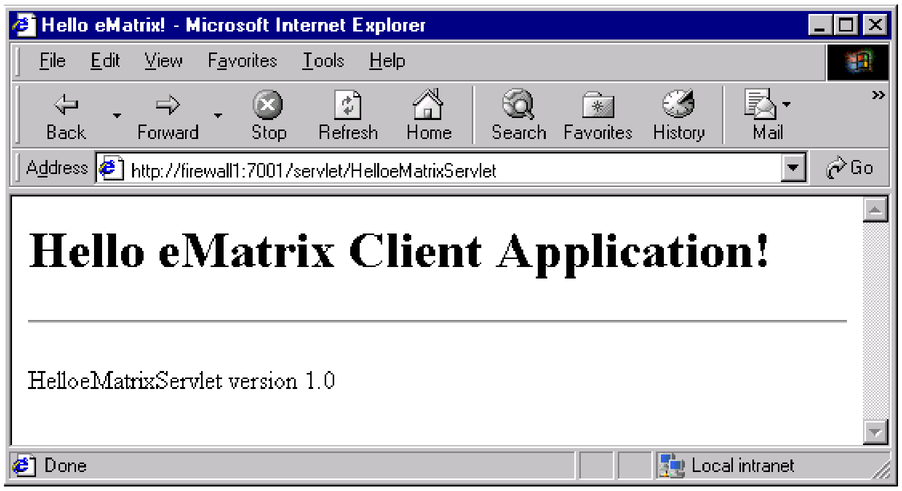


1. A browser requests the URL (alias) for a servlet, such as http://HOSTNAME:PORT/enovia/servlet/login. This request could come from within a JSP, another servlet, an HTML page, or directly from the URL.
2. The Web server (WebLogic, IBM HTTP, Apache, IIS) recognizes the servlet as a program that should be passed to the application server because servlets are registered with the application and Web server.
3. The Web server forwards the request to the application server (WebLogic, WebSphere, etc.), which manages the servlet engine.

4. The servlet engine converts the browser request into a HTTPServletRequest Java object, which is passed to the servlet along with an HTTPServletResponse object that is used to send the response back to the browser.
5. The servlet then generates the response to the browser, which is typically HTML and usually includes dynamic content generated by the servlet processing.

## Example of a Simple Servlet

To see how a servlet is created, let's look at a very simple example. The graphic below shows the output of a servlet.


Hello eMatrix! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address
Go
Done
Local intranet

http://firewall1:7001/servlet/HelloeMatrixServlet

**Hello eMatrix Client Application!**

HelloeMatrixServlet version 1.0

Here is the code that produced the above output. The code is saved in a file called HelloeMatrixServlet.java, which is included with the files for the sample application.

```
1  import java.io.*;
2  import javax.servlet.*;
3  import javax.servlet.http.*;

4  public class HelloeMatrixServlet extends HttpServlet
   {
5    protected void doGet(HttpServletRequest req,
6                         HttpServletResponse res)
7              throws ServletException, IOException
    {
8      res.setContentType("text/html");
9      PrintWriter out = res.getWriter();
10     out.println("<HTML><HEAD><TITLE>Hello eMatrix!</TITLE>"+
11       "</HEAD><BODY><h1>Hello eMatrix Client Application!</h1>"+
12       "<hr>"+
13       "<p>HelloeMatrixServlet version 1.0</p></BODY></HTML>");
14     out.close();
     }
   }
```

Lines 1 to 3 import packages that contain classes used by the servlet. Almost every servlet needs classes from these packages.

Line 4 declares the servlet. Our servlet extends javax.servlet.http.HttpServlet, the standard base class for HTTP servlets.

Lines 5 through 7 implements the HttpServlet's `doGet` method. A servlet that extends the HttpServlet must implement the method (or methods) that handle the HTTP interactions the servlet is designed to handle.

Line 8 uses the setContentType method of the HttpServletResponse object to set the content type of the response that will be sent. You must set the content type before using a PrintWriter object to write the response to the user.

Line 9 requests a PrintWriter object and uses the `getWriter` method to return the response data to the user. The `getWriter` method returns text data. Alternatively, you can use the `getOutputStream` method for binary data.

Lines 10 through 13 use the PrintWriter object to write the text of type text/html (as specified through the content type).

Line 14 closes the PrintWriter.

## Accessing the Servlet

The sample servlet described above is saved as a Java file, HelloeMatrixServlet.java. To run the servlet, you must:

- Compile the servlet, as you would any Java file, to produce a *.class file. Make sure the packages that are imported in the servlet, javax.servlet and javax.servlet.http, are included in the CLASSPATH environment variable. If you don't have the classes, install them separately by downloading the JSDK (Java Servlet Development Kit, http://java.sun.com/products/java-server/servlets/index.html#sdk).

  For example, on Windows, compile the servlet by entering the following command at the command prompt:

  ```
  javac HelloeMatrixServlet.java
  ```

  The result is a HelloeMatrixServlet.class file.
- Copy the class file to the appropriate location within your Web or application server directory structure. For example, c:\weblogic\myserver\servletclasses.
- From your Web browser, access the servlet by requesting the URL for the servlet. For example, http://SERVER_HOST_NAME:PORTNUMBER/servlet/HelloeMatrixServlet.

# Overview of JavaServer Pages

JavaServer Pages (JSP) provide a simple programming method to display and dynamically update content on a Web page. JSPs are actually converted to servlets “behind the scenes” so coding of a JSP is simpler than coding for a servlet. JSPs are a combination of HTML, JavaScript, and Java code.Whereas HTML is basically a document display format, JSPs provides the processing power that lets you create complex client/server applications for the Web and include dynamic substitutions within an HTML page.

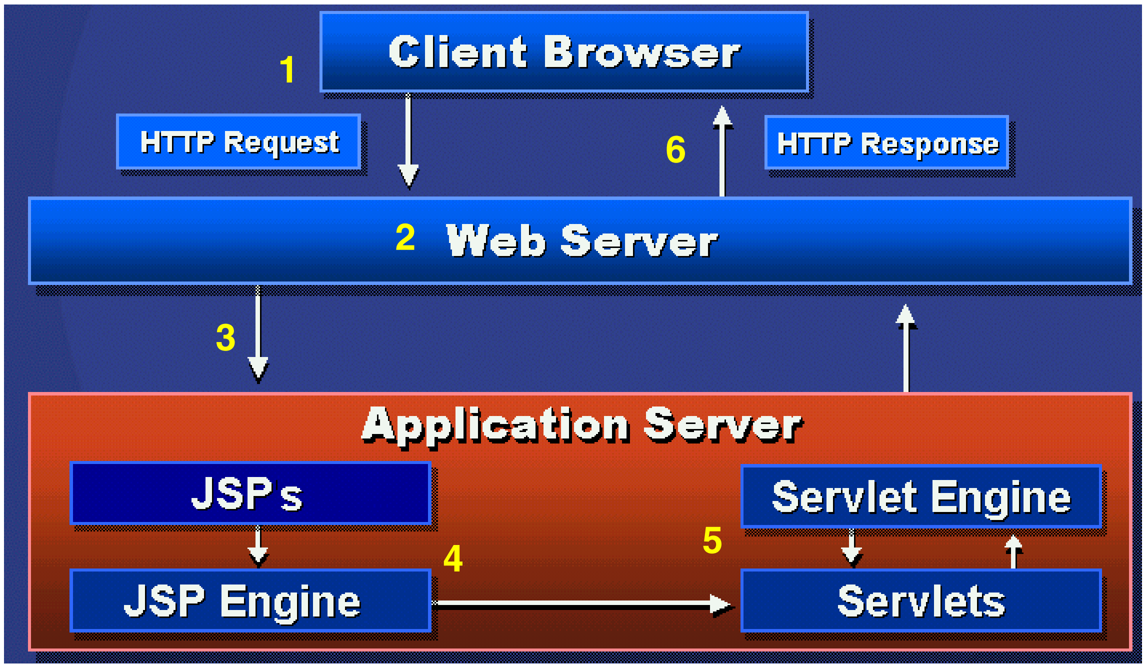
JSP is an extension to the Java servlet technology from Sun, but is much different from ordinary Java in that JSPs are dynamically compiled and interpreted on the JSP-enabled application server. With Java servlets, you need a pre-compilation step, where you compile the text into bytecode which makes it more compact, faster to download and faster to interpret. Then the bytecode still needs to be compiled and deployed. With JSP, the text of the Java code is interpreted and deployed immediately in one step. You can edit JSPs in pure text and instantly see the changes in the Web browser, just as you can with pure HTML.

In ENOVIA Live Collaboration, JSPs may be stored in the database as page objects or on disk. They are delivered through HTTP in response to the browser. JSPs directly call the business object interface of the Studio Customization Toolkit, using that interface to leverage RMI (Remote Method Invocation) or ENOVIA Live Collaboration server, which allows software components stored in the network to be run remotely.

JSPs can also include fragments from other page objects for reusable content. There are many reusable pieces in pure HTML that need to be referred to in multiple pages in an application: the navigation bar, parts of the action bar, the background image, the pieces of table definition that help organize the menu items, etc. The fragments of HTML can be broken up into separate JSPs, and an include technique can be used to dynamically include the page objects that are needed. In this way, a change to an individual component, for example the navigation bar, can be made in a single resource and the whole application will respond because they are all using that shared fragment. This provides the ability to dynamically model and change the look and feel of applications.

## JSP Processing

The following graphic illustrates the processing that occurs when a JSP is requested from a browser. Each numbered point is described below.



1. A browser requests the URL for a JSP, such as http://HOSTNAME:PORT/enovia/eServiceLogin.jsp.
2. The Web server (WebLogic, IBM HTTP, Appache, IIS) recognizes the .jsp extension in the URL, identifying the request as a JavaServer Page which must be handled by the JSP engine.
3. The Web server forwards the request to the application server (Jrun, WebLogic, WebSphere, etc.), which manages the JSP translator and servlet engine.
4. The page is translated into a Java source file and then compiled into a servlet class file. The compilation and translation phase only occurs when the JSP is first called (and the server finds no corresponding class files for the JSP) or when the source JSP changes.
5. The servlet engine converts the browser request into a HTTPServletRequest Java object, which is passed to the servlet along with an HTTPServletResponse object that is used to send the response back to the browser.
6. The servlet then generates the HTML-only response to the browser, including any static HTML included in the JSP and any dynamic content generated by the servlet processing. If you view the source of a JSP from your browser, you will see only HTML content. All Java and Javascript has been converted to static HTML.

## Example of a Simple JSP

To illustrate the difference between a servlet and JSP, we've created a JSP that produces the same output as shown in the Hello eMatrix servlet, shown and described previously. The <% and %> mark the beginning and the end of the embedded Java. The remaining code is straight HTML.

```
1    <%-- HelloeMatrix.jsp   -- displays "Hello eMatrix Client Application"

2    --%>

3    <HTML>
4       <HEAD>
5          <TITLE>Hello eMatrix!</TITLE>
6       </HEAD>
7       <BODY>
8          <h1>Hello eMatrix Client Application!</h1>
9       </BODY>
10   </HTML>

11   <%@include file = "ServletInfo.html"%>
```

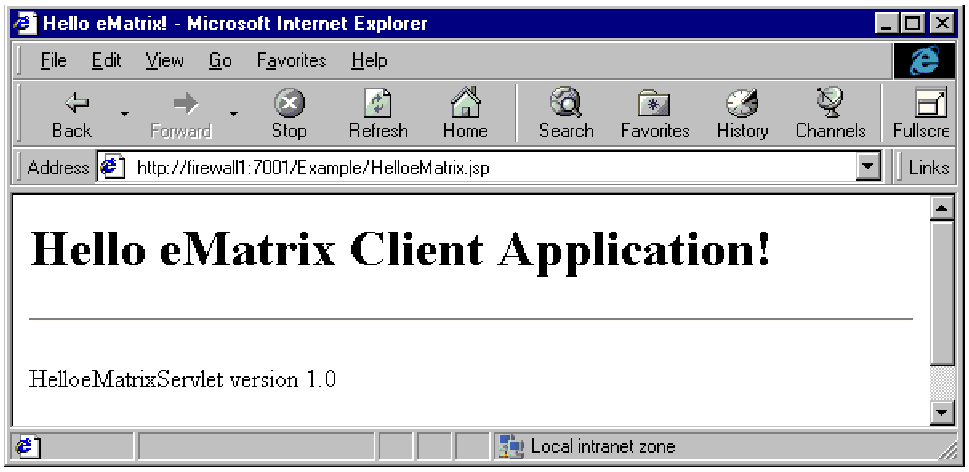
Lines 1 and 2 are hidden JSP comments. These comments are ignored by the compiler and therefore do not appear in the HTML source in the browser.

Lines 3 through 10 contain standard HTML to produce the title and main text.

Line 11 uses an include directive to include the ServletInfo.html page. The ServletInfo.html contains only two lines to make up the same footer information displayed in the sample servlet:

```
<hr>
<p>HelloeMatrixServlet version 1.0</p>
```

The output for the JSP looks the same as the servlet output.



http://firewall1:7001/Example/HelloeMatrix.jsp

Hello eMatrix Client Application!

HelloeMatrixServlet version 1.0

The HTML source for the JSP contains only HTML code. All Java and Javascript is replaced with static HTML. Also notice the hidden comments are not included.

```
<HTML>
   <HEAD>
      <TITLE>Hello eMatrix!</TITLE>
   </HEAD>
   <BODY>
      <h1>Hello eMatrix Client Application!</h1>
   </BODY>
</HTML>
<hr>
<p>HelloeMatrixServlet version 1.0</p>
```

Another way to produce the same output using JSP is to call the Hello eMatrix servlet from a JSP:

```
<%-- HelloeMatrix1.jsp   -- displays “Hello eMatrix Client
Application” by calling the Hello servlet--%>
<html>
<!-- Forward to a servlet using jsp forward statement-->
<jsp:forward page="/servlet/HelloeMatrixServlet" />
</html>
```

## Accessing the JSP

To access the JSP file, place the file, including any associated resource files, within the doc root directory structure of your Web server. Then request the URL for the JSP. For example, if the HelloeMatrix.jsp file is located in a directory called “Example” in the doc root, the URL would be: WEB_SERVER_DOC_ROOT/Example/HelloeMatrix.jsp.

## Using the Custom JSP Filter

The ENOVIA Live Collaboration Server contains a filter class that allows requests from ENOVIA products to check for a prefixed file name for custom JSP pages. You can store custom resources (JSP, HTML, and image files) with prefixed filenames to prevent them from being overwritten on subsequent application installs. You can even define multiple prefixes in order to keep various customization, accelerator, or integration files easy to identify.

*You can also implement a custom directory to store custom files; however, when using custom directories, there are certain jsp directives that will result in run-time (vs. compile time) errors if the relative path is not correct. Therefore, it is recommended that custom prefixes are used instead of a custom directory.*

Implementing the custom JSP filter involves simple changes to the web.xml file for your application server’s WAR file. Refer to the *ENOVIA Live Collaboration Installation Guide* for details.

## Getting More Information on Servlets and JSPs

For more information on the advantages of servlets and JSPs, how to create them, and how they work, refer to the following resources.

### Books

*Core Servlets and JavaServer Pages™* by Marty Hall; published by Sun Microsystems Press/Prentice Hall PTR, 2000.

*Professional JSP* by Karl Avedal, et al; published by Wrox, 2000.

*Web Development with JavaServer Pages* by Duane K. Fields and Mark A. Kolb; published by Manning Publications Co., 2000. The online edition is available for purchase at http://www.manning.com. Note that this book is used in the ENOVIA Live Collaboration Introduction/Studio Customization Toolkit Development class offered by ENOVIA and is a good introductory book.

## Online Resources

Sun’s Servlet home page: http://java.sun.com/products/servlet/.

Sun’s JSP home page: http://java.sun.com/products/jsp/.

Tutorial on creating servlets and JSPs: http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

Tutorial and guide for creating servlets http://www.novocode.com/doc/servlet-essentials/index.html.

Paul Flavin's Servlet Resources page http://www.frontiernet.net/~imaging/servlets_intro.html.

## Overview of Building Java Applets

Included with the ENOVIA Live Collaboration Studio Customization Toolkit is sample code that you can use to build your own applications. You will need to identify the Java class packages you want to use, link the specific classes into your own classes, and invoke the applet from an HTML page.

Throughout the section that describes the applet Studio Customization Toolkit, we use the terms *application* and *applet*. The following explains the difference between the two terms:

- Applications are not run within a Web browser. They do have access to the file system on the machine on which they are running. For example, you can use the Java Class packages to build Java applications, which are run just like any other Studio Modeling Platform component.
- Applets are run within a Web browser. Applets cannot do anything to the local machine on which they are running (reading, writing or deleting files, for example) unless the applet code is signed with an unforgeable digital ID. Further, a user must give permission for an applet to perform any tasks outside the browser. For additional information on applets, see *Information on Code Signing for Java Applets*.

Refer to *Creating a Custom Applet* for details.

### Information on Code Signing for Java Applets

For a review of the process of digitally signing a Java applet's files, see the home page (http://www.suitable.com/docs/signing.html) of Daniel T. Griscom (griscom@suitable.com).

# Studio Customization Toolkit Programming Notes

## Transactions cannot be Nested

If the `start() method` of the context class is called a second time before the commit or abort methods are called for the first transaction, the system throws a MatrixException. Before starting a transaction, use the isTransactionActive() method to determine if a transaction is still active. If the result is false, it’s safe to start another transaction. If true, you must abort or commit the active transaction before starting a new one. Refer to *Transaction Management* for more information.

## Using getPersons()

Storing data in a HttpSession or static member variables of a JSP requires some consideration and careful planning. For example, a page may display a list of users in a select field. The list of users is retrieved with the `matrix.db.Person.getPersons()` method every time the page is displayed. If the installation has a lot users, a large number of objects will be created that only exist for a short period of time and put a strain on the Java garbage collection. A better solution is to retrieve the list once, keep it in a static member variable, and perhaps refresh it every 24 hours.

## Using a Range Program

The same concepts for retrieving a list of users apply for user-specific data. The values of an attribute range may be determined by a range program and made dependent on the actual user. Recalculating these values whenever these values are displayed is costly. The better solution is to determine the values once and then store them in the HttpSession object.

While there are many cases where this caching technique may be beneficial for performance and scalability, one needs to be careful not to create a memory leak by mistake.

## Deadlock

A deadlock situation occurs when two tasks are trying to gain control of objects, and each has a lock on a resource that the other needs to process. For example, transaction A obtains object 1, updates it, then attempts to obtain object 2. Transaction B first obtains object 2, and then attempts to obtain object 1. The transactions are deadlocked because each is waiting to obtain an object the other has, but neither transaction will release the object it currently has. As a result, the transactions will be unable to continue processing. Eventually the transactions will timeout because Oracle does have deadlock detection.

Although it can be very difficult to anticipate deadlock, it is something a JSP developer should be aware of.

## Use of `open` and `close` Methods

Interfaces for stateless objects in Studio Customization Toolkit no longer require a call to open() or close(), thereby improving performance. Stateless objects include:

| | | |
|---|---|---|
| BusinessObject | BusinessType | Cue |
| Filter | Format | Group |
| Page | Person | Policy |
| Query | Relationship | RelationshipType |
| Report | Resource | Role |
| Store | Set | Table |
| Tip | ToolSet | User |
| Vault | | |

The context that is stored in the stateless object is NULL after a call to close(). Using any methods in a stateless class that don't take a context as a parameter require a call to open().

## Open With ID

As of Studio Customization Toolkit version 9.0, a method is provided on matrix.db.BusinessObject that allows the caller to open a business object by specifying a business object ID only. The routine returns type, name, revision, and vault.

## MQLCommand.executeCommand()

MQLCommand.executeCommand() returns a null pointer unless you do an `mqlcommand.open(_context)` and `mqlcommand.close(_context)` before and after.

## Bo.getAttributes()

Bo.getAttributes() returns an empty list unless you call this first:

```
BusinessObjectAttributes list2 = bo.getAttributes(_context);
```

So the new code would look like this:

```
BusinessObjectAttributes list2 = bo.getAttributes(_context); //
new call
AttributeList list = = list2.getAttributes();
AttributeItr itr = new AttributeItr(list);
```

## Setting nowait Transactions for Integrations

To ensure that integration transactions fail gracefully when a locked resource is encountered, use the `set transaction nowait` statement in the MQLCommand.executeCommand() after Context.start(). The `nowait` statement generates an error if a requested object is in use. For example:

```
Context ctx=new Context(":bos", "localhost");
    :
    :
    ctx.start();
    MQLCommand cmd=new MQLCommand();
    cmd.executeCommand(ctx, "set transaction nowait");
    :
```

For more information on MQLCommand.executeCommand(), see *MQLCommand.executeCommand()* in this chapter and *Sample 2: MQL Command Execution* in Appendix B.

# 4 Creating a Sample Server-side Application

## Sample Application

The examples and descriptions in this chapter are from a sample application called Quotation Viewer. Quotation Viewer lets users find quotations, view details about quotations, and checkin and out files. The sample application does not depend on particular data, it works with any type of object. For example, the find page finds any object type, not just quotation objects. This application does not use any of the configurable components of the Studio Customization Toolkit. For information on configurable components, refer to the *ENOVIA Live Collaboration Application Development Guide*.

### Conventions

Conventions used in this chapter are:

- All caps to represent substitutions

  In places where you should substitute information specific for your site, the guide uses all capital letters. For example, WEB_SERVER_DOCUMENT_ROOT, means you should substitute the path for your Web server document root.
- Bold, dark red indicates Java and Javascript

  To help distinguish between HTML and Java/JavaScript, the guide uses bold, dark red font for Java and Javascript. For example:

```
<td width="100%" align="center" ><form name="loginForm"
method="post" action="<%=LoginServlet.getURL()%>">
```

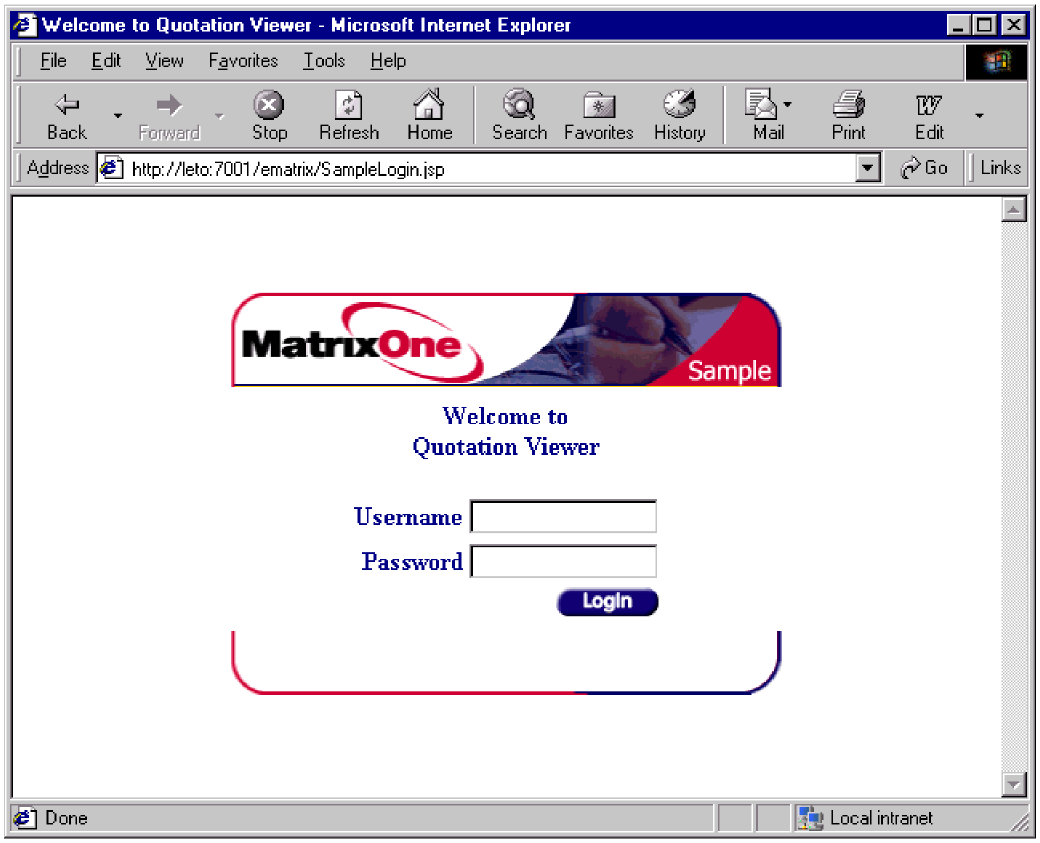
- Bold, dark green indicates highlighted text

  To point out specific pieces of code, the guide uses a dark green font. This code may be standard HTML or a mixture of HTML and Java. For example:

```
<tr>
    <td>&nbsp;&nbsp;&nbsp;
<a href=SampleFindDialog.jsp>Find quotations</a></td>
  </tr>
```

## Creating a Login Page

This section describes how to use the servlet Studio Customization Toolkit to create a simple login page for our sample application. The login page looks like this:


Welcome to Quotation Viewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/ematrix/SampleLogin.jsp
Go Links
MatrixOne
Sample
Welcome to
Quotation Viewer
Username
Password
Login
Done
Local intranet

This login page performs these functions:

- Authenticates the username and password that the user enters.
- If the username and password is authenticated, the page logs the user into the ENOVIA Live Collaboration database and forwards to the application home page.
- If the username and password is invalid, the page presents an error message and lets the user re-enter.

### Complete Sample Code

The sample code shown below is the entire content of SampleLogin.jsp. This JSP produces the login page shown and described in the previous section. The remaining portions of this chapter explain the commands in this code.

```
<%--
  SampleLogin.jsp -- page called to log user into the Matrix
database
--%>



<html>
```

```
<head>
<title>Welcome to Quotation Viewer</title>
</head>
<body text="#000088" bgcolor="#ffffff" >

<%@ page import = "com.matrixone.servlet.*, matrix.db.*" %>
<%

  // Determine the path for the jsps
String dirName = request.getRequestURI();
  dirName = dirName.substring(0,dirName.lastIndexOf("/")+1);

// set the target page
  if (Framework.getTargetPage(session) == null)
    Framework.setTargetPage(session, dirName +
"SampleHome.jsp");

  // check for error messages
String testMsg = null;
  MatrixServletException e = Framework.getError(request);
  if (e != null)
    testMsg = e.getMessage();
}
%>

<%
  if (e != null) {
%>
    <p align="center"><font face=arial size=4 color="red"><b>
Error: <%= errorMsg%> </b></font></p>
<%
  }
  else {
%>
    <p><font face=arial size=4>&nbsp;</font></p>
<%
  }
%>


</p><p>&nbsp;</p>
<div align="center"><center>

<table border="0" width="580" >
  <tr>
    <td bgcolor=white align=center><img
src="<%=dirName%>splashDemoLoginTop.gif"></td>
  </tr>
  <tr>
```

```
    <td width="100%" align="center" ><form name="loginForm"
method="post" action="<%=LoginServlet.getURL()%>">
      <div align="center"><center>
      <table BORDER="0" WIDTH="150"  align="center">
        <tr>
          <td width=100% colspan=2 align=center><b>Welcome to
<br>Quotation Viewer<p></p></b></td>
        </tr>
        <tr>
          <td  width=50% align=right><b>Username</b></td>
          <td width=50% ><input type="text"
name="<%=LoginServlet.FORM_LOGIN_NAME%>" value=""
          size="15" maxlength="36"></td>
        </tr>
        <tr>
          <td width=50% align=right><b>Password</b></td>
          <td width=50% ><input type="password"
name="<%=LoginServlet.FORM_LOGIN_PASSWORD%>" size="15"
maxlength="30"></td>
        </tr>
        <tr>
          <td width=50% align=right>&nbsp;</td>
          <td width=50% align=right><p><a
href="javascript:document.loginForm.submit();"><IMG
src=”<%=dirName%>buttonLogin.gif" border=0></a></td>
        </tr>
      </table>
      </center></div>
     </td>
     </tr>
     <tr>
    <td bgcolor=white align=center><img
src=splashDemoLoginBottom.gif></td>
  </tr>

    </form>

</table>
</center></div>
</body>
</html>
```

## Using the Context Object

The Login servlet logs in users by creating a matrix.db.Context object that is stored in the http session for the user. Before any Studio Customization Toolkit operation can take place, a context object must be established to connect to the ENOVIA Live Collaboration database. This context object contains username and password information, as well as server and host name.

The Login servlet first checks to see if the user has already been authenticated by the Web/application server (for example, through LDAP). If pre-authentication has occurred, the servlet uses its username and password. Otherwise, the servlet uses the entered username and password. The servlet gets other parameters required for creating a context object—server and host—from the web.xml file. Once a context object is stored in the session, the user is considered logged in.

## Context Object User Access

Context objects are never thread safe and should never be used by more than one thread at a time. As described above, the Login servlet creates the context object for you. You should never create a new context object unless you are writing a custom application that doesn’t use the servlet and need to create the context object for the first time.

Furthermore, when the context object is declared as static, it is not user safe. The danger is that one user could begin a transaction on a JSP that has a static context variable, and a second user could access the same JSP. At this point, the context gets switched from the first user to the second user. When the first user is ready to complete his activities, the context object is used to commit the transaction. However, the context is trying to commit a transaction it doesn't have because it no longer is attached to the first user. This results in a stranded transaction for the first user, which has no recovery.

Below is an example of a context defined as static:

```
static String eMatrixPropertiesName;
static Properties eMatrixProperties;
. . .
static Context context;
```

It is better to have the context not defined as static, as shown below:

```
static String eMatrixPropertiesName;
static Properties eMatrixProperties;
. . .
Context context;
```

Following are diagnostic features that provide automatic detection for shared context and dangling transaction problems:

- When a context is shared between multiple threads, an error message will be written to mxtrace.log if `MX_BOS_ALLOW_SHARED_CONTEXT` is set to `FALSE.` The message states “Session [SESSION ID] in use by multiple threads.” The thread will also stop processing. For more information about `MX_BOS_ALLOW_SHARED_CONTEXT`, see “Installing the Collaboration Server” chapter in the *ENOVIA Live Collaboration Installation Guide*.
- Each time a transaction is left dangling, a new database connection will be created to replace the one that is waiting for a commit. A message will be written to mxtrace.log that states “Creating new database connection. Pool size insufficient.” The database connection is created ad hoc and is not part of the connection pool configured by the `MX_CONNECTION_POOL_SIZE` setting.

The above diagnostics are informational and intended to aid developers in solving these types of problems.

## Context Object Thread Safety

Although you cannot create another context object, you can create a clone of the context object that is safe to use on the calling thread by using the Framework class method `getFrameContext()`.

For example, instead of using `Framework.getContext(session)`, use `getFrameContext(session)`, as shown below:

```
. . .
matrix.db.Context context = null;
// if reads are enabled then use the framecontext
context = Framework.getFrameContext(session);
if (context != null){
    setPageContext(context);
}
. . .
```

Create a new frame context object when you need a context that performs transaction management independently of the existing context. For example, you can create a frame context object for each frame on a page. Frame context objects are not cached as part of the session state, so they are available for garbage collection immediately after they're used.

Here is an example of how getFrameContext is used in the emxTopInclude.jsp, which is installed with the Application Exchange Framework.

```
<%
  //Commenting out code for catching last visited page before
Timeout

  String emxTopIncDirName = Framework.getPagePathURL("");

  if (!Framework.isLoggedIn(request)) {
    String emxTopIncLoginURL = "emxLogin.jsp";
    %>
    <jsp:forward page="<%=emxTopIncLoginURL%>" />
    <%
    return;
  }
  matrix.db.Context context =
Framework.getFrameContext(session);
  setPageContext(context);

%>
```

## Specifying the Login Page

You can specify the login page using the `ematrix.login.page` property in web.xml. If a user who is not logged in attempts to use the application, the application will present the login page. For more information on setting properties, see the *ENOVIA Live Collaboration Installation Guide*.

## Tasks for creating the Login Page

Creating the SampleLogin.jsp can be broken down into several tasks:

- Enter comments for the page. See *Entering Comments*.
- Write the page-level HTML code such as the header, title, and body HTML tags. See *Including Page-Level HTML Tags*.

- Write the page directive, which imports classes to be used in the page. See *Importing Java Classes*.
- Determine the path for the JSPs. See *Establishing the JSP Path*.
- Specify the page to forward to when the user submits a valid username and password. See *Setting the Page to Forward to Upon Login*.
- Display error messages, especially those related to login failures. See *Displaying Error Messages on the Page*.
- Authenticate the username and password. See *Authenticating the User*.

## Entering Comments

All the JSPs in the sample application begin with a hidden JSP comment at the top that names the JSP file and gives a brief description of the function of the JSP. Hidden comments are not sent to the client and therefore are not viewable in the displayed page or in the source on the client. Hide comments from the compiler by surrounding the text with <%-- and --%>.

```
<%--
  SampleLogin.jsp -- page called to log user into the Matrix
database
--%>
```

Two other kinds of comments are used throughout the sample application JSPs:

- The HTML comment tags, <!-- Comment text -->. Text surrounded by this tag is sent to the client and therefore is visible in the HTML source.
- The Java comment tags, // for single line and /* for multiline comments. These tags are for use within Java code. These comments are ignored by the compiler and therefore not visible in the HTML source.

## Including Page-Level HTML Tags

The sample JSPs that display to the browser contain these page-level HTML tags:

- The standard <html> tag, which marks the beginning and end of the file.
- The standard header HTML tag <head>, which contains information about the entire page, such as the page title.
- The title tag HTML tag <title>, which must be within the header tag.
  ```
  <title>Welcome to Quotation Viewer</title>
  ```
- The standard <body> HTML tags, which contains the content of the page, including Java content. You can define attributes of the page in the opening <body> tag, such as the text and background color.
  ```
  <body text="#000088" bgcolor="#ffffff" >
  ```

After adding the introductory hidden comments and the page-level HTML tags, the SampleLogin.jsp looks like this:

```
<%--
  SampleLogin.jsp -- page called to log user into the Matrix
database
--%>

<html>
```

```
<head>
<title>Welcome to Quotation Viewer</title>
</head>
<body text="#000088" bgcolor="#ffffff" >

</body>
</html>
```

## Importing Java Classes

The page directive defines attributes for the entire page. Page directives often import the packages that will be used in the page. For example, the SampleLogin.jsp imports all the matrixone.servlet and matrix.db packages.

```
<%@ page import = "com.matrixone.servlet.*, matrix.db.*" %>
```

Page directives also frequently contain an include page command that includes a static file on the page. To see an example of a JSP that uses an include directive, see *Including a Static File: Include Directive*.

## Establishing the JSP Path

The SampleLogin page gets the path for the JSPs so it can prepend the path when it references other pages and image files. Normally when a JSP references another file, the path is relative to the current JSP. If the path begins with "/", the path is relative to the application's document root directory and is resolved by the Web server. By prepending referenced files with an expression that contains the JSP path, we ensure that the page looks in the correct directory for the files.

The page determines the JSP path by getting the current URL up to and including the last "/"—this is the entire URL minus the specific JSP and any query strings.

```
// Determine the path for the jsps
String dirName = request.getRequestURI();
  dirName = dirName.substring(0,dirName.lastIndexOf("/")+1);
```

The page uses dirName in several expressions. The syntax for expressions in JSP is <%=. For example:

```
<td bgcolor=white align=center><img
src="<%=dirName%>splashDemoLoginTop.gif"></td>
```

## Setting the Page to Forward to Upon Login

When a user successfully logs in, our sample application should present the application home page. The login page sets the target page to go to after login using the `setTargetPage` method in the Framework servlet. The target page is defined by appending the name of the JSP, SampleHome.jsp in this case, to the JSP path, which is contained in dirName. The page first checks to make sure the target page isn't already set in the Session object using the getTargetPage method.

```
// set the target page
  if (Framework.getTargetPage(session) == null)
    Framework.setTargetPage(session, dirName +
"SampleHome.jsp");
```

To specify the page to forward to, use the `setTargetPage` method in the Framework servlet.

| **setTargetPage Method Parameters** | | |
|---|---|---|
| **Name** | **Value** | **Description** |
| Session | session | Session for this request. |
| Target | PAGE_TO_FORWARD_TO | In our sample application the page to forward to is the JSP_PATH plus SampleHome.jsp.<br>If not, the pathname is relative to the current JSP file. |

Another way to set the target page is to specify a target page in the file using the web.xml `ematrix.home.page` property. You can also specify a page to forward to if the login fails using the `ematrix.login.failure.page` property.

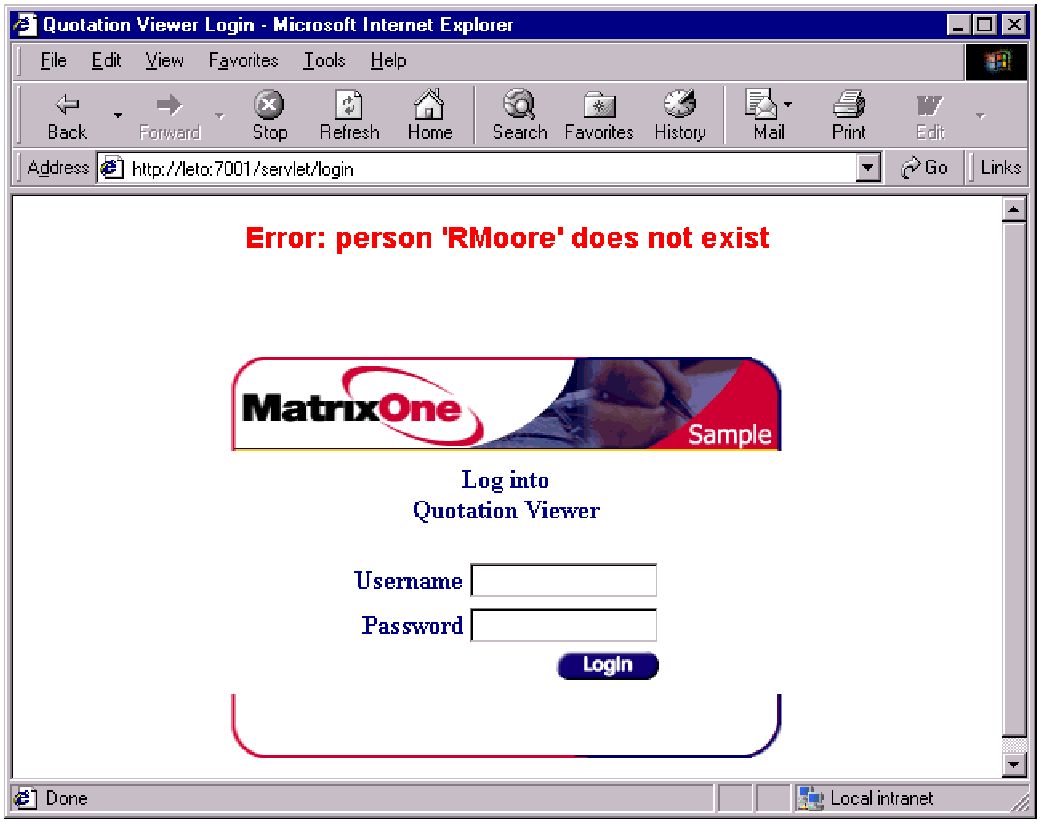
**Session object**

The session object is an implicit object within JSPs. It is used to share information for one user during each login session, no matter where the user goes within the application. A session object is a way of retaining state for a normally stateless HTTP Web site. By default, all JSP pages have access to the implicit session object. The session object exists until the user exits the browser, is inactive for a specific period of time (configurable in Web/application server), or logs out.

Every session objects has a unique identifier (session ID) associated with it. Every time a client access a resource on the server, the client provides the session ID that was assigned.

## Displaying Error Messages on the Page

The SampleLogin.jsp displays errors to the user. For example, if a user enters an invalid username, the page displays an error message:


Quotation Viewer Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/servlet/login
Go Links
MatrixOne Sample
Log into Quotation Viewer
Username
Password
LogIn
Done
Local intranet

Error: person 'RMoore' does not exist

A similar error appears if the user enters a valid username but invalid password:

**Error: Invalid password**

*The error text comes from the Studio Customization Toolkit and may differ slightly depending on the browser used. For example, in Internet Explorer, the error message for invalid usernames includes the username that was entered.*

The SampleLogin.jsp uses the getError method in the Framework servlet to get any errors in MatrixServletException.

```
// check for error messages
String errorMsg = null;
  MatrixServletException e = Framework.getError(request);
  if (e != null)
   errorMsg = e.getMessage();
}
```

The page then displays any errors using:

```
<%
  if (e != null) {
%>
    <p align="center"><font face=arial size=4 color="red"><b>
Error: <%= errorMsg%> </b></font></p>
<%
  }
  else {
%>
    <p><font face=arial size=4>&nbsp;</font></p>
<%
  }
%>
```

## Authenticating the User

The ENOVIA Live Collaboration LoginServlet provides user authentication by accepting the username and password entered by the user in an HTML form. The SampleLogin.jsp creates an HTML form called loginForm. The form's action contains the expression <%=LoginServlet.getURL()%>. The `getURL` method specifies the URL that calls the servlet. (The URL for the LoginServlet is usually something like "c:\HOSTNAME\servlet\login".)

```
<form name="loginForm" method="post"
action="<%=LoginServlet.getURL()%>">
```

The LoginServlet has fields for the username and for the password: FORM_LOGIN_NAME and FORM_LOGIN_PASSWORD. These fields are passed in as expressions for the input box names.

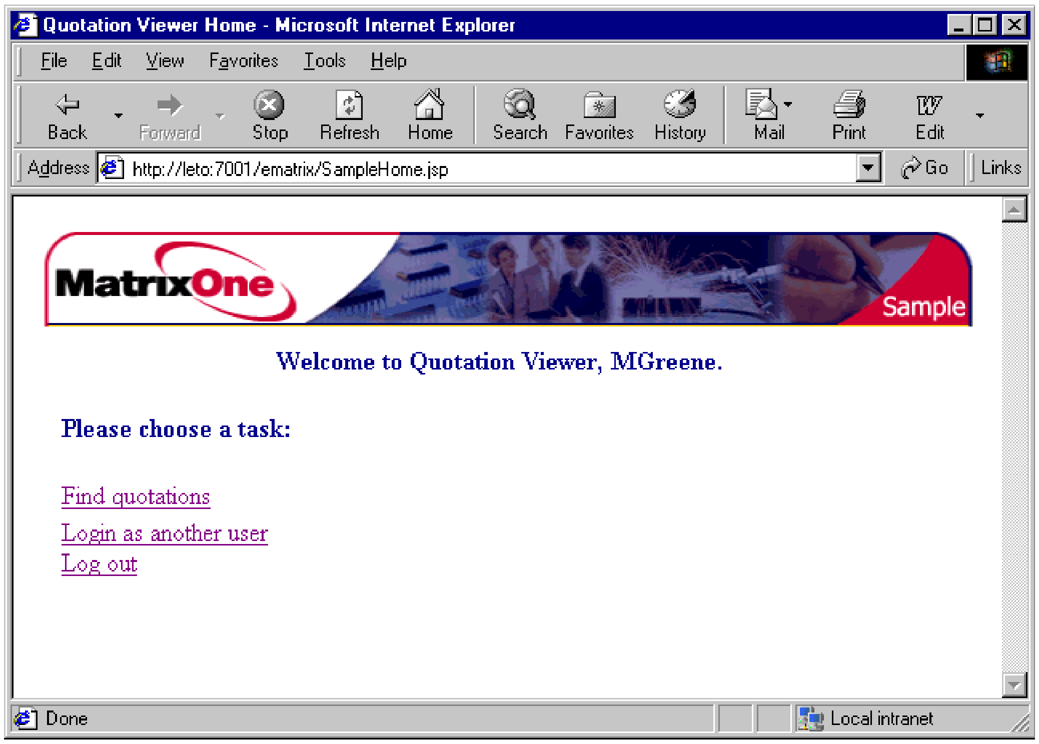
```
<tr>
   <td  width=50% align=right><b>Username</b></td>
   <td width=50% ><input type="text"
name="<%=LoginServlet.FORM_LOGIN_NAME%>" value=""
   size="15" maxlength="36"></td>
</tr>
<tr>
   <td width=50% align=right><b>Password</b></td>
   <td width=50% ><input type="password"
name="<%=LoginServlet.FORM_LOGIN_PASSWORD%>" size="15"
maxlength="30"></td>
</tr>
```

Finally, the page displays a Login button that uses the Javascript `document` object and `submit` method to send the data in the loginForm to the LoginServlet. The data is sent to the URL specified by the `getURL` method used in the action for the form.

```
<td width=50% align=right><p><a
href="javascript:document.loginForm.submit();"><IMG
src="<%=dirName%>buttonLogin.gif" border=0></a></td>
```

## Creating a Home Page

This section describes how to use the Studio Customization Toolkit to create a home page and an about page in the sample application. The home page looks like this:


Quotation Viewer Home - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/ematrix/SampleHome.jsp
Go Links
MatrixOne
Sample
Done
Local intranet

Welcome to Quotation Viewer, MGreene.

Please choose a task:

Find quotations
Login as another user
Log out

The home page performs these functions:

- Welcomes the user who is currently logged in.
- Provides links to the tasks that can be performed using the application.
- Performs certain "utility" tasks that should be performed on all displayed pages in the application:
  - Forwards to the login page if the user is not logged in.
  - Imports packages to be used throughout the application's JSPs: matrix.db, matrix.util, matrixone.servlet.
  - Specifies a JSP to send exceptions to. This error page displays information about the exception that has occurred.

### Complete Sample Code

The SampleHome.jsp includes a SampleUtility.jsp, which performs certain utility tasks needed for most pages in the application. In turn, the SampleUtility.jsp references a SampleError.jsp, which appears when an error occurs. The sample code for all three of these pages is shown below. The remaining portions of this section explain the code in these three pages.

## Code for the Home Page (SampleHome.jsp)

The sample code shown below is the entire content of SampleHome.jsp. This JSP produces the home page shown and described in the previous section.

```
<%-- SampleHome.jsp -- The home page for the application
--%>
<html>

<head>
<title>Quotation Viewer Home</title>
</head>
<body text="#000088" bgcolor="#ffffff" >

<%@include file = "SampleUtility.jsp"%>

<!-- content begins here -->
<center><table width=580 border=0>
  <tr>
    <td>
<p align=center><img src=splashHome.gif ><br><b>Welcome to
Quotation Viewer, <%=context.getUser()%>.<br></p><p></p>
<%
  }
%>
    </td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;<b>Please choose a task:<p></p></b></
td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;<a href=SampleFindDialog.jsp>Find
quotations</a></td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;<a href=SampleLogin.jsp>Login as
another user</a></td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;<a href=SampleLogout.jsp>Log out</a></
td>
  </tr>

</table>
</center>
<!-- content ends here -->
</body>
</html>
```

## Code for the Utility Page (SampleUtility.jsp)

```
<%--  SampleUtility.jsp -- imports standard packages and
forwards users who are not logged in to the login page
--%>

<%@ page import = "matrix.db.*, matrix.util.*
,com.matrixone.servlet.*" errorPage="SampleError.jsp"%>
<%
  String pageName = request.getRequestURI();
  if (request.getQueryString() != null) {
    pageName += "?" + request.getQueryString();
  }
  Framework.setTargetPage(session, pageName);
  if (!Framework.isLoggedIn(request)) {
    %>
    <jsp:forward page ="eServiceLogin.jsp"/>
    <%
    return;
  }
  matrix.db.Context context = Framework.getContext(session);
%>
```

## Code for the Error Page (SampleError.jsp)

```
<%--
   SampleError.jsp -- error page for application
--%>

<html>
<body>

<%@ page import = "matrix.db.*,matrix.util.*"
isErrorPage="true"%>

<br>
<%=exception.getMessage()%>
<br>

</body>
</html>
```

## Tasks for Creating the Home Page

Creating the home page for the Quotation Viewer can be broken down into these tasks:

*Tasks accomplished in SampleHome.jsp*

- Enter comments about the page. See *Entering Comments*.
- Write the page-level HTML code including the header, title, and body HTML tags. See *Including Page-Level HTML Tags*.

- Use an include directive to include a static file in the JSP (this is the SampleUtility.jsp). See *Including a Static File: Include Directive*.
- Find and display the name of the user who is currently logged in. See *Displaying the User's Name*.
- Provide links to the tasks that can be performed using the application. See *Providing Links to Other Pages in the Application*.

*Tasks accomplished in SampleUtility.jsp*

- Import packages to be used throughout the application's JSPs: matrix.db, matrix.util, matrixone.servlet. See *Importing Java Classes*.
- Forward to the login page if the user is not logged in. See *Setting the Page to Forward to Upon Login*.
- Get the context object for the page. See *Getting the Context for the Page*.

*Tasks accomplished in SampleError.jsp*

- Specify a page to send exceptions to (this task actually occurs in SampleUtility.jsp). This error page, SampleError.jsp, displays information about the exception that has occurred. See *Creating an Error Page*.

## Including a Static File: Include Directive

The include directive statically includes the content of a file in the page. For example, the SampleHome.jsp includes the SampleUtility.jsp:

```
<%@include file = "SampleUtility.jsp"%>
```

When the include directive is used, the JSP engine inserts the content of the file when the JSP page is translated into a servlet. This makes it an efficient way to include a page but can be more difficult to maintain than a dynamic include. Any changes made to the included file after the JSP is translated are not detected.

Another way to include pages is to use the `jsp:include` action, which inserts the file at the time the page is requested. For information on this command, see ***Including Page-Level HTML Tags***.

## Displaying the User's Name

The SampleHome.jsp contains an expression that uses the getUser method on the context object to display the user's name.

```
<p align=center><img src=splashHome.gif ><br><b>Welcome to
Quotation Viewer, <%=context.getUser()%>.<br></p>
```

## Providing Links to Other Pages in the Application

Like most home pages, our home page lets users choose the task they want to perform. The links are standard HTML anchors to other JSPs. The links are embedded in a table for display purposes so there are many table tags in the code.

```
<tr>
    <td>&nbsp;&nbsp;&nbsp;<b>Please choose a task:<p></p></b></
td>
  </tr>
  <tr>
```

```
    <td>&nbsp;&nbsp;&nbsp;
<a href=SampleFindDialog.jsp>Find quotations</a></td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;
<a href=SampleLogin.jsp>Login as another user</a>
   </td>
  </tr>
  <tr>
    <td>&nbsp;&nbsp;&nbsp;
<a href=SampleLogout.jsp>Log out</a></td>
  </tr>
```

## Creating an Error Page

The page directive of the SampleUtility.jsp contains the errorPage attribute, which specifies the SampleError.jsp. The errorPage attribute names the JSP that the current JSP sends exceptions to. If the pathname begins with "/", the path is relative to the application's document root directory and is resolved by the Web server. If not, the pathname is relative to the current JSP file.

```
<%@ page import = "matrix.db.*, matrix.util.*
,com.matrixone.servlet.*" errorPage="SampleError.jsp"%>
```

The page directive of the SampleError.jsp contains the isErrorPage attribute defined as True. Setting this attribute to True causes the JSP to display an error page. And since isErrorPage is set to true, we can use the implicit exception object to capture exceptions sent from SampleUtility.jsp. The getMessage method displays the exception. Below is the entire content of the SampleError.jsp, with the code relevant for displaying the error messages highlighted in dark green.

```
<%--
   SampleError.jsp -- error page for application
--%>

<html>
<body>

<%@ page import = "matrix.db.*,matrix.util.*"
isErrorPage="true"%>

<br>
<%=exception.getMessage()%>
<br>

</body>
</html>
```

## Forwarding Non-Logged In Users to Login Page

The SampleUtility.jsp, which is included in SampleHome.jsp and all other displayed pages in the application, checks to see if the user is logged in. If the user is not logged in, it forwards the user to the login page. This ensures that no one can access the application simply by typing in the correct URL for a page.

First, the utility page gets the entire URL of the current page, including any query strings following the JSP name. It sets this URL equal to pageName.

```
<%
  String pageName = request.getRequestURI();
  if (request.getQueryString() != null) {
    pageName += "?" + request.getQueryString();
  }
```

The page then sets pageName as the target page to come back to after the user logs in.

```
Framework.setTargetPage(session, pageName);
```

Then the page checks to see if the user is logged in using the `isLoggedIn` method. If the method returns false, the page forwards to the login page.

```
if (!Framework.isLoggedIn(request)) {
    %>
    <jsp:forward page ="SampleLogin.jsp"/>
    <%
    return;
  }
```

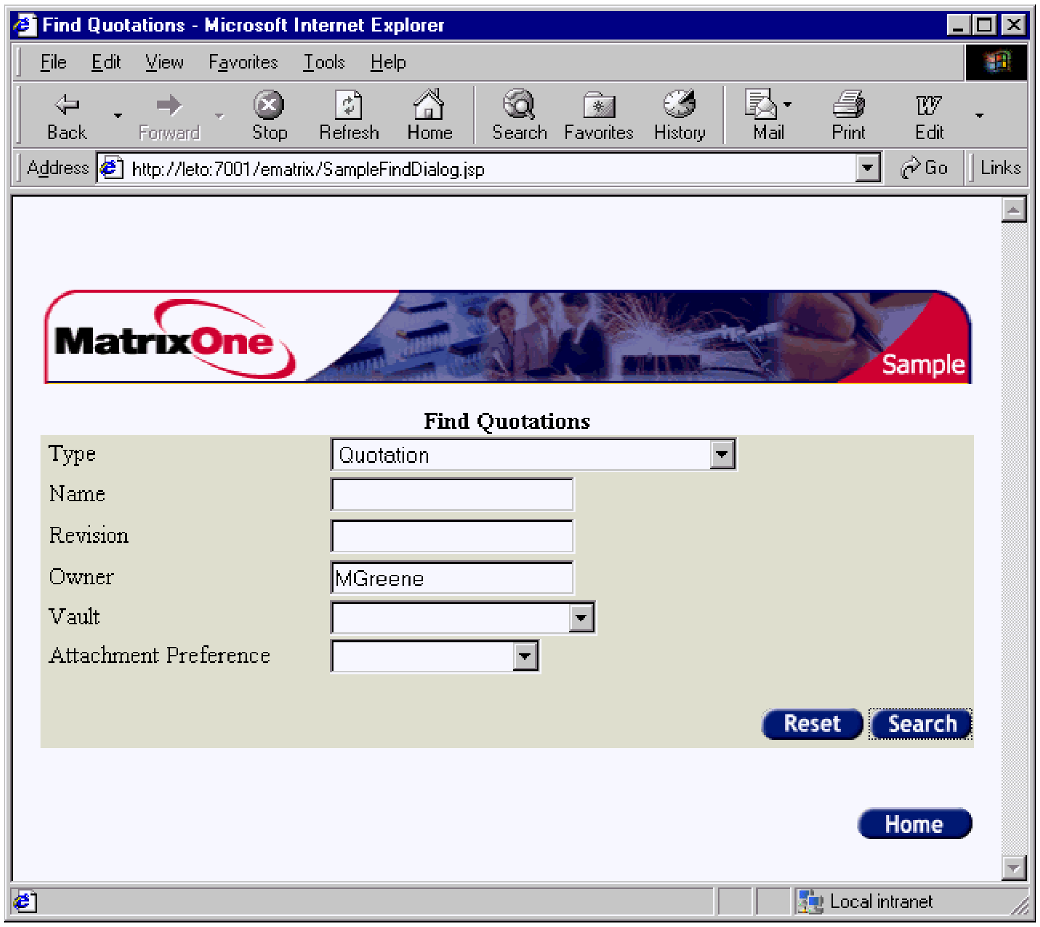
## Getting the Context for the Page

The final command in the SampleUtility.jsp gets the current context object from the session object, allowing the current page to use the context object as needed. (See *Using the Context Object*.)

```
matrix.db.Context context = Framework.getContext(session);
```

## Letting Users Find Objects

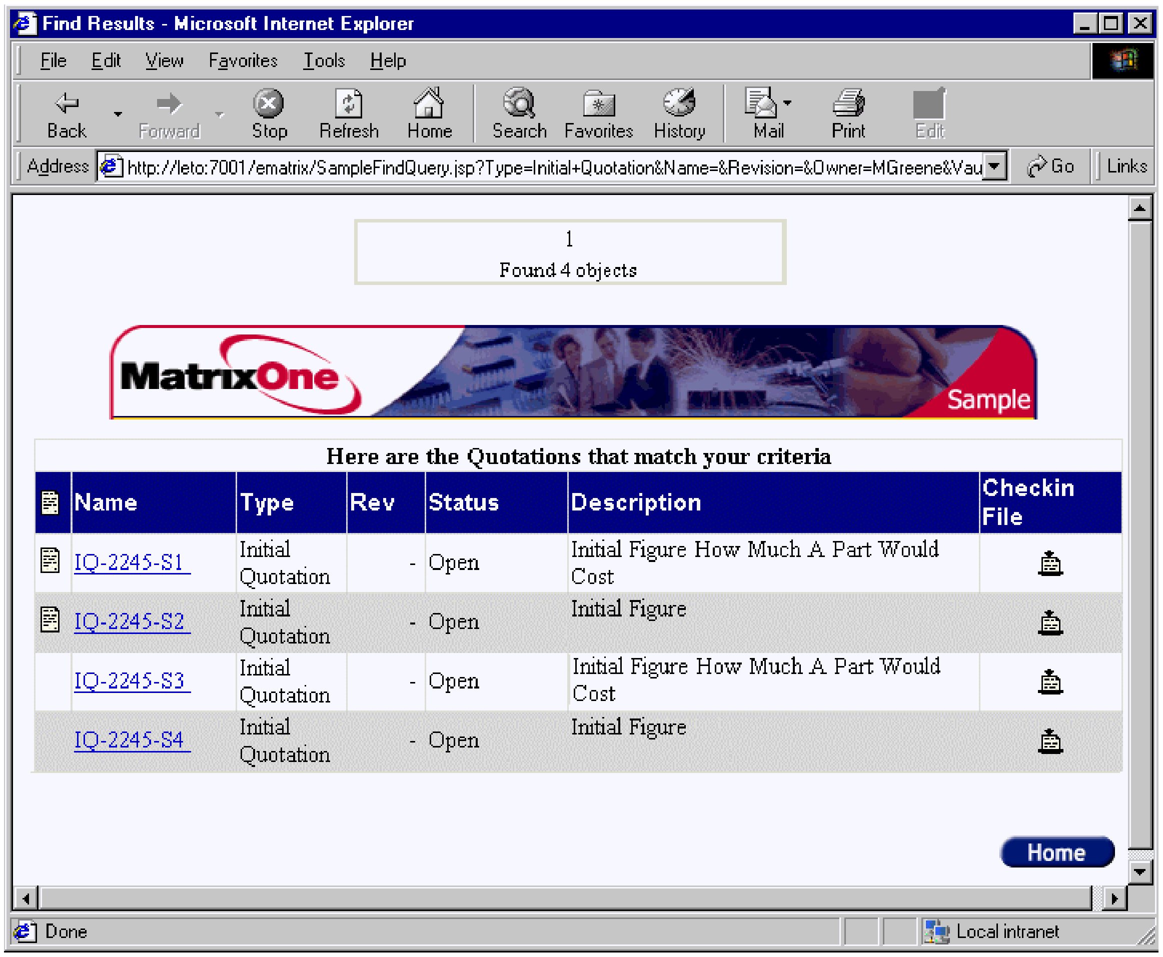
This section describes how to use the Studio Customization Toolkit to let users search for objects in the database. First, you need a page that users can enter search criteria on. The Find Quotation page in the Quotation Viewer application lets users search for quotations. This page can easily be modified to search for any type of object in the database.


Find Quotations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/ematrix/SampleFindDialog.jsp
Go Links
MatrixOne
Sample
Reset
Search
Home
Local intranet

Find Quotations

| | |
|---|---|
| Type | Quotation |
| Name | |
| Revision | |
| Owner | MGreene |
| Vault | |
| Attachment Preference | |

After entering search criteria and clicking Search, the Quotation Viewer application displays the search results:

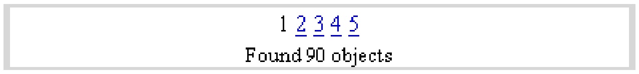

Find Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/ematrix/SampleFindQuery.jsp?Type=Initial+Quotation&Name=&Revision=&Owner=MGreene&Vau
Go Links
MatrixOne Sample
Home
Done
Local intranet

1
Found 4 objects

Here are the Quotations that match your criteria

| | Name | Type | Rev | Status | Description | Checkin File |
|---|---|---|---|---|---|---|
| | IQ-2245-S1 | Initial Quotation | - | Open | Initial Figure How Much A Part Would Cost | |
| | IQ-2245-S2 | Initial Quotation | - | Open | Initial Figure | |
| | IQ-2245-S3 | Initial Quotation | - | Open | Initial Figure How Much A Part Would Cost | |
| | IQ-2245-S4 | Initial Quotation | - | Open | Initial Figure | |

If more than 20 objects are found, the find feature displays only the first 20 on the page and provides links to access the remaining objects.



1 2 3 4 5
Found 90 objects

The find objects feature of Quotation Viewer performs these functions:

- Lets users type in search criteria: type, name, revision, owner, vault, and whether the object has files checked in.
- Searches for objects that meet the entered criteria.
- If matching objects are found, the application lists the objects. The results page lists information about each found object, such as the revision and status. The page also lets users view details about the objects and check in or check out files.
- If no matching objects are found, the application displays a message that says no objects were found and lets users enter new criteria.
- Lets users clear all criteria so they can enter new criteria.
- Lets users return to the home page at any point.

## Complete Sample Code

The find feature for the Quotation Viewer is created using four JSPs, not including the SampleUtility.jsp described in *Creating a Login Page*. The SampleUtility.jsp is included, statically, in all Quotation Viewer JSPs that display to the browser. The SampleFindDialog.jsp displays the page that lets users enter their search criteria. The SampleFindQuery.jsp actually performs the search. It does not display to the browser. The SampleFindResults.jsp displays found objects. Finally, SampleSetDisplay.jsp is included on the results page. It shows the number of objects found and gives links to other results pages when too many found objects to display on one page. The sample code for all four of these pages is shown below.

### Find Dialog Page (SampleFindDialog.jsp)

```
<%--  SampleFindDialog.jsp   -  This page displays the generic
find dialog. --%>
<html>

<head>
<title>Find Quotations</title>
</head>
<body>
<%@ include file = "SampleUtility.jsp" %>

 <link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >

<script language="javascript">
function clear() {
  document.findForm.reset();
}
</script>

<form name="findForm" method=get action="SampleFindQuery.jsp" >
  <table class=inner align=center width=581 cellspacing=0
border=0 bottommargin=2>
   <tr>
      <td colspan=3 align=center bgcolor=white><img
src=splashHome.gif></td>
    </tr>
   <tr>
      <td colspan=3 align=center bgcolor=white>
<%
  String errormessage =
(String)session.getValue("error.message");
  if ((errormessage != null) && (errormessage.length() > 1 )){
%>
<p class=error><b><%=errormessage%>
<%
    session.removeValue("error.message");
  }
%>
```

```
</td>
    </tr>
    <tr>
      <td colspan=3 bgcolor=white><b>Please enter search
criteria to find Quotations:</b><p></p></td>
    </tr>
    <tr>
      <td>&nbsp;Type
      </td>
      <td colspan=2> <select name="Type" >
           <option value="">
<%
          BusinessTypeList typeList =
BusinessType.getBusinessTypes(context, true);
            typeList.sort();
          BusinessTypeItr objectTypeItr = new
BusinessTypeItr(typeList);
          while(objectTypeItr.next()) {
%>
             <option value=
"<%=objectTypeItr.obj().getName()%>"><%=objectTypeItr.obj().get
Name()%>
<%
          }
%>
          </select>
      </td>
    </tr>

    <tr>
      <td>&nbsp;Name
      </td>
      <td colspan=2>
          <input type=Text name="Name" value="" >
      </td>
    </tr>

    <tr>
      <td>&nbsp;Revision
      </td>
      <td colspan=2>
          <input type=Text name="Revision" value="" >
      </td>
    </tr>

    <tr>
      <td>&nbsp;Owner
      </td>
      <td colspan=2>
          <input type=Text name="Owner" value="" >
      </td>
    </tr>

    <tr>
      <td>&nbsp;Vault
      </td>
      <td colspan=2> <select name="Vault" >
           <option value="">
<%
           VaultItr vaultItr = new
VaultItr(context.getVaults());
           while(vaultItr.next())
{
%>
               <option value=
"<%=vaultItr.obj().getName()%>"><%=vaultItr.obj().getName()%>
<%
               }

%>
           </select>
      </td>
    </tr>
   <tr>
      <td>&nbsp;Attachment Preference</td>
      <td> <select name="hasFiles">&nbsp;
              <option value= "">
              <option value= "Yes">One or More Files
              <option value= "No">Contains No Files
           </select>
      </td>
   </tr>
    <tr>
      <td>&nbsp;</td>
      <td>&nbsp;</td>
      <td>&nbsp;</td>
    </tr>
    <tr>
      <td>&nbsp;</td>
      <td>&nbsp;</td>
     <td align=right>
        <a href="Javascript:clear()"><img src=buttonReset.gif
border=0></a>
        <a href="javascript:document.findForm.submit()"><img
src="buttonSearch.gif" border=0></a>
      </td>
    </tr>
    <tr>
      <td bgColor=white>&nbsp;</td>
      <td bgColor=white>&nbsp;</td>
      <td bgColor=white align=right><p>&nbsp;</p><a
href=SampleHome.jsp><img src=buttonHome.gif border=0></a></td>
    </tr>
  </table>
</form>

</body>
</html>
```

## Search Execution (SampleFindQuery.jsp)

```
<%--  SampleFindQuery.jsp   -  This page Evaluates the Query
based on Criteria entered in the FindDialog
--%>

<%@include file = "SampleUtility.jsp"%>
<%
  try {
    String typePattern  = request.getParameter("Type");
    String namePattern  = request.getParameter("Name");
    String revision   = request.getParameter("Revision");
    String owner = request.getParameter("Owner");
    String vault = request.getParameter("Vault");

    Query query = new Query();


    // open the query or create it if needed
     try {
      query.create(context);
     } catch (MatrixException e) {
     }

    if (typePattern.length() < 1) {
      query.setBusinessObjectType("*");
    } else {
      query.setBusinessObjectType(typePattern);
    }

    if (namePattern.length() < 1) {
      query.setBusinessObjectName("*");
    } else {
      query.setBusinessObjectName(namePattern);
    }

    if (revision.length() < 1) {
      query.setBusinessObjectRevision("*");
    } else {
     query.setBusinessObjectRevision(revision);
    }
    if (owner.length() < 1) {
      query.setOwnerPattern("*");
    } else {
      query.setOwnerPattern(owner);
    }

    if (vault.length() < 1) {
      query.setVaultPattern("*");
    } else {
      query.setVaultPattern(vault);
    }

    // set the has files where clause
    if ( request.getParameter("hasFiles").equals("Yes")) {
      query.setWhereExpression("format.hasfile");
    } else if ( request.getParameter("hasFiles").equals("No")) {
      query.setWhereExpression("!format.hasfile");
    } else {
      query.setWhereExpression("");
    }

    BusinessObjectList boList = query.evaluate(context);
    query.close(context);

    // eval into set
     matrix.db.Set tempSet = new matrix.db.Set(boList);
    tempSet.setName(".emxTempSet");
    tempSet.open(context);
     tempSet.setBusinessObjects(context);

    // tempSet.open(context);
    if (boList.size() > 0) {
      // pass the set name to the results page
       tempSet.close(context);
      request.setAttribute("setName", tempSet.getName());
%>
      <jsp:forward page="SampleFindResults.jsp"/>
<%
    } else {
       tempSet.close(context);
      session.putValue("error.message", "No Match Found");
%>
      <jsp:forward page="SampleFindDialog.jsp"/>
<%
    }
  } catch (Exception e) {
    session.putValue("error.message", e.getMessage());
%>
```

```
    <jsp:forward page =
"<%=Framework.getCurrentPage(session)%>"/>
<%
    return;
  }
%>
```

## Results Page (SampleFindResults.jsp)

```
<%--  SampleFindResults.jsp   -  This page displays Search
Results
--%>

<html>

<head>
<title>Find Results</title>
</head>
<body>

<link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >
<%@include file = "SampleUtility.jsp"%>
<%@include file = "SampleSetDisplay.jsp"%>


<!-- content begins here -->

<table align="center"  width=680 cellspacing="1"
cellpadding="1" bottommargin=2>
  <tr>
    <td align=center colspan=7 bgcolor=white><img
src=splashHome.gif ></td>
  </tr>
</table>
<table align="center" class=inner width=680 cellspacing="1"
cellpadding="1" bottommargin=2>
  <tr>
    <td align=center colspan=7 bgcolor=white><b>Here are the
Quotations that match your criteria</b></td>
  </tr>
  <tr align=left>
    <th width=3%><img src=iconFile.gif  border=0></th>
    <th width=15%>Name</th>
     <th width=10%>Type</th>
    <th width=7%>Rev</th>
    <th width=13%>Status</th>
    <th width=37%>Description</th>
    <th width=13%>Checkin File</th>
  </tr>
<%
```

```
    BusinessObjectWithSelectItr busItr =  new
BusinessObjectWithSelectItr(returnSet);
    int i = 0;
    int j = i % 2;

     Hashtable typeHash = new Hashtable();

    while (busItr.next()) {
      String rowStr = "odd";
      if (j == 0) {
        rowStr = "even";
      }
      BusinessObjectWithSelect objectWithSelect = busItr.obj();
     String objectId = objectWithSelect.getSelectData("id");
      BusinessObject object = new BusinessObject(objectId);

      int readAccess = 0;
      // if the user does not have read access open will fail ,
so put it in try catch block
      try {
        object.open(context);
       // Get the basics of business object. Name, Revision and
Description
        String objType = object.getTypeName();
        String objName = object.getName();
        String objRevision  = object.getRevision();
        String objcurrentState =
objectWithSelect.getSelectData("current");
        String objDescription = object.getDescription();

          // check if this type contains methods or wizards
        if (!typeHash.containsKey(objType)) {
          BusinessType busObjType =
object.getBusinessType(context);
          busObjType.open(context);
          typeHash.put(objType, new
Boolean(busObjType.hasMethods()));
          busObjType.close(context);
        }
          Boolean tempBol = (Boolean)typeHash.get(objType);
          boolean hasMethods = true;
%>
        <tr class='<%=rowStr%>' >
          <td>
<%
          int checkoutAccess = 3;
         // give access to FileCheckout option only if user has
checkout access
          if (object.checkAccess(context,
(short)checkoutAccess)) {
```

```
            FormatItr formatItr = new
FormatItr(object.getFormats(context));
            while (formatItr.next()) {
              if (formatItr.obj().hasFiles()) {
%>
                <a class=one
HREF="SampleFileCheckout.jsp?busId=<%=objectId%>"><img
src=iconFile.gif  border=0 alt="File Checkout"></a>
<%
                break;
              } // end of if check hasFiles
            } // end of while loop for formats
          } else {
%>
            &nbsp;
<%
          }
%>
        </td>
        <td>
<%
       // allow user to see Details page only if has read access
          boolean hasRead = object.checkAccess(context,
(short)readAccess);
        if (hasRead) {
%>
        <a class=one
href="SampleObjectDetailsDialog.jsp?busId=<%=object.getObjectId
()%>" title="<%=object.getTip()%>"><%=objName%>&nbsp;</a>
<%
        } else {
%>
          <%=objName%>&nbsp;
<%
        }
%>
        </td>
        <td><%=objType%>&nbsp;</td>
        <td align=right><%=objRevision%>&nbsp;</td>
<%
        // give access to State Browser only if read access
        if (hasRead) {
%>
          <td><%=objcurrentState%>&nbsp;</td>
<%
        } else {
%>
          <td><%=objcurrentState%>&nbsp;</td>
<%
        }
%>
```

```
        <td><%=objDescription%>&nbsp;</td>
        <td align=center><a class=one
HREF='SampleFileCheckin.jsp?busId=<%=objectId%>'><img
src=iconCheckin.gif border=0 alt='File Checkin/out'></a></td>
      </tr>
<%
        object.close(context);
      } catch (MatrixException ex) {
%>
      </tr>
      <tr class='<%=rowStr%>'>
        <td width=3%>&nbsp;</td>
        <td
width=20%><%=objectWithSelect.getSelectData("name")%>&nbsp;</
td>
        <td
width=10%><%=objectWithSelect.getSelectData("type")%>&nbsp;</
td>
        <td
width=5%><%=objectWithSelect.getSelectData("revision")%>&nbsp;<
/td>
        <td width=10%>&nbsp;</td>
        <td width=37%>&nbsp;</td>
        <td width=13%>&nbsp;</td>
      </tr>
<%
      }
      ++i;
      j = i % 2;
    }
%>
</table>

<table width=680 style="width:680" align=center class=inner>
  <tr>
    <th width=100% colspan=2 ><img src=utilSpace3x3px.gif
height=3 width=3></th>
  </tr>

  <tr>
    <td align=left width=50%></td>
    <td width=50%></td>
  </tr>
  <tr>
    <td colspan=2 align=right>
      <input type="hidden" name="index" value=<%=currentIndex%>>
      <input type="hidden" name="span" value=<%=span%>>
      <input type="hidden" name="startIndex"
value=<%=startIndex%>>
      <input type="hidden" name="count" value=<%=count%>>
    </td>
  </tr>
  </table>
<table align="center" width=680 cellspacing="1" cellpadding="1"
>
   <tr>
      <td bgColor=white align=right ><p>&nbsp;</p><a
href=SampleHome.jsp><img src=buttonHome.gif border=0></a></td>
    </tr>

  </table>

</body>
</html>
```

## Display Number of Results Found (SampleSetDisplay.jsp)

```
<%--  SampleSetDisplay.jsp   -  load a subset from the specified
set
--%>

<%
  int maxPages = 10;
  int defaultSpan = 20;
  String defaultSet = ".emxTempSet";

  // read the set/collection name
  String setName = getParam(request, "setName");
  if (setName == null) {
    setName = defaultSet;
  }

  // read the index value
  int currentIndex = getIntParam(request, "index");
  int startIndex = getIntParam(request, "startIndex");

  // read the span value
  int span = getIntParam(request, "span");
  if (span == 0) {
    span = defaultSpan;
  }

  // read and sort the set/collection the first time only
  matrix.db.Set resultSet = null;
  int count = getIntParam(request, "count");
  if (count == 0) {
    matrix.db.Set tempSet = new matrix.db.Set(setName);
    tempSet.open(context);

    BusinessObjectList resultSetList =
tempSet.getBusinessObjects(context);
```

```
      tempSet.remove(context);
     tempSet.close(context);

     resultSetList.sort();

     // save the sorted set/collection entries
      resultSet = new matrix.db.Set(setName);
     resultSet.open(context);
     resultSet.appendList(resultSetList);
      resultSet.setBusinessObjects(context);

      count = resultSetList.size();
  }

  // open the set/collection each time in
  if (resultSet == null) {
    resultSet = new matrix.db.Set(setName);
    resultSet.open(context);
  }

  // define select clause and return the subset
  SelectList selectStmts = new SelectList();
  selectStmts.addType();
  selectStmts.addName();
  selectStmts.addRevision();
  selectStmts.addId();
  selectStmts.addCurrentState();

  BusinessObjectWithSelectList returnSet = new 
BusinessObjectWithSelectList();
  returnSet = resultSet.subsetSelect(context, currentIndex, 
span, selectStmts);
  resultSet.close(context);
%>

<div align=center>
<table width="40%" align=middle class=inner cellspacing=0 
border=0 cellpadding=0>
  <tr>
    <td width="1%" ><img src=utilSpace.gif height=1 width=1></
td>
    <td width="98%"><img src=utilSpace.gif height=2 width=1></
td>
    <td width="1%" ><img src=utilSpace.gif height=1 width=1></
td>
  </tr>
  <tr>
    <td width="1%" ><img src=utilSpace.gif height=22 width=1></
td>
    <td width="98%" bgcolor=white align=center valign=middle>
      <p class=darkblue>
```

```
<%
  int pageCount = 0;
  int displayPageCount;
  int index = startIndex;
  int mstartIndex;
  boolean isMore = false;

  int currentSpan = startIndex + (span * maxPages);
  if ( currentSpan < count )
    isMore = true;

  if (startIndex > 0 ) {
    mstartIndex = startIndex - (span * maxPages);
%>
    <a class=one
href="<%=request.getRequestURI()%>?index=<%=mstartIndex%>&span=
<%=span%>&startIndex=<%=mstartIndex%>&count=<%=count%>&setName=
<%=setName%>">prev</a>
<%
  }

  while (true) {
    pageCount++;

   // see if we need to output this page link
   if ((pageCount * span) > count) {
     int remander = count - ((pageCount - 1) * span);
     if (remander <= 0)
       break;
     }

   if (pageCount > maxPages)
     break;

   index = startIndex + ((pageCount - 1) * span);
    displayPageCount = ( startIndex / maxPages ) + pageCount;

   if (index == currentIndex) {
%>
      <%=displayPageCount%>
<%
    } else {
%>
      <a class=one
href="<%=request.getRequestURI()%>?index=<%=index%>&span=<%=spa
n%>&startIndex=<%=startIndex%>&count=<%=count%>&setName=<%=setN
ame%>"><%=displayPageCount%></a>
<%
    }
  }
  if (isMore) {
    mstartIndex = startIndex + (span * maxPages);
%>
    <a class=one
href="<%=request.getRequestURI()%>?index=<%=mstartIndex%>&span=
<%=span%>&startIndex=<%=mstartIndex%>&count=<%=count%>&setName=
<%=setName%>">more</a>
<%
  }
%>
      </p>
    </td>
    <td width="1%" ><img src=utilSpace.gif height=1 width=1></
td>
  </tr>
  <tr>
    <td width="1%"><img src=utilSpace.gif height=1 width=1></td>
    <td width="98%"  bgcolor=white align=center
valign=middle><font size=-1>Found <%=count%> objects</font></
td>
    <td width="1%"><img src=utilSpace.gif height=1 width=1></td>
  </tr>
  <tr>
    <td width="1%"><img src=utilSpace.gif height=1 width=1></td>
    <td width="98%"><img src=utilSpace.gif height=1 width=1></
td>
    <td width="1%"><img src=utilSpace.gif height=2 width=1></td>
  </tr>
 </table>
</div>

<%!
  //
  // check for URL appened values if not found, then look in
Request Headers and return the string
  //
  static public int getIntParam(HttpServletRequest request,
String parameter)
  {
    int returnInt = 0;

    String parameterValue = request.getParameter(parameter);
    if (parameterValue == null) {
      Integer temp = (Integer)request.getAttribute(parameter);
     if (temp != null) {
       returnInt = temp.intValue();
     }
   } else {
     returnInt = Integer.parseInt(parameterValue);
   }
   return returnInt;
```

```
    }
  %>
```

## Tasks for Creating the Find Object Feature

Some of the tasks involved in creating the find feature are the same tasks discussed in previous chapters, so we won't describe them again here. These tasks include:

- Enter comments about the page. See *Entering Comments*.
- Write the page-level HTML code including the header, title, and body HTML tags. See *Including Page-Level HTML Tags*.
- Use an include directive to include a static file in the JSP (this is the SampleUtility.jsp). See *Including Page-Level HTML Tags*. Also see other sections of Chapter 4 for help creating the SampleUtility.jsp.

Additionally, most the pages in the application reference a cascading style sheet called SampleStyleSheet.css so the look and feel of the pages is consistent. The style sheet is referenced using:

```
<link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >
```

The remaining tasks for creating the find feature can be broken down into these tasks:

*Tasks accomplished in SampleFindDialog.jsp*

- Create a form that lets users enter search criteria. The form submits the criteria to SampleFindQuery.jsp. See *Creating a Query Form*.
- Let users clear the contents of all the criteria boxes. See *Clearing the Criteria Boxes*.
- Display a message when no objects match the criteria. See *Displaying No Results Message*.
- Let users return to the home page at any point.

*Tasks accomplished in SampleFindQuery.jsp*

- Searches for objects that meet the entered criteria. See *Performing the Query*.

*Tasks accomplished in SampleFindResults.jsp*

- Lists information about each found object, such as the revision and status.
- Provide links that let users view details about the objects and check in or check out files.

*Tasks accomplished in SampleSetDisplay.jsp*

- Show only a subset of all found objects on a page. *Displaying a Subset of All Objects Found*.
- Create links to let users see the found objects that can't fit on the first results page (only 20 objects fit on a page).
- Display the total number of objects found.

### Creating a Query Form

The SampleFindDialog.jsp creates a form that lets users enter query criteria, including type, name, revision, owner, vault, and attachment preference. When the user clicks the Submit button, the pages sends the content of the form to the SampleFindQuery.jsp. Here is the code that defines the form and the Submit button.

```
<form name="findForm" method=get action="SampleFindQuery.jsp" >
```

```
-
-
-
<a href="javascript:document.findForm.submit()"><img
src="buttonSearch.gif" border=0></a>
```

To let the user search for an object type, the form displays all types in the database in a list using the `getBusinessTypes` method.

```
<%
          BusinessTypeList typeList =
BusinessType.getBusinessTypes(context, true);
            typeList.sort();
          BusinessTypeItr objectTypeItr = new
BusinessTypeItr(typeList);
          while(objectTypeItr.next()) {
%>
             <option value=
"<%=objectTypeItr.obj().getName()%>"><%=objectTypeItr.obj().get
Name()%>
<%
          }
%>
```

To let the user search for an object by name, revision, or owner, the form displays a standard text box. For example, here is the code for displaying the text box for the object name.

```
<tr>
      <td>&nbsp;Name
      </td>
      <td colspan=2>
          <input type=Text name="Name" value="" >
      </td>
    </tr>
```

Here is the code for creating a list of the vaults in the database to let users search by vault.

```
<tr>
      <td>&nbsp;Vault
      </td>
      <td colspan=2> <select name="Vault" >
           <option value="">
<%
           VaultItr vaultItr = new
VaultItr(context.getVaults());
           while(vaultItr.next())
{
%>
               <option value=
"<%=vaultItr.obj().getName()%>"><%=vaultItr.obj().getName()%>
<%
               }

%>
           </select>
```

```
            </td>
        </tr>
```

And to search based on whether the object has files attached or not:

```
        <tr>
            <td>&nbsp;Attachment Preference</td>
            <td> <select name="hasFiles">&nbsp;
                        <option value= "">
                        <option value= "Yes">One or More Files
                        <option value= "No">Contains No Files
                    </select>
            </td>
        </tr>
```

## Clearing the Criteria Boxes

Users can click the Reset button to clear all criteria entered in the criteria boxes. To create this feature, the SampleFindDialog.jsp creates a clear function that resets the form.

```
function clear() {
  document.findForm.reset();
}
```

The page then calls the function when the user clicks Reset:

```
<a href="Javascript:clear()"><img src=buttonReset.gif
border=0></a>
```

## Displaying No Results Message

If the query finds no results, the application returns the user to the SampleFindDialog.jsp so the user can enter new criteria. The page displays the error message stored in the session object, which is placed there by the SampleFindQuery.jsp. The removeValue method removes the message when the user enters new criteria.

```
<%
  String errormessage =
(String)session.getValue("error.message");
  if ((errormessage != null) && (errormessage.length() > 1 )){
%>
<p class=error><b><%=errormessage%>
<%
    session.removeValue("error.message");
  }
%>
```

## Performing the Query

To perform a query against the ENOVIA Live Collaboration database, you must perform these steps:

1. Create a new Query object using the Query class.
2. Queries can be named (to use an existing query) or unnamed (uses .finder).
3. Open or create the query.

4. Set the query options for search.
5. Evaluate the query.
6. Process the BusinessObjectList.
7. Close the Query.

The SampleFindQuery.jsp performs these steps by first getting the values the user entered in the form on SampleFindDialog.jsp.

```
<%
  try {
    String typePattern  = request.getParameter("Type");
    String namePattern  = request.getParameter("Name");
    String revision   = request.getParameter("Revision");
    String owner = request.getParameter("Owner");
    String vault = request.getParameter("Vault");
```

Next, the page creates the Query object.

```
    Query query = new Query();


    // create query
     try {
      query.create(context);
     } catch (MatrixException e) {
     }
```

Then the page enters the criteria values entered by the user. If no value was chosen or entered, the wildcard * is used.

```
    if (typePattern.length() < 1) {
      query.setBusinessObjectType("*");
    } else {
      query.setBusinessObjectType(typePattern);
    }

    if (namePattern.length() < 1) {
      query.setBusinessObjectName("*");
    } else {
      query.setBusinessObjectName(namePattern);
    }

    if (revision.length() < 1) {
      query.setBusinessObjectRevision("*");
    } else {
     query.setBusinessObjectRevision(revision);
    }

    if (owner.length() < 1) {
      query.setOwnerPattern("*");
    } else {
      query.setOwnerPattern(owner);
    }
```

```
        if (vault.length() < 1) {
          query.setVaultPattern("*");
        } else {
          query.setVaultPattern(vault);
        }

        // set the has files where clause
        if ( request.getParameter("hasFiles").equals("Yes")) {
          query.setWhereExpression("format.hasfile");
        } else if ( request.getParameter("hasFiles").equals("No")) {
          query.setWhereExpression("!format.hasfile");
        } else {
          query.setWhereExpression("");
        }
```

Next, the page evaluates and closes the query.

```
        BusinessObjectList boList = query.evaluate(context);
        query.close(context);
```

The returned objects are placed into a temporary set. If the set contains objects (at least one object matches the criteria), the page forwards to the results page, SampleFindResults.jsp, and the set name is passed to the results page. If the set contains no objects, the page stores the message “No Match Found” in the session object and returns to the SampleFindDialog.jsp, which then displays the message.

```
        // eval into set
         matrix.db.Set tempSet = new matrix.db.Set(boList);
        tempSet.setName(".emxTempSet");
        tempSet.open(context);
         tempSet.setBusinessObjects(context);

        // tempSet.open(context);
        if (boList.size() > 0) {
          // pass the set name to the results page
           tempSet.close(context);
          request.setAttribute("setName", tempSet.getName());
    %>
          <jsp:forward page="SampleFindResults.jsp"/>
    <%
        } else {
           tempSet.close(context);
          session.putValue("error.message", "No Match Found");
    %>
          <jsp:forward page="SampleFindDialog.jsp"/>
    <%
        }
      } catch (Exception e) {
        session.putValue("error.message", e.getMessage());
    %>
        <jsp:forward page =
    "<%=Framework.getCurrentPage(session)%>"/>
```

```
<%
    return;
  }
%>
```

## Displaying a Subset of All Objects Found

The SampleSetDisplay.jsp is responsible for displaying only portion of the found objects on a single page and providing links that let users access the remaining objects. To select only a subset of the entire set of found objects, the page uses the `subsetSelect` method, which is in the Set class in the db package. The subsetSelect method takes four parameters:

- context—the context for this request
- index—where to start in the set.
- count—how many objects to get.
- selectStatements—the StringList of select statements to be applied.

Here is the code in the page that defines the select clause and returns the subset. To see how the index and count parameter values for the subsetSelect method are defined, see the beginning of the JSP (*Display Number of Results Found (SampleSetDisplay.jsp)*).

```
SelectList selectStmts = new SelectList();
  selectStmts.addType();
  selectStmts.addName();
  selectStmts.addRevision();
  selectStmts.addId();
  selectStmts.addCurrentState();

  BusinessObjectWithSelectList returnSet = new
BusinessObjectWithSelectList();
  returnSet = resultSet.subsetSelect(context, currentIndex,
span, selectStmts);
  resultSet.close(context);
```

## Programming Notes for Queries

### Default .Finder Query Read by Default

By default, all users have the query named .finder, which contains the search criteria of the last query performed by that user. Thus, when a program calls getQueries(Context), at least this one query is returned.

A query constructed using the Studio Customization Toolkit constructor new Query() always references the default ".finder" query. Having constructed such a query:

- the open method reads the definition of .finder from the database and must be overridden by explicitly setting each field.

  When the open method of the query class is executed, the client executes the method on the corresponding server object, which opens the existing query in the database. The query fields (name pattern, type pattern, where pattern, etc.) are then communicated back to the client, and the client must either accept what is stored in the database or override it. This is similar to the mql modify query command, where only fields that are specified are changed on the server. Therefore, in order to assure expected results, Studio Customization Toolkit users must set query fields explicitly every time they open a query.

- the update method will write the current fields in the Java Query object back to the .finder definition in the database.
- the close method will release the Query object without updating the .finder definition in the database (nor clearing it).

It's possible to create a very temporary query by passing the emptystring as a name to the Query constructor:

```
Query q = new Query("")'
```

In this case:

- the open method initializes all fields to default values.
- the update method will error.
- the close method will release the Query object.

Creating unnamed queries in this manner is recommended because it creates a completely in-memory query that cannot conflict with any previously-existing queries in the database.

### Query Errors Reported in ClientTaskList, not MatrixException

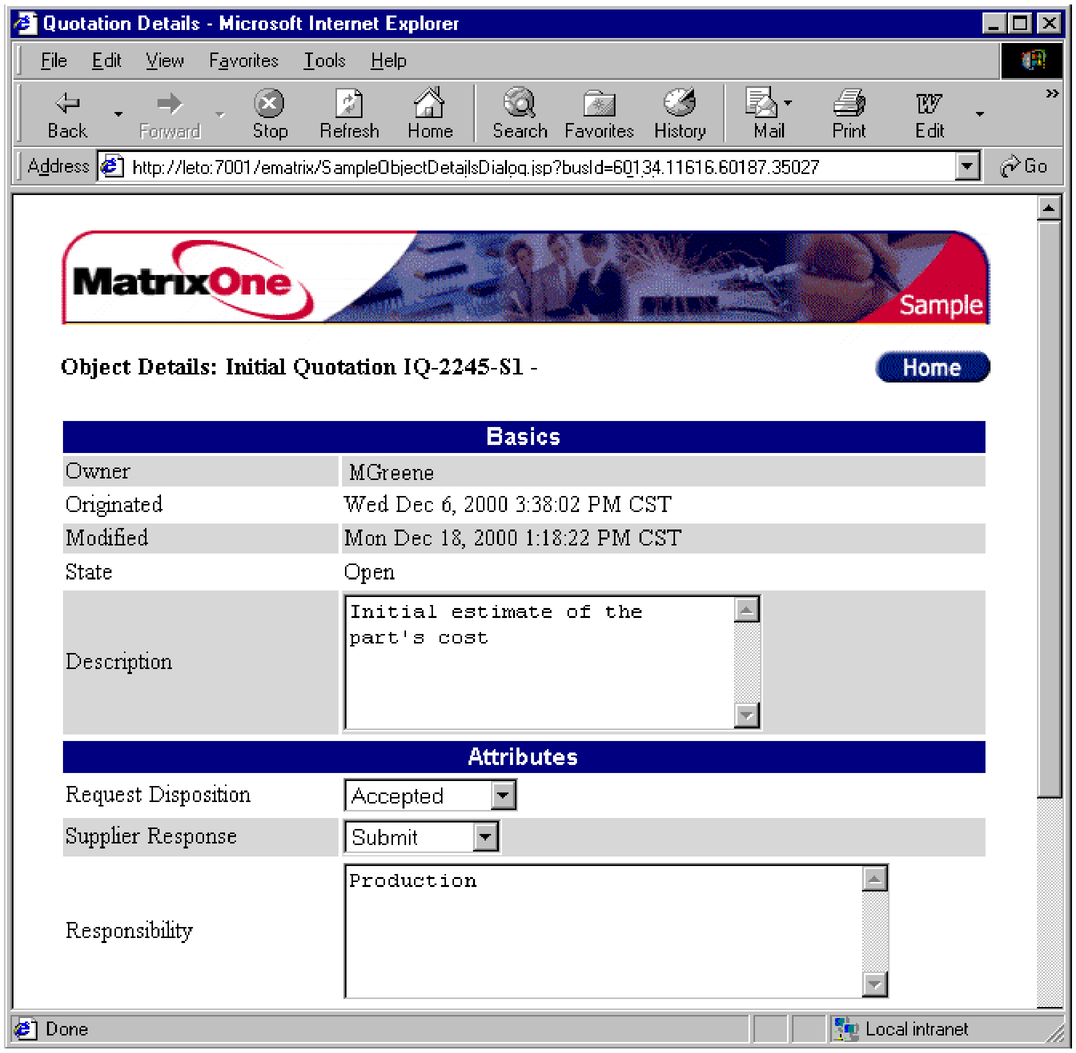
Query errors are not be reported via the standard MatrixException. This allows some data to be returned even if a subset of the interfaces are unavailable. You must check the Context object's ClientTaskList for error messages after the query and throw a MatrixException of your own if needed.

Here is sample code that shows how to check the ClientTaskList after evaluating the query and throw a MatrixException if errors are found. The vault Scott is an adapted vault on a Windows machine with the OracleTnsListener80 service stopped:

```
try
{
// create a new query
q.setVaultPattern(^Scott^);
q.setBusinessObjectType(^*^);
q.setBusinessObjectName(^*^);
q.setBusinessObjectRevision(^*^);
q.setOwnerPattern(^*^);
q.setWhereExpression(^^);
q.setObjectLimit((short)256);
q.setExpandType(true);
q.create(context);

// Evaluate the query (using the default visuals)
list = q.evaluate(context,context.getDefaultVisuals());
q.close(context);

System.out.println(^ Found bus=^ + list.size());

// Check client task list to see if there were any errors.
ClientTaskList tasks = context.getClientTasks();
System.out.println(^tasks.size() = ^ + tasks.size());
String errors = ^^;
for (int i = (tasks.size() - 1); i >= 0; i--) {
ClientTask task = (ClientTask) tasks.elementAt(i);
if (task.getReason() == ClientTask.reasonErrorMsg) {
tasks.removeElementAt(i);
if (errors != ^^)
errors += ^
^;
errors += task.getTaskData();
}
}
if (errors != ^^)
throw new MatrixException(errors);
} catch(MatrixException me) {
System.out.println(^an error occurred during query^);
me.printStackTrace();
}
```

## Displaying Object Details

This section describes how to use the Studio Customization Toolkit to create a Web page that displays the basics and attributes for an object. The following graphic shows the Quotation Details page for the Quotation Viewer application. This is the page that displays when the user clicks the object name from the Find Results page or from the Checkin/ Checkout pages.


Quotation Details - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address
Go
MatrixOne
Sample
Home
Done
Local intranet

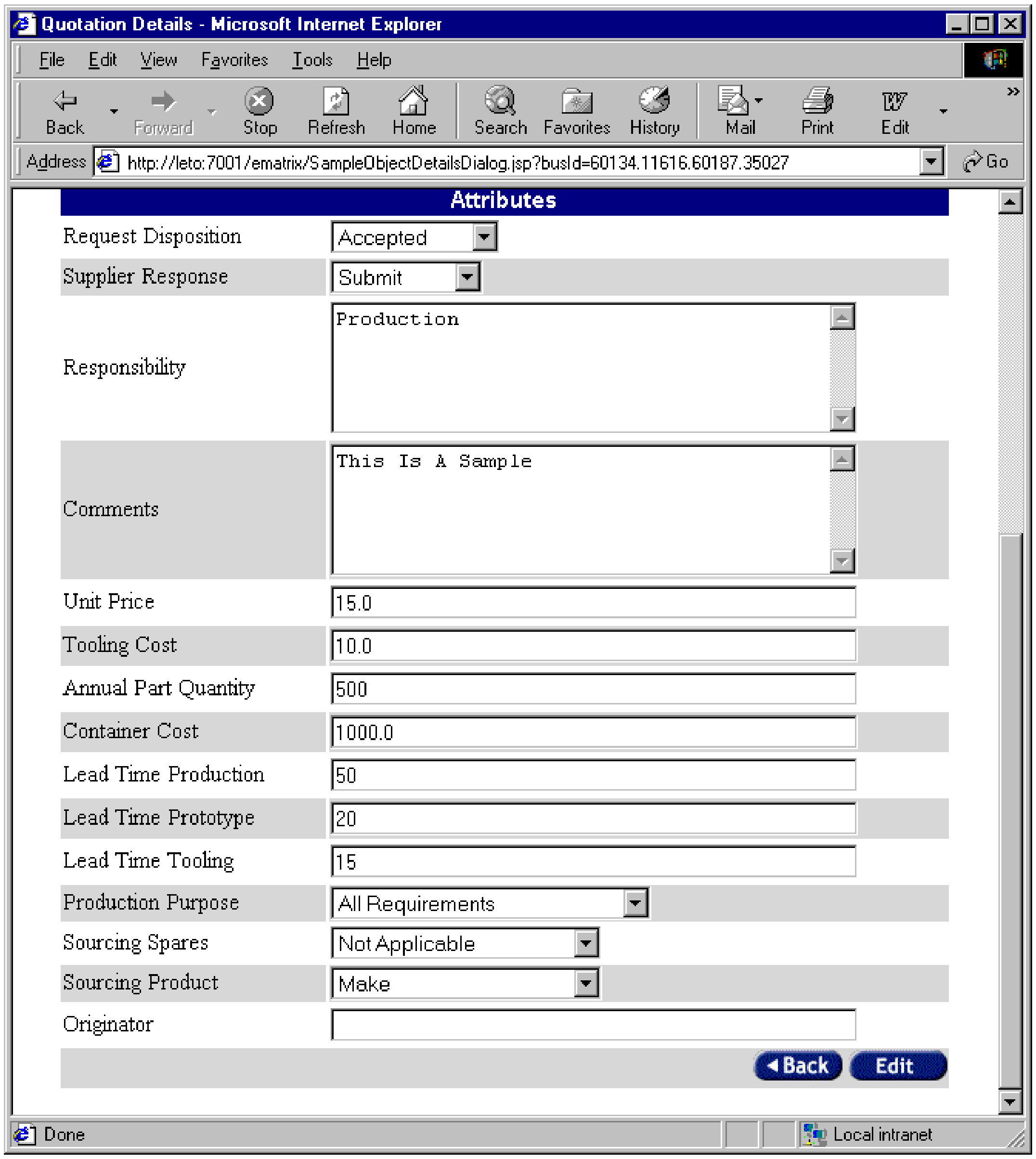
http://leto:7001/ematrix/SampleObjectDetailsDialog.jsp?busId=60134.11616.60187.35027

**Object Details: Initial Quotation IQ-2245-S1 -**

| Basics | |
|---|---|
| Owner | MGreene |
| Originated | Wed Dec 6, 2000 3:38:02 PM CST |
| Modified | Mon Dec 18, 2000 1:18:22 PM CST |
| State | Open |
| Description | Initial estimate of the part's cost |

| Attributes | |
|---|---|
| Request Disposition | Accepted |
| Supplier Response | Submit |
| Responsibility | Production |

Here is the bottom of the Quotation Details page.


Quotation Details - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://leto:7001/ematrix/SampleObjectDetailsDialog.jsp?busId=60134.11616.60187.35027
Go
Back
Edit
Done
Local intranet

**Attributes**

| | |
|---|---|
| Request Disposition | Accepted |
| Supplier Response | Submit |
| Responsibility | Production |
| Comments | This Is A Sample |
| Unit Price | 15.0 |
| Tooling Cost | 10.0 |
| Annual Part Quantity | 500 |
| Container Cost | 1000.0 |
| Lead Time Production | 50 |
| Lead Time Prototype | 20 |
| Lead Time Tooling | 15 |
| Production Purpose | All Requirements |
| Sourcing Spares | Not Applicable |
| Sourcing Product | Make |
| Originator | |

Note that the title of the page is Quotation Details but the page will display the details of any object found in the database.

The object details page performs these functions:

- Shows the selected object’s type, name, and revision (if any).
- Shows some basic information for the object, including the owner, originated and modified dates, and description.
- Shows the object’s current state.
- Shows all the attributes for the object.
- If the user has modify access for the object, lets the user update all attributes and the description.
- Lets the user return to the application home page or go back to the previous page.

## Complete Sample Code

The object details feature of the Quotation Viewer uses two JSPs (not including the SampleUtility.jsp described in *Code for the Utility Page (SampleUtility.jsp)*): SampleObjectDetailsDialog.jsp and SampleObjectDetails.jsp. The first JSP creates the user interface for displaying details about the selected object and lets the user enter updated values. The second JSP commits any updates to the database. The sample code for both pages is shown below. The remaining portions of this chapter explain the code in these two pages.

### Code for the Quotation Details Page (SampleObjectDetailsDialog.jsp)

The sample code shown below is the entire content of SampleObjectDetailsDialog.jsp. This JSP produces the Quotation Details page shown and described in the previous section.

```
<%--  SampleObjectDetailsDialog.jsp   -  This Page Displays the
Basics and Attributes of a Business Object
--%>

<html>

<head>
<title>Quotation Details</title>
</head>
<body>

<%@include file = "SampleUtility.jsp"%>
<link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >

<%!
  //
  // return the current state for this object
  //
  static public State getCurrentState(HttpSession session,
                                    BusinessObject busObj)
throws MatrixException
  {
    matrix.db.Context context = Framework.getContext(session);

    // get the current state for this object
    StateItr stateItr = null;
    try {
      stateItr = new StateItr(busObj.getStates(context));
    }
    catch (MatrixException e)
    {
      busObj.open(context);
      stateItr = new StateItr(busObj.getStates(context));
      busObj.close(context);
    }
    finally {
      while(stateItr.next()){
        if (stateItr.obj().isCurrent()) {
          break;
        }
      }
      return stateItr.obj();
    }
  }
%>

<!-- content begins here -->
<!--Java Script for client side validations-->
<script language=javascript>

function isNumeric(textBox){
  var n = textBox.value;
  if(isNaN(n) == true) {
    alert("Please enter a numeric value");
    textBox.focus();
  }
}

function editAttributes() {
  document.editAttributesForm.action =
"SampleObjectDetails.jsp";
  document.editAttributesForm.submit();
}
</script>

<%
  String returnPage= request.getParameter("returnPage");
  if (returnPage == null) {
    returnPage = request.getHeader("REFERER");
  }

  // open the current BusinessObject
  String busId = request.getParameter("busId");
  BusinessObject busObj = new BusinessObject(busId);
  busObj.open(context);

  //get the BusinessObject TNR
  String busType = busObj.getTypeName();
  String busName = busObj.getName();
  String busRevision = busObj.getRevision();

  //get the BusinessObject basics
  BusinessObjectBasics busBasics = busObj.getBasics(context);
  String busOwner = busBasics.getOwner();
  String busOriginated = busBasics.getCreated();
  String busModified = busBasics.getModified();
  String busDescription = busBasics.getDescription();

  //get the current state of the businessobject
  State busState = getCurrentState(session, busObj);
  String busCurrentState = busState.getName();

%>
<center>
<table width="580" border=0>
  <tr>
    <td align=center bgcolor=white colspan=2><img
src=splashHome.gif></td>
  </tr>
    <tr>
      <td align=left bgcolor=white><b>Object Details: <b><%=
busType %>&nbsp;<%= busName %>&nbsp;<%= busRevision %></b></
td><td align=right><a href=SampleHome.jsp><img
src=buttonHome.gif border=0></a></td>
    </tr>
</table>
<form name="editAttributesForm" method="post">
  <input type=hidden name="busId"
value="<%=request.getParameter("busId")%>">
  <input type=hidden name="returnPage" value="<%=returnPage%>">

<center>
<table width="580" border=0>

  <tr>
    <th colspan=2 align=center>Basics</th>
  </tr>

  <tr class='odd'>
    <td width="30%">Owner</td>
    <td width="70%"><%= busOwner %>&nbsp;</td>
  </tr>

  <tr class='Even'>
    <td width="30%">Originated</td>
    <td width="70%"><%= busOriginated %>&nbsp;</td>
  </tr>

  <tr class='odd'>
    <td width="30%">Modified</td>
    <td width="70%"><%= busModified %>&nbsp;</td>
  </tr>

  <tr class='Even'>
```

```
    <td width="30%">State</td>
    <td width="70%">
      <%= busCurrentState %>&nbsp;
    </td>
  </tr>

  <tr class='odd'>
    <td width="30%">Description</td>
   <td width="70%" valign="top"><textarea name="busDescription"
rows="5" cols="30" wrap><%= busDescription %></
textarea>&nbsp;</td>
  </tr>
</table>

<table width="580">
  <tr>
    <th colspan="2" align=center>Attributes</th>
  </tr>
<%
  boolean colorFlag = true;
  String rowColor = "even";

  // read the related business object attributes
  AttributeItr attItr = new
AttributeItr(busObj.getAttributes(context).getAttributes());
  while (attItr.next()) {
    Attribute attribute = attItr.obj();
    AttributeType attrType = attribute.getAttributeType();
    attrType.open(context);
    if (colorFlag) {
      rowColor = "even";
      colorFlag = false;
    } else {
      rowColor = "odd";
      colorFlag = true;
    }
%>
    <tr class="<%= rowColor%>">
      <td width="30%"><%= attItr.obj().getName() %></td>
<%
    if (attribute.hasChoices()) {
%>
    <td width="70%"> <select name="<%=attribute.getName()%>" >
<%
    StringItr choiceItr = new StringItr(attribute.getChoices());
    while (choiceItr.next()) {
      String activeSelection = "";
      if (attribute.getValue().equals(choiceItr.obj())) {
          activeSelection = "Selected";
      }
%>
      <option <%=activeSelection %>
value="<%=choiceItr.obj()%>"><%=choiceItr.obj()%></option>
<%
    }
%>
    </select>
    &nbsp;</td>
<%
    } else if (attrType.getDataType().equals("real") ||
attrType.getDataType().equals("integer")) {
%>
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47"
onBlur=isNumberic(this)>&nbsp;</td>
<%

    } else if (attrType.getDataType().equals("timestamp")) {
%>
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47" >&nbsp;</td>
<%
    } else if (attribute.isMultiLine()) {
%>
      <td width="70%" valign="top"><textarea
name="<%=attribute.getName()%>" rows="5" cols="40"
wrap><%=attribute.getValue()%></textarea>&nbsp;</td>
<%
    } else {
%>
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47" >&nbsp;</td>
<%
    }
%>
    </tr>
<%
    attrType.close(context);
  }
%>
    <tr class="odd">
      <td colspan=2  align=right>
        <a href="<%=returnPage%>"><img src="buttonBack.gif"
border=0 alt="Previous Page" ></a>
<%
  int modifyAccess = 1;
  if (busObj.checkAccess(context, (short)modifyAccess)) {
%>
```

```
        <a href="javascript:editAttributes()"><img
src="buttonEdit.gif"  border=0 alt="Edit Attributes" ></a>
<%
  }
  busObj.close(context);
%>
       </td>
    </tr>
</table>
</form>
</center>

<!-- content ends here -->


</body>
</html>
```

## Code for Attribute Update Page (SampleObjectDetails.jsp)

```
<%-- SampleObjectDetails.jsp - Updates Attributes of any
Business Object
--%>

<%@ include file = "SampleUtility.jsp" %>

<%

  try {
    BusinessObject busObj = new
BusinessObject(request.getParameter("busId"));

    busObj.open(context);

busObj.setDescription(request.getParameter("busDescription"));

    AttributeItr attrItr = new
AttributeItr(busObj.getAttributes(context).getAttributes());
    AttributeList attrList = new AttributeList();
    while (attrItr.next())
    {
      Attribute attr = attrItr.obj();

      if (request.getParameter(attr.getName()) != null) {
        attr.setValue(request.getParameter(attr.getName()));
        attrList.addElement(attr);
      }
    }

    // update the attributes on the Business Object
    busObj.setAttributes(context, attrList);
    busObj.update(context);
    busObj.close(context);
    String pg = "SampleObjectDetailsDialog.jsp?busId=" +
request.getParameter("busId") ;
%>
    <jsp:forward page = "<%=pg%>" />
<%

  } catch (Exception e) {
    session.putValue("error.message", e.getMessage());
%>
    <jsp:forward page =
"<%=Framework.getCurrentPage(session)%>"/>
<%
    return;
  }
%>
```

## Tasks for Creating the Object Details Page

Some of the tasks involved in creating the object details page are the same tasks discussed in previous sections, so we won't describe them again here. These tasks include:

- Enter comments about the page. See *Entering Comments*.
- Write the page-level HTML code including the header, title, and body HTML tags. See *Including Page-Level HTML Tags*.
- Use an include directive to include a static file in the JSP (this is the SampleUtility.jsp). See *Including a Static File: Include Directive*. Also see other sections for help creating the SampleUtility.jsp.

Additionally, most the pages in the application reference a cascading style sheet called SampleStyleSheet.css so the look and feel of the pages is consistent. The style sheet is referenced using:

```
<link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >
```

The remaining tasks for creating the object details page can be broken down into these tasks:

- Display the object's basics and attributes. See *Displaying an Object's Details*.
- Show the Edit button only if the user has modify access for the object. When the user clicks Edit, submit the attributes to the SampleObjectDetails.jsp page, where the attributes for the object are updated. See *Updating the Object's Attributes*.
- Define the page to return to when the user clicks the Back button and when the object details are submitted to SampleObjectDetails.jsp. See *Defining a Return Page*.

### Displaying an Object's Details

To view or modify any business object, you must perform these three steps, each of which are described in detail below:

**1.** Open the object. See *Opening and closing a business object*.

2. Perform the access or modify task for the object.

   For the object details example used in Quotation Viewer, this step can be further broken down:

   a) Getting the object's type, name, and revision. See *Getting the type, name, and revision*.

   b) Get basics for the object. See *Getting basics*.

   c) Get the object's state. See *Getting the current state*.

   d) Display the basics and current state for the object. See *Displaying the object's TNR, basics, and state*.

   e) Display the object's attributes. See *Displaying attributes*.

3. Close the object.

### Opening and closing a business object

The object details page opens the current business object using the `open` method.

```
// open the current BusinessObject
  String busId = request.getParameter("busId");
  BusinessObject busObj = new BusinessObject(busId);
  busObj.open(context);
```

After all processes is complete, the page closes the object.

```
<%
  }
  busObj.close(context);
%>
```

Whenever you change or delete an object that has an `open` method (for example, business objects, types, formats, programs, etc.), you must use the `open` and `close` methods on that object. When you open an object to modify or delete, the ENOVIA Live Collaboration server creates a temporary object on the server to use in communicating with the client. After the object is closed, the Live Collaboration server deletes the temporary object. See the Studio Customization Toolkit Programmer's Online Reference (Javadocs) for details on which objects contain an `open` method.

For example, if you want to modify an instance of a business object, you would first call BusinessObject.open, make your modifications, then call BusinessObject.close.

When creating an object, you do not use the `open` method, since the ENOVIA Live Collaboration server creates the temporary object as a part of the `create` method. You should, however, use `close`.

### `open` and `close` Methods Not Required for Stateless Objects

Interfaces for stateless objects in Studio Customization Toolkit no longer require a call to open() or close(), thereby improving performance. Stateless objects include:

| | | |
|---|---|---|
| BusinessObject | BusinessType | Cue |
| Filter | Format | Group |
| Page | Person | Policy |
| Query | Relationship | RelationshipType |
| Report | Resource | Role |
| Store | Set | Table |

Tip ToolSet User

Vault

The context that is stored in the stateless object is NULL after a call to close(). Using any methods in a stateless class that don't take a context as a parameter require a call to open().

### Getting the type, name, and revision

Get the object’s type, name, and revision (TNR) using methods in the BusinessObject class of the db package.

```
//get the BusinessObject TNR
  String busType = busObj.getTypeName();
  String busName = busObj.getName();
  String busRevision = busObj.getRevision();
```

### Getting basics

Get basics for the object using methods in the BusinessObjectBasics class of the db package.

```
//get the BusinessObject basics
  BusinessObjectBasics busBasics = busObj.getBasics(context);
  String busOwner = busBasics.getOwner();
  String busOriginated = busBasics.getCreated();
  String busModified = busBasics.getModified();
  String busDescription = busBasics.getDescription();
```

### Getting the current state

One of the first things the SampleObjectDetailsDialog.jsp does is to declare a `getCurrentState` method that is used later in the page to get the current state of the object. To declare a method or variable in JSP, use the <%! syntax.

```
<%!
  //
  // return the current state for this object
  //
  static public State getCurrentState(HttpSession session,
                                      BusinessObject busObj)
throws MatrixException
  {
    matrix.db.Context context = Framework.getContext(session);

    // get the current state for this object
    StateItr stateItr = null;
    try {
      stateItr = new StateItr(busObj.getStates(context));
    }
    catch (MatrixException e)
    {
      busObj.open(context);
      stateItr = new StateItr(busObj.getStates(context));
      busObj.close(context);
    }
    finally {
      while(stateItr.next()){
        if (stateItr.obj().isCurrent()) {
          break;
        }
      }
      return stateItr.obj();
    }
  }
%>
```

Later in the page, the method is used to display the object's state:

```
//get the current state of the businessobject
  State busState = getCurrentState(session, busObj);
  String busCurrentState = busState.getName();
```

**Displaying the object's TNR, basics, and state**

The object details page displays the information gotten in the previous steps using expressions. For example, the type, name, and revision are displayed at the top of the object details page using:

```
<td align=left bgcolor=white><b>Object Details: <b><%= busType
%>&nbsp;<%= busName %>&nbsp;<%= busRevision %></b></td>
```

The object basics and current state are displayed in the Basics table. For example, this HTML code creates the row that displays the object owner.

```
<tr class='odd'>
    <td width="30%">Owner</td>
    <td width="70%"><%= busOwner %>&nbsp;</td>
  </tr>
```

**Displaying attributes**

To get all the attributes of a business object returned in an AttributeList, use the `getAttribute` method. Use the AttributeItr class to iterate over the attributes to display current values as well as set new values. The business object can then be updated using the new modified AttributeList.

```
// read the related business object attributes
  AttributeItr attItr = new
AttributeItr(busObj.getAttributes(context).getAttributes());
  while (attItr.next()) {
    Attribute attribute = attItr.obj();
    AttributeType attrType = attribute.getAttributeType();
    attrType.open(context);
```

The object details page needs to handle the display and update of attributes differently depending on the characteristics of the attributes, such as the data type, whether the attribute has ranges, and whether it is multiline. For example, if an attribute has range values, all values should be listed but the selected value should be highlighted.

```
<tr class="<%= rowColor%>">
      <td width="30%"><%= attItr.obj().getName() %></td>
<%
    if (attribute.hasChoices()) {
%>
    <td width="70%"> <select name="<%=attribute.getName()%>" >
<%
    StringItr choiceItr = new StringItr(attribute.getChoices());
    while (choiceItr.next()) {
      String activeSelection = "";
      if (attribute.getValue().equals(choiceItr.obj())) {
          activeSelection = "Selected";
      }
%>
      <option <%=activeSelection %>
value="<%=choiceItr.obj()%>"><%=choiceItr.obj()%></option>
<%
    }
%>
    </select>
    &nbsp;</td>
```

The input boxes on the object details page should also do some validation to make sure the user enters valid values. Near the top of the page, a function is defined that displays a message to let users know when they enter non-numeric values in a field that requires numeric values:

```
function isNumeric(textBox){
  var n = textBox.value;
  if(isNaN(n) == true) {
    alert("Please enter a numeric value");
    textBox.focus();
  }
}
```

The function is then called for attributes with data types that are real or integer.

```
<%
    } else if (attrType.getDataType().equals("real") ||
attrType.getDataType().equals("integer")) {
%>
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47"
onBlur=isNumberic(this)>&nbsp;</td>
```

The page also handles attributes with date/time values and multiline characteristics:

```
<%

    } else if (attrType.getDataType().equals("timestamp")) {
%>
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47" >&nbsp;</td>
<%
    } else if (attribute.isMultiLine()) {
%>
      <td width="70%" valign="top"><textarea
name="<%=attribute.getName()%>" rows="5" cols="40"
wrap><%=attribute.getValue()%></textarea>&nbsp;</td>
<%
    } else {
%>
```

```
      <td width="70%"  valign="top"><input type="text"
name="<%=attribute.getName()%>"
value="<%=attribute.getValue()%>" size="47" >&nbsp;</td>
<%
    }
%>
```

## Updating the Object's Attributes

This section describes the coding needed to update the objects attributes when the user clicks the Edit button.

**Sending updates to the SampleObjectDetails.jsp**

The object details page defines a function that sends the content of a form to SampleObjectDetails.jsp when the form is submitted:

```
function editAttributes() {
  document.editAttributesForm.action =
"SampleObjectDetails.jsp";
  document.editAttributesForm.submit();
}
```

The page only shows the Edit button if the user has modify access for the object. The page uses the `checkAccess` method in the BusinessObject class.

```
<%
  int modifyAccess = 1;
  if (busObj.checkAccess(context, (short)modifyAccess)) {
%>
```

When the user clicks the Edit button on the object details page, the editAttributes function is called.

```
<a href="javascript:editAttributes()"><img src="buttonEdit.gif"
border=0 alt="Edit Attributes" ></a>
```

**Updating the attributes**

The SampleObjectDetails.jsp page updates the object's attributes. First, it creates a new instance of the object submitted in the form on SampleObjectDetailsDialog.jsp and opens the object.

```
<%

  try {
    BusinessObject busObj = new
BusinessObject(request.getParameter("busId"));

    busObj.open(context);
```

Next, the page gets the object's attributes.

```
busObj.setDescription(request.getParameter("busDescription"));

    AttributeItr attrItr = new
AttributeItr(busObj.getAttributes(context).getAttributes());
    AttributeList attrList = new AttributeList();
    while (attrItr.next())
```

```
        {
          Attribute attr = attrItr.obj();

          if (request.getParameter(attr.getName()) != null) {
            attr.setValue(request.getParameter(attr.getName()));
            attrList.addElement(attr);
          }
        }
```

The page then sets the attributes and commits the changes to the object using the `setAttributes` and `update` methods. Then the page closes the object and forwards back to the SampleObjectDetailsDialog.jsp for the current object.

```
    // update the attributes on the Business Object
        busObj.setAttributes(context, attrList);
        busObj.update(context);
        busObj.close(context);
        String pg = "SampleObjectDetailsDialog.jsp?busId=" + 
    request.getParameter("busId") ;
    %>
        <jsp:forward page = "<%=pg%>" />
    <%
```

The page catches any errors and places them in the session object. If an error occurs within the try/catch block, the page forwards to the last completed page.

```
      } catch (Exception e) {
        session.putValue("error.message", e.getMessage());
    %>
        <jsp:forward page = 
    "<%=Framework.getCurrentPage(session)%>"/>
    <%
        return;
      }
    %>
```

## Defining a Return Page

Defining a return page lets you implement the Back button and lets you return to the object details page when the form is submitted to the SampleObjectDetails.jsp. The following code uses the REFERER header information to get the URL of the page that contains the link the user followed to get to the current page (unless returnPage is already defined).

```
    String returnPage= request.getParameter("returnPage");
      if (returnPage == null) {
        returnPage = request.getHeader("REFERER");
      }
```

The page then uses returnPage in an expression to specify the page to go to when the user clicks Back. For Quotation Viewer, this will be the Find Results page or the Checkin/ Checkout pages.

```
    <a href="<%=returnPage%>"><img src="buttonBack.gif"  border=0 
    alt="Previous Page" ></a>
```

The page also uses returnPage expression to return to the object details page when the form that contains the object details is submitted to the update page (SampleObjectDetails.jsp).

```
<form name="editAttributesForm" method="post">
  <input type=hidden name="busId"
value="<%=request.getParameter("busId")%>">
  <input type=hidden name="returnPage" value="<%=returnPage%>">
```

## Appending an Object to the End of a Revision Chain

You can add an object to the end of an existing revision sequence, as long as the object is not already a member of another revision sequence. Attempts to add an object in the middle of an existing revision chain, or to add an object that is already part of a revision sequence will cause the Studio Customization Toolkit method to throw a MatrixException.

Creating new revisions in ENOVIA Live Collaboration has several side affects:

- **Float/Replicate rules.** Ordinarily the float/replicate rules for relationships cause relationships to be moved/copied to new revisions. These rules are ignored when using these interfaces; no relationships are added or removed to/from the inserted object, nor are any relationships added or removed to/from any of the previously existing members of the target sequence.
- **File Inheritance.** Ordinarily, a new revision of an existing object inherits all files checked into the original object. When using these interfaces, if the appended object already has checked in files, it does not inherit any files from the revision sequence. If the appended object has no files checked in, the behavior is controlled by the file keyword in MQL or the inheritFiles argument in Studio Customization Toolkit (if true, inherits files from previous revision; if false, no inheritance).
- **Triggers.** No triggers will fire.
- **History.** Revision history records will be recorded on both the new object and its previous revision (that is, both objects that are specified in the MQL command or Studio Customization Toolkit method).

Both of the following interfaces (methods on “this” business object) will insert newbus as the revision following “this” in the latter's revision sequence:

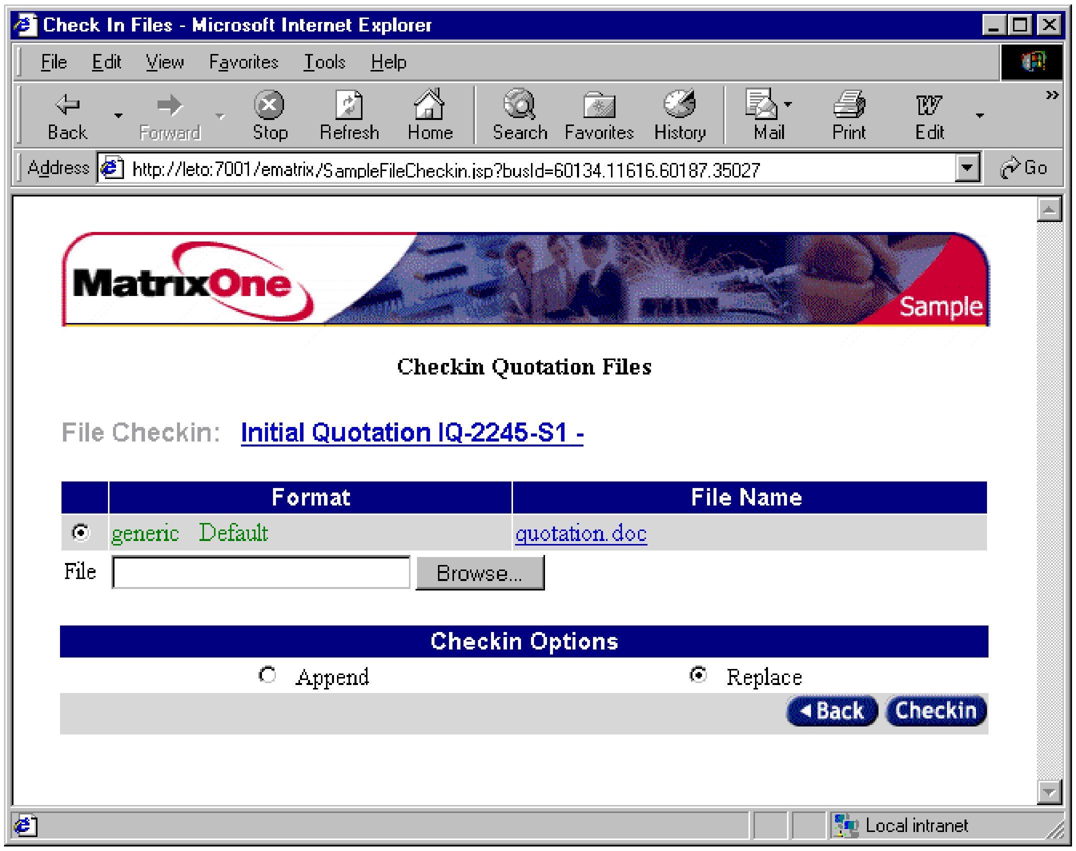
```
    public BusinessObject revise(Context context,
BusinessObject newbus,
String newvault)
            throws MatrixException
    {
        return (revise(context, (Visuals) null, newbus,
newvault));
    }



    public BusinessObject revise(Context context,
Visuals visuals,
BusinessObject newbus,
String newvault)
            throws MatrixException
```

### Example Code

```
    /**
     * Insert a revision of this BusinessObject
     *
     * @param context the context for this request
     * @param newbus the object to insert as a revision of this
     * @param newvault the vault to assign the revised
BusinessObject
     * @return a revision of this BusinessObject
     * @exception MatrixException when unable to retrieve data
from the collaboration server
     * @since ADK 6.2
     */
    public BusinessObject revise(Context context,
BusinessObject newbus,
String newvault)
            throws MatrixException
    {
        return (revise(context, (Visuals) null, newbus,
newvault));
    }

    /**
     * Insert a revision of this BusinessObject
     *
     * @param context the context for this request
     * @param visuals visuals to apply to this request
     * @param newbus the object to insert as a revision of this
     * @param newvault the vault to assign the revised
BusinessObject
     * @return a revision of this BusinessObject
     * @exception MatrixException when unable to retrieve data
from the collaboration server
     * @since ADK 7.0
     */
    public BusinessObject revise(Context context,
Visuals visuals,                             BusinessObject
newbus,
String newvault)
            throws MatrixException
```

## Checking In and Out Files

This section describes how to use the Studio Customization Toolkit to create features for checking in and checking out files. The following graphic shows the Checkin page for the Quotation Viewer application. This is the page that displays when the user clicks the file checkin icon on the Find Results page .


Check In Files - Microsoft Internet Explorer

Address: http://leto:7001/ematrix/SampleFileCheckin.jsp?busId=60134.11616.60187.35027

Checkin Quotation Files

File Checkin: Initial Quotation IQ-2245-S1 -

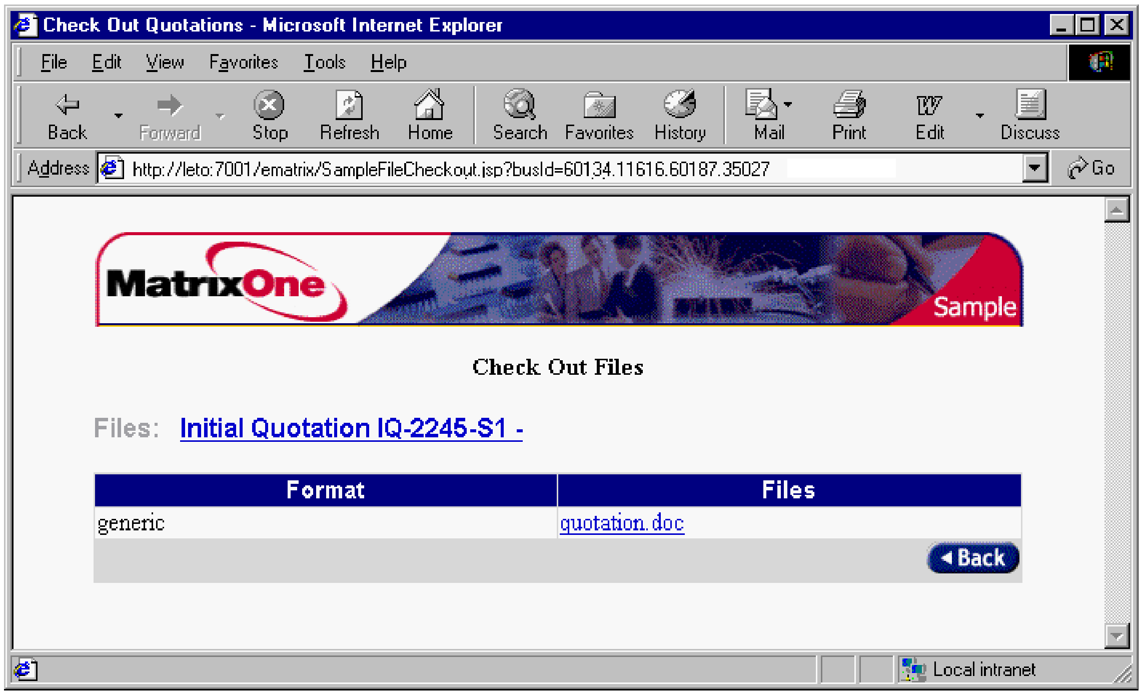
|  | Format | File Name |
|---|---|---|
| ● | generic Default | quotation.doc |

File [ ] Browse...

Checkin Options

○ Append ● Replace

When the user clicks the checkout icon on the find results page , this Checkout page displays.


Check Out Quotations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://leto:7001/ematrix/SampleFileCheckout.jsp?busId=60134.11616.60187.35027
Go
MatrixOne
Sample
Local intranet

Check Out Files

Files: Initial Quotation IQ-2245-S1 -

| Format | Files |
|---|---|
| generic | quotation.doc |

The checkin page performs these functions:

- Shows the selected object's type, name, and revision (if any).
- Lets the user click the object name to see object details.
- Shows the format and checked in files for the object.
- Lets the user check in files using the Append or Replace options.
- Lets the user check out a file by clicking on the file name.
- Lets the user back to the previous page.

The checkout page performs these functions:

- Shows the selected object's type, name, and revision (if any).
- Lets the user click the object name to see object details.
- Shows the format and checked in files for the object.
- Lets the user check out the file by clicking on the file name.
- Lets the user back to the previous page.

## Complete Sample Code

The checkin and checkout features of Quotation Viewer each have a JSP (not including the SampleUtility.jsp described in *Code for the Utility Page (SampleUtility.jsp)*): SampleFileCheckin.jsp and SampleFileCheckout.jsp. The sample code for both pages is shown below.

### Code for the Checkin Page (SampleFileCheckin.jsp)

The sample code shown below is the entire content of SampleCheckin.jsp. This JSP produces the Checkin page shown and described in the previous section.

```
<%--    SampleFileCheckin.jsp -- used for Checkin/out files into 
BusinessObject
--%>
<html>

<head>
<title>Check In Files</title>
</head>
<body>

<%@include file = "SampleUtility.jsp"%>
<link REL="stylesheet" type="text/css" 
HREF="SampleStyleSheet.css" >

<!-- content begins here -->
<script language=javascript>
function checkinFile(){
  if(document.checkinForm.file.value == "") {
    document.checkinForm.file.focus();
    return;
  }
  else {
    document.checkinForm.submit();
  }
}
function appendFunction(flag){
  document.checkinForm.append.value = flag;
  return;
}
</script>


<%
  String busId = request.getParameter("busId");
  BusinessObject busObject = new BusinessObject(busId);
  busObject.open(context);

  String type = busObject.getTypeName();
  String name = busObject.getName();
  String rev  = busObject.getRevision();

  // display Formats of the BusinessObject
   FormatItr formatItr = new 
FormatItr(busObject.getFormats(context));

  String checkinFileURL = Framework.encodeURL(response, 
FileCheckinServlet.getURL( busObject ));
  String defaultFormat = busObject.getDefaultFormat(context);
%>
```

```
<center>
<table width="580" border=0>
  <tr>
    <td align=center bgcolor=white><img src=splashHome.gif></td>
  </tr>
    <tr>
      <td align=center bgcolor=white><b>Checkin Quotation 
Files</b></td>
    </tr>
</table>
<table align=center class=inner width=580 cellspacing="0" 
cellpadding="1" bottommargin=2>
  <tr class="even">
    <td>&nbsp;</td>
  </tr>
  <tr class=even>
    <td align="left">
        <font face=arial size=3 color=#a9a9a1><b>File 
Checkin:&nbsp;&nbsp;
        <a class=one 
href="SampleObjectDetailsDialog.jsp?busId=<%=busObject.getObjec
tId()%>">
        <%=type%>&nbsp;<%=name%>&nbsp;<%=rev%></a></b>
      </font>
    </td>
  </tr>
</table>

<form name=checkinForm method="post" enctype="multipart/
form-data" action="<%=checkinFileURL %>">
<input type="hidden" name = "append" value = "false">
<table class = inner align=center cellspacing=1 width=70%>
  <tr>
    <th width="5%">&nbsp;</th>
    <th align=center >Format</th>
    <th >File Name&nbsp;</th>
  </tr>
<%
  int i = 0;
  int j = i % 2;
  String rowValue = "even";
  while (formatItr.next())
  {
    rowValue = "even";
    if (j == 0) {
      rowValue="odd";
    }
    Format format = formatItr.obj();
    String formatName = format.getName();
    String isDefaultFormat = "";
    boolean firstFile = true;
    if (formatName.equals(defaultFormat))
      isDefaultFormat = "checked";

    if (formatName.equals(defaultFormat)) {
%>
    <tr class=<%=rowValue%>>
      <td width="5%"><input type=radio name=format
<%=isDefaultFormat%> value="<%=formatItr.obj().getName()%>"> </
td>
      <td><font face="Times New Roman" color=green
size=3><%=formatItr.obj().getName()%> &nbsp;&nbsp;Default</
font></td>
<%
    } else {
%>
    <tr class=<%=rowValue%>>
      <td width="5%"><input type=radio name=format
<%=isDefaultFormat%> value="<%=formatItr.obj().getName()%>"> </
td>
      <td><font face="Times New Roman"
size=3><%=formatItr.obj().getName()%></font></td>
<%
    }

    // link the format if it contains a file
    if (!formatItr.obj().hasFiles()) {
%>
      <td>&nbsp;</td>
      </tr>
<%
    } else {
      // iterate over each file in the format
       FileItr fileItr = new
FileItr(busObject.getFiles(context,
formatItr.obj().getName()));
      while (fileItr.next())
      {
        // build the checkout url
        String fileStr = fileItr.obj().getName();
        String checkoutUrl =
FileCheckoutServlet.getURL(busObject) + "/" + fileStr
                             + "?format=" +
formatItr.obj().getName()
                             + "&file=" + fileStr;
        if (firstFile) {
%>
            <td >
            <a class=one HREF="<%=Framework.encodeURL(response,
checkoutUrl)%>"><font face="Times New Roman"
size=3><%=fileItr.obj().getName()%></font></a>
            </td>
          </tr>
<%
          firstFile = false;
        } else {
%>
          <tr class=<%=rowValue%>>
            <td width="5%">&nbsp;</td>
            <td>&nbsp;</td>
            <td>
             <a class=one
HREF="<%=checkoutUrl%>"><%=fileItr.obj().getName()%></a>
            </td>
          </tr>
<%
        }
      }
    }
    ++i;
    j = i % 2;
  }
  busObject.close(context);
%>
  <tr class=even><td colspan =3> </td></tr>
  <tr class=odd>
    <td width="5%"><font face="Times New Roman">File</font></td>
    <td colspan=2>  <input type=file name="file" size=25
value="">  </td>

  </tr>
</table>
<br>
<table class = inner align=center cellspacing=0 width=70%>
  <tr>
    <th align=center colspan=3>Checkin Options</th>
  </tr>
  <tr class=even>
      <td width="33%" align=right><font face="Times New
Roman"><input type="radio"  name="appendFlag" value="true"
onClick =
"javascript:appendFunction('true')">&nbsp;&nbsp;Append</font></
td>
      <td width="33%">&nbsp;</td>
      <td width="33%" align=left><font face="Times New
Roman"><input type="radio" name="appendFlag" checked
value="false" onClick =
"javascript:appendFunction('false')">&nbsp;&nbsp;Replace</
font></td>
  </tr>
  <tr class=even><td colspan =3> </td></tr>
  <tr class=odd>
    <td colspan=3 align=right>
```

```
   <a href="<%=request.getHeader("Referer")%>"><img
src=buttonBack.gif border=0 alt="Click here to return"></a>
   <a href="javascript:checkinFile()"><img
src="buttonCheckin.gif"  border=0 alt="Click here to check in
the file" ></a></td>
  </tr>
</table>
</form>

<!-- content ends here  -->

</body>
</html>
```

## Code for the Checkout Page (SampleFileCheckout.jsp)

```
<%--  SampleFileCheckout.jsp   -  This page allows file checkin/
out
--%>
<html>

<head>
<title>Check Out Quotations</title>
</head>
<body>

<%@include file = "SampleUtility.jsp"%>

<link REL="stylesheet" type="text/css"
HREF="SampleStyleSheet.css" >

<!-- content begins here -->

<%
  // read the related business object items
  BusinessObject busObj = new
BusinessObject(request.getParameter("busId"));
  busObj.open(context);
%>

<center>
<table width="580" border=0>
  <tr>
    <td align=center bgcolor=white><img src=splashDemo.gif></td>
  </tr>
    <tr>
      <td align=center bgcolor=white><b>Check Out Files</b></td>
    </tr>
</table>
<table align=center class=inner width=580 cellspacing="0"
cellpadding="0" bottommargin=2>
```

```
  <tr class="even">
    <td>&nbsp;</td>
  </tr>
  <tr class=even>
    <td align="left">
        <font face=arial size=3
color=#a9a9a1><b>Files:&nbsp;&nbsp;
        <a class=one
href="SampleObjectDetailsDialog.jsp?busId=<%=busObj.getObjectId
()%>">

<%=busObj.getTypeName()%>&nbsp;<%=busObj.getName()%>&nbsp;<%=bu
sObj.getRevision()%></b></a>
      </font>
    </td>
  </tr>
  <tr class="even">
    <td>&nbsp;</td>
  </tr>
</table>

<table align=center class=inner width=580 cellspacing="1"
border="0" cellpadding="1">
  <tr>
    <th align=center>Format</th>
    <th align=center>Files</th>
  </tr>
<%
  int i = 0;
  int j = i % 2;
  String rowValue = "odd";

  // loop over each format
  FormatItr formatItr = new
FormatItr(busObj.getFormats(context));
  while (formatItr.next())
  {
    rowValue = "odd";
    if (j == 0) {
      rowValue="even";
    }

    boolean newFormat = true;

    // link the format if it contains a file
    if (!formatItr.obj().hasFiles()) {
%>
      <tr class=<%=rowValue%>>
        <td width="50%"><%=formatItr.obj().getName()%></td>
        <td width="50%">&nbsp;</td>
      </tr>
<%
    } else {
      // iterate over each file in the format
        FileItr fileItr = new FileItr(busObj.getFiles(context,
formatItr.obj().getName()));
      while (fileItr.next())
      {
        // build the checkout url (temp work around for
destination name)
        String fileStr = fileItr.obj().getName();
        String checkoutUrl = FileCheckoutServlet.getURL(busObj)
+ "/" + fileStr
                             + "?format=" +
formatItr.obj().getName()
                             + "&file=" + fileStr;
        if (newFormat) {
%>
          <tr class=<%=rowValue%>>
            <td width="50%"><%=formatItr.obj().getName()%></td>
            <td width="50%">
            <a class=one HREF="<%=Framework.encodeURL(response,
checkoutUrl)%>"><%=fileItr.obj().getName()%></a>
            </td>
          </tr>
<%
          newFormat = false;
        } else {
%>
          <tr class=<%=rowValue%>>
            <td width="50%">&nbsp;</td>
            <td width="50%">
             <a class=one
HREF="<%=checkoutUrl%>"><%=fileItr.obj().getName()%></a>
            </td>
          </tr>
<%
        }
      }
    }
    ++i;
    j = i % 2;
  }
  busObj.close(context);
%>
  <tr class=odd>
    <td colspan=2 align=right>
      <a href="<%=request.getHeader("Referer")%>"><img
src=buttonBack.gif border=0></a>
    </td>
  </tr>
</table>
```

```
<!-- content ends here -->

</body>
</html>
```

## Tasks for Creating the Checkin and Checkout Pages

The checkin and checkout pages contain the standard elements described in previous chapters such as JSP comments, page-level HTML tags, a link to the style sheet, and the inclusion of the SampleUtilility.jsp.

The remaining tasks for creating the checkin and checkout pages can be broken down into these tasks:

*Tasks accomplished in SampleFileCheckin.jsp*

- Show the type, name, and revision for the object and let the user see details for the object. See *Providing Links to Object Details*.
- Show the formats for the object type.
- Show any files that are already checked in and let the user check them out.
- Let the user enter a file to check in or browse for it. Provide options for appending or replacing existing files.
- Let users return to the page they came from.

*Tasks accomplished in SampleFileCheckout.jsp*

- Show the type, name, and revision for the object and let the user see details for the object. See *Providing Links to Object Details*.
- Show the formats for the object type.
- Show all checked in files and let the user check them out.
- Let users return to the page they came from.

### Providing Links to Object Details

Both the checkin and checkout pages list the type, name, and revision for the object. Clicking on the type, name, and revision brings the user to the object details page. Here is the code from the checkout page that provides this functionality.

```
<a class=one
href="SampleObjectDetailsDialog.jsp?busId=<%=busObj.getObjectId
()%>">

<%=busObj.getTypeName()%>&nbsp;<%=busObj.getName()%>&nbsp;<%=bu
sObj.getRevision()%></b></a>
```

## Creating a Logout Feature

This section describes how to use the Studio Customization Toolkit to provide a Logout option on the home page of an application. When a user chooses the Logout option, a SampleLogout.jsp is called which uses the LogoutServlet. The sample logout page:

- Disconnects the user from the database.
- Removes the user's context, along with all data stored in the session (this happens after a specific period of time as determined by the Web server).
- Displays the login page.

The SampleLogout.jsp does not display to the browser.

### Complete Sample Code

The sample code shown below is the entire content of SampleLogout.jsp. This JSP performs the functions described in the previous section. The remaining portions of this chapter explain the commands in this code.

```
<%--  SampleLogout.jsp   -- disconnects the user from Matrix

--%>

<%@include file = "SampleUtility.jsp"%>
<jsp:include page = "<%=LogoutServlet.getURL()%>"/>


<%
response.sendRedirect("SampleLogin.jsp");
%>
```

### Tasks for Creating the Logout Page

Creating the SampleLogout.jsp can be broken down into several tasks:

- Enter comments for the page. See *Entering Comments*.
- Use an include directive to include a static file in the JSP (this is the SampleUtility.jsp). See *Including Page-Level HTML Tags*. Also see other sections of Chapter 4 for help creating the SampleUtility.jsp.
- Use the jsp:include action to dynamically call the LogoutServlet. See *Importing Java Classes*.
- Forward to the login page. See *Establishing the JSP Path*.

#### Including a Page or Servlet Dynamically

The SampleLogout.jsp uses a dynamic include action statement to call the LogoutServlet. The getURL method gives the URL for the servlet.

```
<jsp:include page = "<%=LogoutServlet.getURL()%>"/>
```

Unlike the include directive, the dynamic include action inserts the file at the time the page is requested. The JSP engine detects any changes to the included file and regenerates the servlet. The dynamic include is handy because any changes made to the included page are detected whenever the JSP is called. However, the dynamic include pays a high price in runtime processing time and can cause performance problems if used too much.

For information about using the static include page directive, see *Including a Static File: Include Directive*.

## Redirecting to the Login Page

The SampleLogout.jsp uses the sendRedirect method to redirect the browser to the login page.

```
<%
response.sendRedirect("SampleLogin.jsp");
%>
```

# 5 Creating a Custom Applet

## Applet Overview

There are several applets provided with the ENOVIA Live Collaboration Studio Customization Toolkit. Applets are simply containers that are filled with components and reside inside a Web page. The components could be labels with text fields, images, or buttons. For information on filling containers, refer to *A Main Application Dialog*.

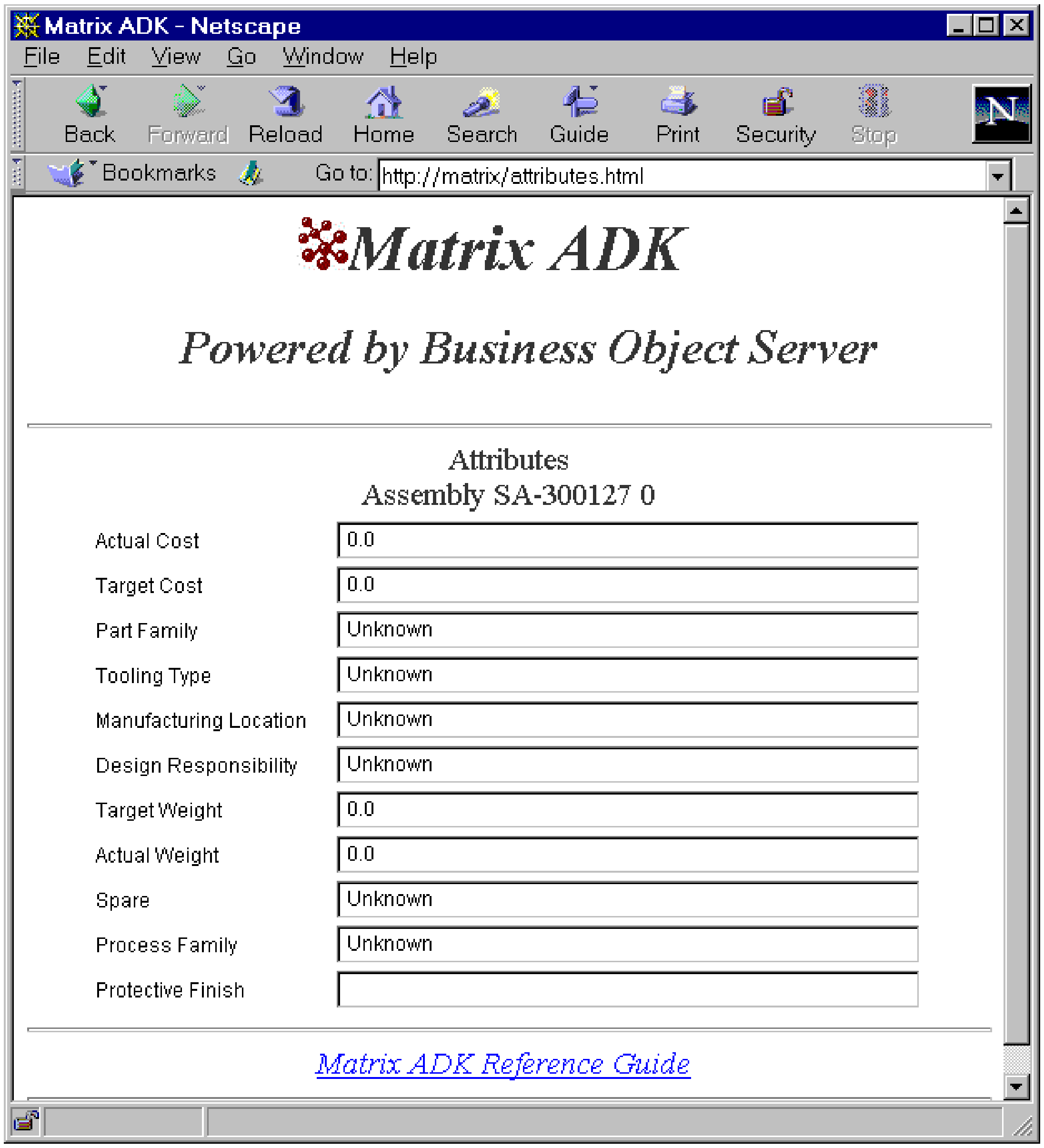
The applets mimic the ENOVIA Live Collaboration dialogs for which they are named because they contain some of the same components. For example, the Attribute applet is comprised of the same components as the Attributes dialog: the Description label with scrolling description field, attribute1 label with field, attribute2 label with field, and so on. Refer to the *Inventory of Provided Source Code* in Chapter 3 for a list of provided applets.

The applets must receive parameters to control their behavior and display the appropriate information. These parameters could be passed through Java programs or hard-coded directly into an HTML page. In any case, an HTML page that passes the parameters is generated. Below is part of the Attributes.HTML page. Notice that a valid context must be passed in order to access the information in the ENOVIA Live Collaboration database. The type, name, and revision of a business object must also be passed.

**Code Sample - Attributes.HTML**

```
<CENTER><APPLET CODEBASE=
   "http://yosemite.matrixnet.com/matrix/classes"
   CODE="matrix.applet.Attributes.class"
   WIDTH=500 HEIGHT=300>
<PARAM name="Host" value="yosemite.matrixnet.com">
<PARAM name="Server" value=":bos">
<PARAM name="Protocol" value="http:">
<PARAM name="Background" value="#ffffff">
<PARAM name="User" value="Des">
<PARAM name="Password" value="">
<PARAM name="Type" value="Assembly">
<PARAM name="Name" value="SA-300127">
<PARAM name="Revision" value="0">
</APPLET></CENTER>
```

The page would be displayed as shown below.


Matrix ADK - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Go to: http://matrix/attributes.html
Matrix ADK
Powered by Business Object Server
Matrix ADK Reference Guide

Attributes
Assembly SA-300127 0

| | |
|---|---|
| Actual Cost | 0.0 |
| Target Cost | 0.0 |
| Part Family | Unknown |
| Tooling Type | Unknown |
| Manufacturing Location | Unknown |
| Design Responsibility | Unknown |
| Target Weight | 0.0 |
| Actual Weight | 0.0 |
| Spare | Unknown |
| Process Family | Unknown |
| Protective Finish | |

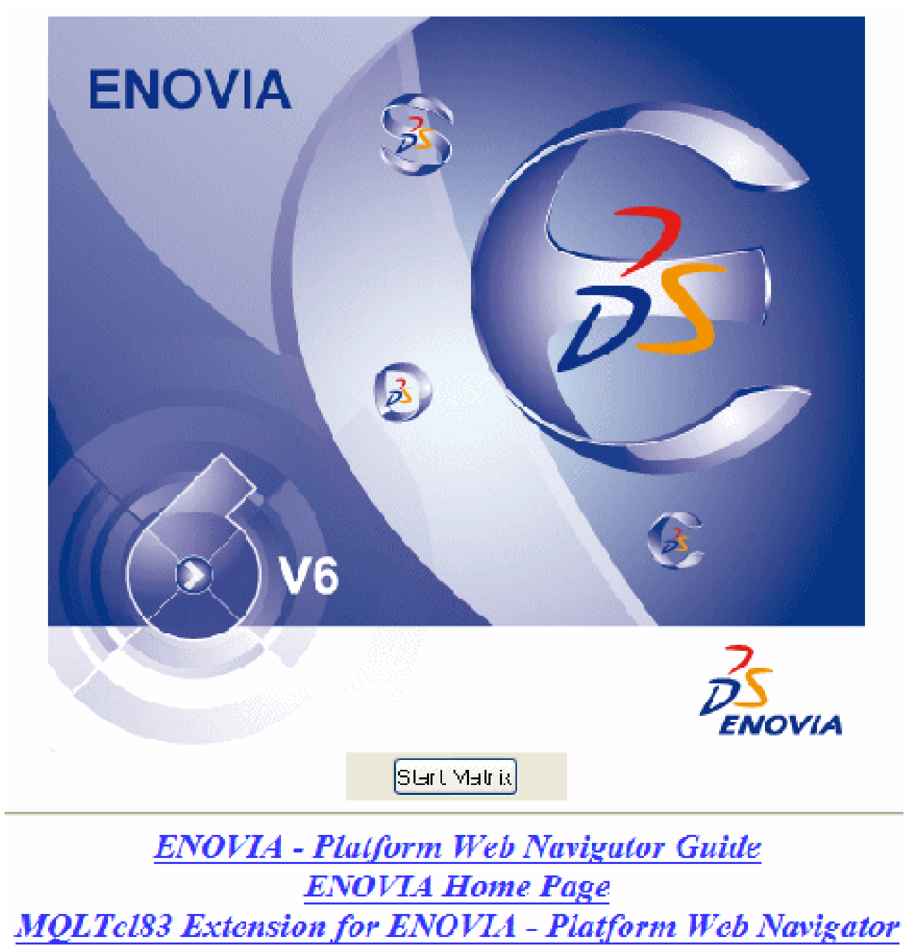
Notice that action buttons, tool buttons, and menus are not included. Since these applets, with all their components, live inside the page, the communication mechanism for parameters is one-way only: from the HTML page to the applet. However, applets may contain buttons that launch dialogs that live outside the page, so that a dialog approach can be used.

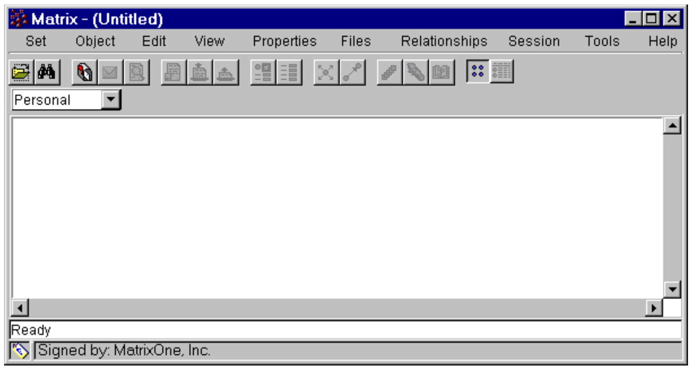
## Living Outside the Page

However, they are not constrained to stay there. Buttons can be placed in the applet container that have the ability to display a completely new window.

### Example

The `matrix.HTML` file contains the Matrix Web Navigator applet. The Start Matrix button is the applet that lives inside the page. Code within the HTML page tells it where to display and how big it should be, as well as how to connect to the ENOVIA Live Collaboration Servers. When the button is pressed, a new window that contains components of the ENOVIA Studio Customization Toolkit is presented.


ENOVIA
V6
ENOVIA
Start Matrix
ENOVIA - Platform Web Navigator Guide
ENOVIA Home Page
MQLTcl83 Extension for ENOVIA - Platform Web Navigator



## Creating a New Window

You can use the Window class, provided by Java, to create (instantiate) a new object of the class. Each class, which is a definition of how the object is going to behave, is a template containing all the attributes of the object. You use the name of the class to instantiate a new object of that class.

You can think of the class as a blueprint and the object as the actual thing the blueprint represents.

We have created a class called MainApplicationDialog, which is derived from the Java AWT Window class. You should instantiate an object of that class and call the show method to make the window visible. The requestFocus method will make the dialog box active.

### Code sample - Window Class

```
MainApplicationDialog _matrixDialog;
_matrixDialog = new MainApplicationDialog(_context, this);
_matrixDialog.show();
_matrixDialog.requestFocus();
```

## A Main Application Dialog

The name of the class/window that ENOVIA Live Collaboration uses, as provided in the Studio Customization Toolkit, is MainApplicationDialog. You can keep this name for your application, or you can copy and change the name to whatever name you want to use.

Before opening your applet window, you should add menu bars and toolbars to the window.

Java provides you with a JMenuBar class, a JMenu class, and a JMenuItem class that you can use to add menus to the window. ENOVIA Live Collaboration provides you with a Toolbar class and an Image Button class that you can use to add toolbars to your window.

### Adding Menu Bars, Menus, and Menu Items

You can use each of the available Java JMenu classes to create the menu bar, menu, and menu item objects in the way you want them to appear in your window. For each class, you will be adding the text that will be displayed on the menu bar and on the menu when it is selected.

To the JMenuBar class, you add menus, and to the Menu class, you add the menu items. Add the menu items in the order you want them to appear on the drop-down menu.

You should use the JMenuBar method to create the menu bar by providing the text that you want to have displayed in the menu bar. Then, use the JMenu method to create each of the menus. Finally, create menu items for each menu, providing the text in the order in which the items will be displayed when the menu is selected.

Save the JMenuItem objects in the MainApplicationDialog class so they are available to compare to an event later on.

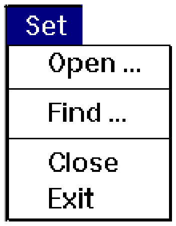
The code sample below shows you how to add menu bars, menus, and menu items.

**Code sample - Menu Bar/Menu/Menu Item**

```
JMenuItem _openSetMenuItem;
JMenuItem _findMenuItem;
JMenuItem _closeMenuItem;
JMenuItem _exitMenuItem;
/**
* Initialize the menu.
*/
private void initMenu()
{
   JMenuBar mb = new JMenuBar();
   setMenuBar(mb);

   Menu menu;

   // Set
   menu = new JMenu (“Set”);
   mb.add(menu);
   menu.add(_openSetMenuItem = new JMenuItem("Open");
   menu.addSeparator();
```

```
        menu.add(_findMenuItem = new JMenu-Item("Find");
        menu.addSeparator();
        menu.add(_closeMenuItem = new JMenuItem("Close");
        menu.add(_exitMenuItem = new JMenuItem("Exit");
    }
```

The code shown above will create this menu:



## Adding Toolbars and Toolbar Buttons

You can use each of the available ENOVIA Live Collaboration classes to create the toolbar and the toolbar button objects in the way you want them to appear in your window. For each Toolbar class, you will be adding the buttons that will be displayed on the toolbar.

First, instantiate a new toolbar and add the toolbar buttons to it. Create each of the ImageButtons, from left to right, in the order you want them to be displayed on the toolbar.

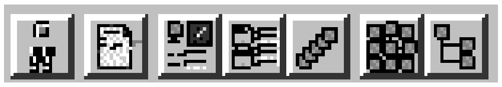
Be sure to save the ImageButton objects in the MainApplicationDialog class so they are available to compare to an event later on.

### Code sample - Toolbar/Toolbar Buttons

```
ImageButton _findButton;
ImageButton _viewButton;
ImageButton _attributesButton;
ImageButton _statesButton;
ImageButton _indentedButton;
/**
 * Initialize the toolbar.
 *
 * @return the tool bar
 * @since ADK 2.0
 */
private Toolbar initToolbar()
{
    ImageButton button;
    Toolbar toolbar = new Toolbar(5,0);
    _findButton =
new ImageButton(ImageCache.getImage(new FindGIFResource()));
    _findButton.addActionListener(this);
    toolbar.add(_findButton);
    toolbar.addSpacer(3);
    _viewButton =
new ImageButton(ImageCache.getImage(new ViewGIFResource()));
    _viewButton.addActionListener(this);
    toolbar.add(_viewButton);
```

```
        toolbar.addSpacer(3);
        _formatButton =
    new ImageButton(ImageCache.getImage(new FormatGIFResource()));
        _formatButton.addActionListener(this);
        toolbar.add(_formatButton);
        toolbar.addSpacer(3);
        _attributesButton = new ImageButton(ImageCache.getImage(
    new AttributeGIFResource()));
        _attributesButton.addActionListener(this);
        toolbar.add(_attributesButton);
        toolbar.addSpacer(3);
        _indentedButton = new ImageButton(ImageCache.getImage(
    new IndentedGIFResource()));
        _indentedButton.addActionListener(this);
        toolbar.add(_indentedButton);
        toolbar.addSpacer(3);
        _statesButton = new ImageButton(ImageCache.getImage(
    new StatesGIFResource()));
        _statesButton.addActionListener(this);
        toolbar.add(_statesButton);
        return (toolbar);
    }
```

The code shown above will create this toolbar:



## Laying Out the UI with Layout Managers

Layout managers are provided by Java to make it easy for you to lay out components in a window. Java provides five different layout managers. The Studio Customization Toolkit uses the two layout managers that are explained here: BorderLayout and GridBagLayout.

The layout managers allow you to set up the rules for how you want the components to appear in the window. You can have several buttons or several label fields of varying lengths and have all of them line up in one area of the screen.

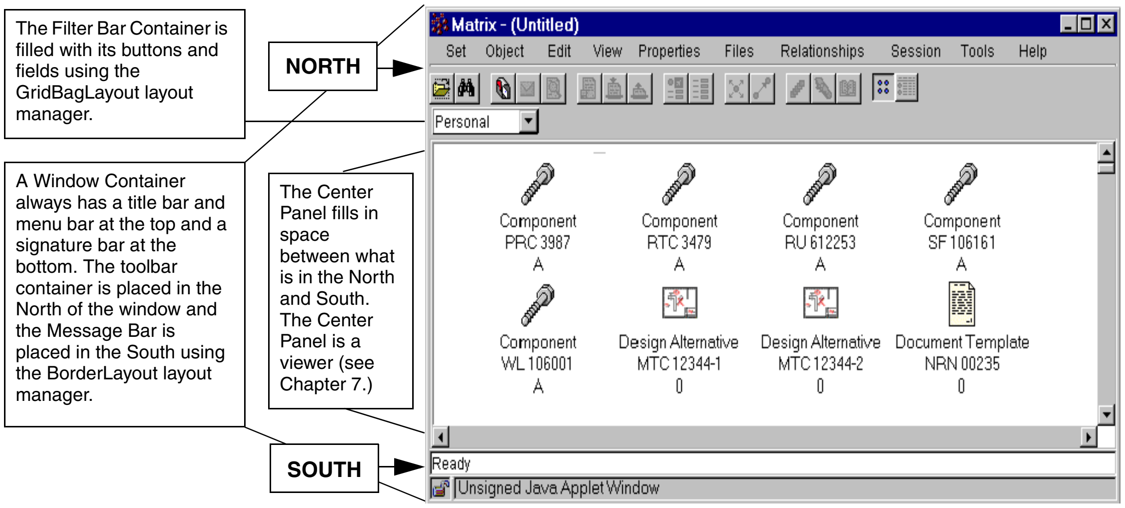
For example, on different platforms, buttons can be different sizes; or, with different languages, a label can contain more letters in German, for example, than in English. These layout managers help you define how you want the components to appear in the window and then line them up and position them for you. The positions are retained so your interface can always have a clean and professional appearance, regardless of the platform on which the application is run or the way a user resizes the window.

Some of the features of the BorderLayout and the GridBagLayout are explained here. For additional information on these two layout managers, or on the other layout managers, refer to the documentation provided with your Java Studio Customization Toolkit.

### Using Layout Managers

You can use the BorderLayout manager to position and size the major elements of your screen. For example, BorderLayout lets you put the toolbar in the North – always at the top of the screen, and lets you put the Action Buttons in the South – always on the bottom of the screen. The viewing area is in the Center and will increase or decrease in size to fill the remaining space. The object defined to be in the North will always go on the top; the object defined to go in the South will always go on the bottom; the object defined to go in the Center will always fill the remaining space on the screen.

This is the Main Application Dialog with populated viewer:


The Filter Bar Container is filled with its buttons and fields using the GridBagLayout layout manager.
A Window Container always has a title bar and menu bar at the top and a signature bar at the bottom. The toolbar container is placed in the North of the window and the Message Bar is placed in the South using the BorderLayout layout manager.
NORTH
The Center Panel fills in space between what is in the North and South. The Center Panel is a viewer (see Chapter 7.)
SOUTH
Matrix - (Untitled)
Set Object Edit View Properties Files Relationships Session Tools Help
Personal
Component PRC 3987 A
Component RTC 3479 A
Component RU 612253 A
Component SF 106161 A
Component WL 106001 A
Design Alternative MTC 12344-1 0
Design Alternative MTC 12344-2 0
Document Template NRN 00235 0
Ready
Unsigned Java Applet Window

You can use the GridBagLayout manager to position the components of the viewing area such as labels, text fields, radio buttons, and so on. These objects would live in a Panel, which is a container. (Panel is a class provided by Java).

Anything into which you can add components or into which you can put objects is a container. Each container can have its own layout manager. A window is a container since menu bars, toolbars, icons, and so on, can be put into it. A toolbar is also a container since toolbar buttons can be added to it.

Each of these containers can have a different layout manager. You can use a different layout manager for different areas of the window. Each one does not know about the others. Each layout manager allows you to specify how large each component within the container should be. It then lays out the components intelligently. And since the container is a component, it is asked by another layout manager how large it is to be.

For the window, which is a container, the layout manager is BorderLayout. Inside the window container is a panel, which is also a container. The layout manager for this panel (the container within the window) is GridBagLayout. You can put several containers within a container and each container can have its own layout manager. This makes it easier for you to design and develop your screens and all of the objects within each screen.

## Handling Events

Any user interface component can have an actionPerformed method that defines what is to take place when a certain action is taken by the user. In the Studio Customization Toolkit, the actionPerformed method in the MainApplicationDialog class handles events for components added to it.

When you create menu items and toolbar buttons, you save a reference to each of these objects in the MainApplicationDialog class. Then, when an event is initiated, the actionPerformed method is called to test for a match. You now have an event to compare with the saved objects. You can have the event tested on all objects or some objects, determined by the code you have written, until you find the object that matches. The test will determine if the referenced object that had the current event matches a referenced object that you saved. When it finds a match, you can take the defined action.

For example, when you create the Find menu item, and when you create the toolbar button for Find, you save a reference to each of these objects in the MainApplicationDialog class. When the user clicks on the Find menu item *or* on the Find toolbar button, the actionPerformed method is called automatically to test the event that occurred – in this case, the Find menu item that was selected *or* the Find toolbar button that was clicked. You previously saved each of these objects and now you have an event to match against them. Your test is to determine if the event matches any object that was saved. The handleEvent method will search through the list of saved components until it finds a match. Since a toolbar function is a shortcut to a menu item, whether the event is the Find menu item *or* the Find toolbar button, the defined action will be taken.

### Code sample - actionPerformed Method

```
/**
 * ActionListener interface: an action event occurred
 *
 * @param event the event
 * @since ADK 5.0
 * @see ActionListener
 */
public void actionPerformed(ActionEvent event)
{
   if ((event.getSource() == _findMenuItem) ||
       (event.getSource() == _findButton))
   {
      // do find stuff
   }
}
```

## Displaying Other Dialogs

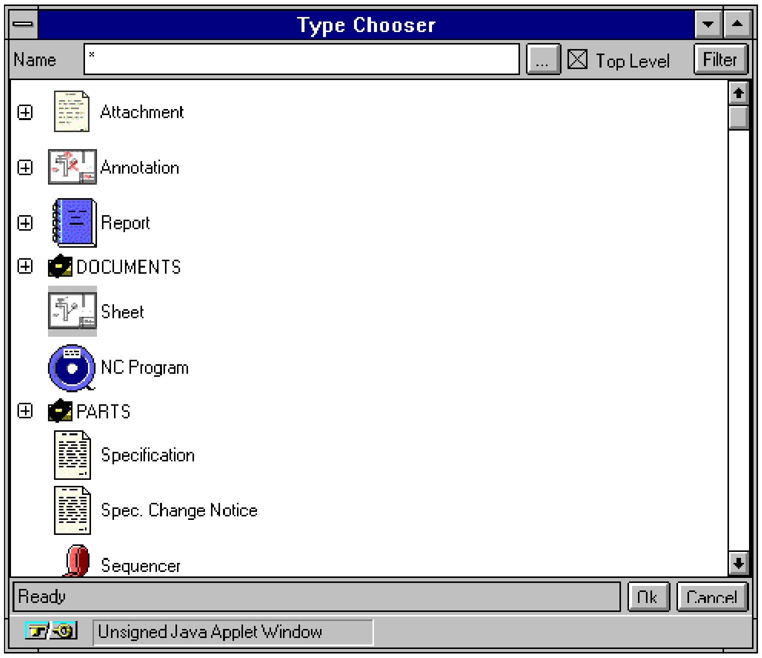
When you are using the MainApplicationDialog class, you are not restricted to only that dialog class. You can pop up other dialog boxes as well, just like any Windows program. The code shown below demonstrates how to create and show a new dialog.

### Code sample - Type Chooser Dialog

```
/**
 * Show the type chooser dialog
 */
public void showTypeChooserDialog()
{
   Window dialog =
       findHelperDialog(TypeChooserDialog.getIdentifier());
   if (dialog == null)
   {
      try
      {
          dialog = new TypeChooserDialog(getContext(), this);
          addHelperDialog(dialog);
      }
```

```
            catch (MatrixException e)
            {
               dialog = error(e);
            }
         }
         if (dialog != null)
             dialog.requestFocus();
      }
```

The Type Chooser window is shown below:



## Invoking the Applet

Since your applets will be used from a Web browser, they will be contained on an HTML page that is accessible from a Web server. When a server receives a request to send a file (HTML page) to a browser, it not only sends the file, but also sends information about the file so the browser can determine if it can display it. When the page contains a Java program, the applet is downloaded to the browser (if it is Java-enabled) to be run there, having received all the information it needs from the HTML file. Therefore, besides designing the look of the page, be sure to include all the information necessary to run the applet.

### HTML Page Parameters

The following is a partial list of parameters which are sent from the ENOVIA Live Collaboration HTML page so the applet can connect to the ENOVIA Live Collaboration server and obtain information from its database.

- Host—specifies the Host machine of the ENOVIA Live Collaboration server.
- Server—will always be :bos to specify the ENOVIA Live Collaboration server.
- Protocol—will generally be http:

- AutoStart—automatically starts the web user or a web wizard without having to press the start button (no start button is displayed). The applet will autostart if this parameter is found on the page; its value (true or false) is ignored.
- StartLabel—lets the user select the text for the applet start button, if a button is displayed.
- LAF—look and feel settings. Valid values are windows, metal, CDE/Motif.

For a complete list of parameters the ENOVIA Live Collaboration server supports refer to the *ENOVIA Live Collaboration Installation Guide*.

### Code Sample - HTML Page Parameters

```
<CENTER>
<APPLET CODEBASE="http://foo/matrix/classes"
CODE="matrix.matrix.Matrix.class" WIDTH=354 HEIGHT=380>
<PARAM name="Host" value="foo">
<PARAM name="Server" value=":bos">
<PARAM name="Protocol" value="http:">
<PARAM name="AutoStart" value="anything">
<PARAM name="StartLabel" value="start me">
<PARAM name="LAF" value="windows">
</APPLET>
</CENTER)
```

## Testing the Applet

Java allows you to include applets in an HTML page, similar to the way an image can be included. When the browser you are using views a page containing a Java applet, the code from the applet is copied to your computer and is executed by the browser.

For testing purposes, applets will also run in AppletViewer, a tool distributed in the Java Development Kit. The AppletViewer utility runs on your own PC, allowing you to view Java applets outside the context of a World-Wide Web browser.

### The AppletViewer Command

When you execute the AppletViewer command, it checks which URLs are defined in the command, connects to the document(s) or resource(s), and displays each applet referenced by that document in its own window.

*Note: If AppletViewer does not find any APPLET tags in the documents referred to by the URLs, it does nothing.*

```
appletviewer [-options] args ...
```

Possible [-options] values are:

| Option | Description |
|---|---|
| -debug | Starts AppletViewer in jdb (the Java debugger), allowing you to detect and correct errors in the applets. |
| -encoding *encoding name* | Allows you to specify the input HTML file *encoding name*. |
| -J *javaoption* | Passes the string *javaoption* as a single argument to the Java interpreter which runs the AppletViewer. Ensure that the argument does not contain spaces. If you include multiple arguments, each must begin with the prefix -J, which is stripped. This is useful for adjusting the compiler's execution environment or memory usage. |

The argument (args ...) can be one or more HTML files or URLs. The AppletViewer checks the code for APPLET tags and runs the applets as defined in the tags in separate windows.

## Platform-Specific Details

The following commands show platform differences in starting the AppletViewer.

| Platform | Command Syntax |
|---|---|
| UNIX or DOS shell (Windows): | appletviewer file:/newpage/java/Test.html |
| MacOS: | Start up the AppletViewer. From the File menu, choose OpenURL and enter the URL of the HTML file you created (for example, file:/newpage/java/Test.html). |

## Creating a User Interface

This section discusses strategies for making newly created ENOVIA Live Collaboration dialogs function appropriately. Because session context is the first action that must be performed when you are accessing the ENOVIA Live Collaboration database, we will use that dialog as an example throughout this section. Later, we will look at the Find dialog, because it has the ability to display other dialogs, called Choosers and pattern dialog boxes.

### Implementing Interfaces

An interface, in Java programming, is a list of basic methods that may be used in various ways by the different classes that implement them. They do not include the method code because in different contexts, the code would vary. When interfaces are implemented, the required methods must be defined in the implementing class, so there is code that can be referenced for the methods.

#### The DialogClient Interface

The DialogClient interface provides the following key methods:

- dialogDismissed
- dialogDataAvailable

The CommonDialog class implements the DialogClient interface.

**Code sample - DialogClient Interface**

```
public interface DialogClient
{
   abstract public void dialogDismissed(Frame dialog);
   abstract public void dialogDataAvailable(Frame dialog);
}
```

### The CommonDialog Class

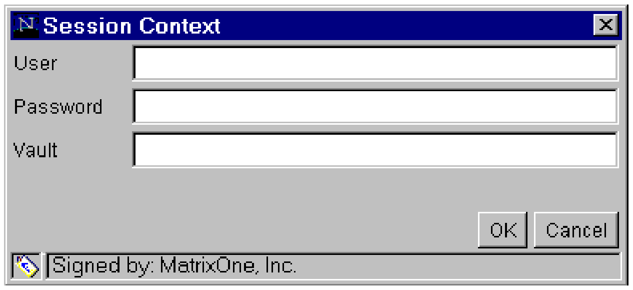
In Java programming, classes can be derived from other classes. This makes it possible to provide common methods at a low level, and then create derivations to enhance the functionality. Since many ENOVIA Live Collaboration dialogs are derived from the CommonDialog class, the functionality of this class is available to all dialogs. Therefore, the CommonDialog class contains:

- the *enter* method, which is called by clicking **OK** or pressing **Enter**.
- the *escape* method, which is called by pressing **Esc** or clicking on the **Cancel** button.
- the implementation of DialogClient.
- the methods associated with generating messages in the status bar.

Even though MainApplicationDialog and ContextDialog are found in the ENOVIA Live Collaboration package, it is the CommonDialog class that provides their basic functionality. Therefore, although it is possible for the CommonDialog class to be modified, there will generally be no reason to change it.

## ContextDialog

The Session ContextDialog has three fields: User, Password, and Vault, as shown below:



Signed by: MatrixOne, Inc.

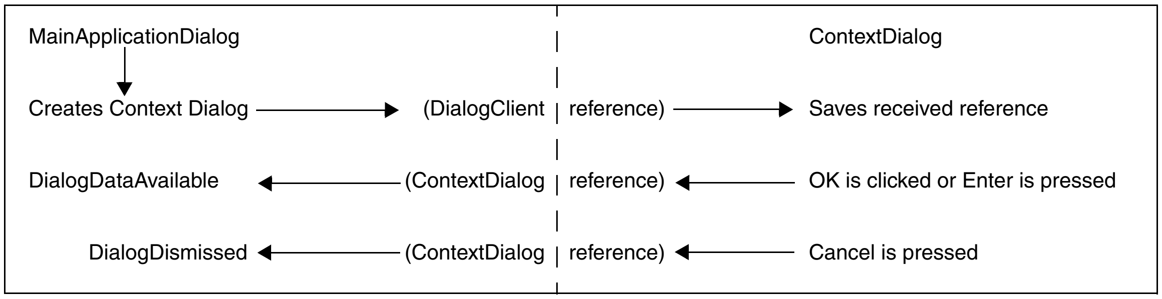
When you create a ContextDialog, the MainApplicationDialog passes a reference to the DialogClient interface, so that it knows where the information it receives should be delivered. When the user enters the information and the OK button is clicked, the following code is executed:

```
DialogClient.dialogDataAvailable
```

This action calls the dialogDataAvailable method in MainApplicationDialog and passes a reference to the ContextDialog. When the ContextDialog was instantiated in the MainApplicationDialog class, a reference to it was saved:

```
_contextDialog = ContextDialog.popup(getContext(), this);
```

The dialogDataAvailable method compares the received reference with all of the saved references in MainApplicationDialog, and knows what kind of information has become available.



### Code sample - CommonDialog Class

```
/**
 * A class representing a dialog common for
 * our applications.
 */
public class CommonDialog extends MxFrame implements
DialogClient, ActionListener
{
   .
   .
   .
   /**
    * ActionListener interface: an action event occurred
    *
    * @param event the event
    * @since ADK 2.0
```

```
       * @see ActionListener
       */
      public void actionPerformed(ActionEvent event)
      {
         if (event.getSource() == _cancelButton)
            escape();
         else if (event.getSource() == _okButton)
            enter();
      }

      /**
       * Override of the MxDialog enter method.
       * The ENTER key was pressed.
       *
       * @since ADK 2.0
       * @see MxDialog
       */
      public void enter()
      {
         _dialogClient.dialogDataAvailable(this);
         dismiss();
      }

      .
      .
      .
   }
```

## Getting Information from the Session Dialog

Because a reference to the dismissed dialog is returned when the dialogDataAvailable method is executed, we know what kind of dialog has data to be retrieved. In this case, it is: user, password, and vault. The ContextDialog methods are executed to capture the data.

### Code sample - Session ContextDialog

```
   /**
    * Set context from info in context dialog
    */
   private void setContext()
   {
      .
      .
      .
      String user = _contextDialog.getUser();
      String password = _contextDialog.getPassword();
      String vault = _contextDialog.getVault();

      getContext().setUser(user);
      getContext().setPassword(password);
      getContext().setVault(vault);
      .
      .
      .
   }
```

## Setting Context

The Context class has methods to set the user, the password, and the vault. This is the information needed to set a session context. These functions communicate with the ENOVIA Live Collaboration database through the ENOVIA Live Collaboration server, and pass the information to identify the user to ENOVIA Live Collaboration. As shown in the code sample below, connections to the server are established and then disconnected between commands.

### Code sample - Setting Context

```
try
{
   getContext().connect();
   getContext().disconnect();
}
catch (MatrixException e)
{
   error(e, MatrixErrorStrings.matrixCantSetContext);
}
try
{
   context context = new Context("HOST", "SERVER");
   context.setUser(USER);
   context.setPassword(PASSWORD);
   context.connect();
}
catch (MatrixException e)
{
   error(e, MatrixErrorStrings.matrixCantSetContext);
}
finally
{
   try
   {
       context.disconnect();
   }
   catch (MatrixException e)
   {
      //do nothing
   }
}
```

### Initializing the Context Object

To connect a standalone Studio Customization Toolkit application to ematrix.ini when running the ENOVIA Live Collaboration server in RIP mode, specify the host as “ ” or an empty string.

```
Context context = new Context("","");
```

The standalone application is directly connected to the database without going through the Live Collaboration server. The connection (bootstrap) file in the ENOVIA_INSTALL directory is used to connect to the database.

To connect a standalone Studio Customization Toolkit application via the application server, specify the application server’s hostname and port number.

In RIP mode (HTTP mode):

```
Context context = new Context("", "http://localhost:port/
ematrix");
```

In non-RIP mode:

```
Context context = new Context("", "localhost:port");
```

### Using the Context Object

The context object is not thread safe and it should only be used within the thread it is created. If it is necessary to use a context object in a new thread, you must make a clone of the context object. For example:

```
Context context;
Context newContext;

// ....

if (context != null) {
    try {
        newContext = new Context(null, context, null);
    } catch (MatrixException e) {
        newContext = null;
        // error handling
    }
}

if (newContext != null) {
    // use the newContext
}
```

### Closing Context

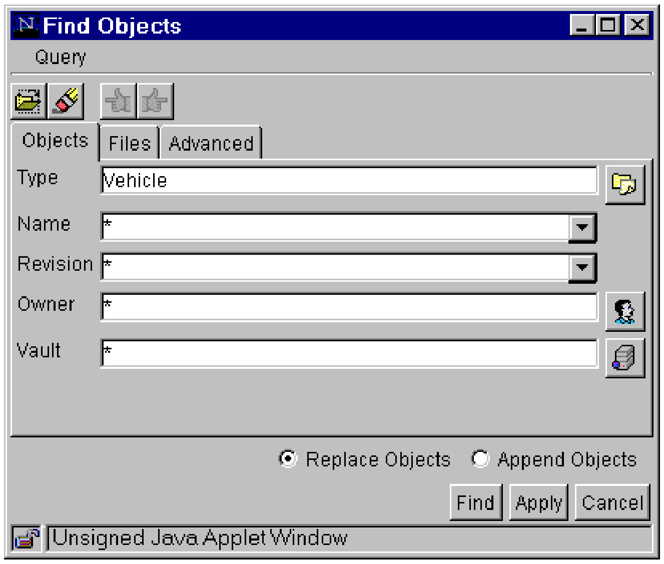
Use context.shutdown() to call both context.disconnect() and context.closeContext().

## The FindObjectsDialog

The FindObjectsDialog provides a graphical way to do a query in ENOVIA Live Collaboration. The results of the query will populate the viewer component of MainApplicationDialog.

Many of the fields provide buttons which are used to launch various “chooser” dialogs.

Pictured below is the Find Objects dialog box:



Type: Vehicle
Name: *
Revision: *
Owner: *
Vault: *

Replace Objects [✓] Append Objects

## Using the Chooser Dialogs

The “Chooser” dialogs are simple to use and are available if you prefer to use your mouse to select items, represented by icons, rather than entering text directly in a text field. The icons are graphical representations of all of the objects available for that field. When you click on your selection, the field will be filled in.

For example, to enter your user name in the Owner field, you can go to the list of all users and click on the icon representing yourself. Your own user name will then be placed in the Owner field.

### Code sample -VaultChooser Dialog

```
/**
 * ActionListener interface: an action event occurred
 * @param event the event
 * @since ADK 5.0
 * @see ActionListener
 */
public void actionPerformed(ActionEvent event)
{
   if (event.getSource() == _vaultButton)
   {
      Window dialog =
         findHelperDialog(VaultChooserDialog.getIdentifier());
      if (dialog == null)
      {
         try
         {
            dialog = new VaultChooserDialog(getContext(), this);
            addHelperDialog(dialog);
         }
         catch (MatrixException e)
         {
            dialog = error(e);
         }
      }
      if (dialog != null)
          dialog.requestFocus();
   }
}
```

## Viewing Business Objects

In ENOVIA Live Collaboration, when an object is opened for view, its checked-in file is opened in the application that is specified as the view program of the object in its default format. In order for this to work from within a Web browser, you must set up the proper associations between the viewing application and the browser.

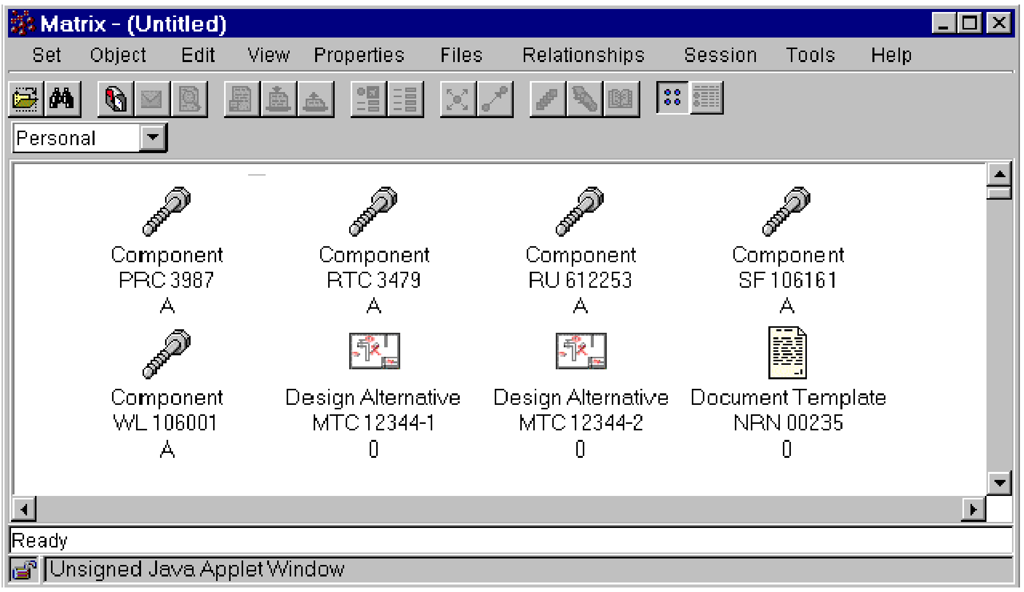
### Back To the Application Dialog

After a query is evaluated, a set of business objects (type-name-revision) is returned to the MainApplicationDialog. This dialog contains a method, which then opens each business object and retrieves its icon. It then creates a new BusinessObjectIcon for each object in the set and adds it to the viewer. The Viewer Sort method is then called to put them in the proper order, and then the Arrange method will line them up in rows and columns, based on the biggest icon and longest name.

### Loading Objects Into a Viewer

The viewer where the business objects are loaded is a FlatViewer; that is, it loads and displays the images from left to right, top to bottom. You are to instantiate a BusinessObjectIcon which is derived from a FlatViewerIcon for each business object by including the following components:

- the image
- the text that goes below the icon
- the viewer where it is going to reside
- the string by which it is to be sorted, with the type, name, and revision concatenated into one string



When you want to instantiate a new BusinessObjectIcon, specify the image you want to use for the icon and the text to appear below the icon. Add the image and text to the viewer. The image will be added in the correct position and be displayed properly.

The following code sample shows you how to create the BusinessObjectIcon and add it to the viewer based on the information in the business object. The code sample also shows you how to iterate through a list, in this case, a Set which is a list of BusinessObjects.

### Code sample - BusinessObject Icon

```
try
{
   if (_workingSet != null)
   {
      getContext().connect();
      BusinessObjectList list =
_workingSet.getBusinessObjects();
      if (getViewBy() == viewbyIcon)
         BusinessType.loadIcons(getContext(),
         BusinessType.getTypes(list), defaultImage);
      Component icons[] = new Component[list.size()];
      BusinessObjectItr itr = new BusinessObjectItr(list);
      while (itr.next())
      {
         message(StringResource.format(
               MatrixMessageStringResource.getString(
              MatrixMessageStringResource.LoadingObjectsStatus,
                  getContext().getLocale()),
               "current", Integer.toString(i + 1),
               "total", Integer.toString(list.size())));
         BusinessObject bo = itr.obj();
         if (matchesFilter(bo) == true)
         {
            icons[j] = constructIcon(bo);
            j = j + 1;
         }
         i = i + 1;
      }
      _objectViewer.sort(icons);
      _objectViewer.add(icons);
      _objectViewer.arrange();
      _objectViewer.resetScrollbars();
      getContext().disconnect();
      title = _workingSet.getName();
   }
}
catch (MatrixException e)
{
   error(e, MatrixErrorStringResource.getString(
               MatrixErrorStringResource.CantLoadObjects,
               getContext().getLocale()));
}
```

## Setting Up Associations

You can set up associations in Web browsers so the browser knows which application to open to display files of a particular type. The browser knows how to handle some extensions; for example, when the extension is `.gif`, the browser knows how to display a `.gif` file. But the browser needs to be told how to handle other extensions in the kinds of files you use. For example, you must tell the browser to open your word processing application when it sees the extension `.doc`, or to open your CAD/CAM application when it sees the extension `.cad`. Refer to your Web browser's on-line help.

## Selecting Business Objects: ViewerClient Interface

In the Viewer, an object can be selected by clicking on it with the mouse. The getSelections function, in the Viewer, returns a list of selected icons.

When you instantiate a Viewer, you provide a ViewerClient interface that is notified of certain events, such as when an object is selected in the Viewer, or when an object is double-clicked. These events are referred to as *selections changed* and *double-click*. The ViewerClient interface is usually implemented by the dialog where the Viewer resides.

The ViewerClient interface establishes a relationship between the Viewer, which is one panel of a dialog box, and the rest of the dialog box. If the selection has changed in the Viewer, you may have to take some action, such as graying out some menu items.

### Code sample - ViewerClient Interface

```
/**
* This interface is used to notify clients of certain events
occurring
* in the life of a Viewer.
*/
public interface ViewerClient
{
   /**
   * The viewer selections have changed.
   * @param viewer the viewer where the selections have changed.
   * @param selections the number of selected objects
   */
   abstract public void selectionChanged(Viewer viewer, int
selections);
   /**
   * A ViewerGizmo has been double clicked.
   * @param viewer the viewer where the double click occurred.
   * @param gizmo the object that was double clicked.
   */
   abstract public void doubleClick(Viewer viewer,ViewerGizmo
gizmo);
}
```

## Opening the Object for Viewing

Objects can be opened for viewing in three different ways:

- By selecting a toolbar button from the toolbar and clicking on it.
- By selecting a menu item from the menu bar and drop-down menu.
- By double-clicking on the object (an icon) with the mouse.

Any of these three methods produces the same effect, namely, opening the object for viewing.

## Toolbar or Menu

The following code sample shows you how to handle the event where an object has been selected using the toolbar or the menu.

### Code sample - Toolbar or Menu

```
/**
 * ActionListener interface: an action event occurred
 *
 * @param event the event
 * @since ADK 5.0
 * @see ActionListener
 */
public void actionPerformed(ActionEvent event)
{
    if ((event.getSource() == _viewMenuItem) ||
        (event.getSource() == _viewButton))
    {
        viewObject(_objectViewer.getSelections());
    }
}
```

## Double-Click on Object

The following code sample shows you how to handle the event where an object has been selected by double-clicking on it.

### Code sample - Double-Click

```
/**
 * ViewerClient interface: object was double clicked
 *
 * @param viewer the viewer containing the object
 * @param object the object that was double clicked
 * @since ADK 2.0
 * @see ViewerClient
 */
public void doubleClick(Viewer viewer, ViewerGizmo object)
{
   if (object instanceof BusinessObjectIcon)
      openForView(((BusinessObjectIcon)
object).getBusinessObject());
}
```

## Displaying the Object

The method called ViewObject, in the BusinessObjectDialog class, tells the browser that you want to view an object which has been selected by double-clicking on it. The browser opens a window and displays the object.

### Code sample - ViewObject

```
    /**
     * Opens files of default format for viewing
     * @param bo the business object to view
     */
    public void openForView(BusinessObject bo)
    {
       try
       {
          getContext().connect();
          getContext().createWorkspace();
          bo.open(getContext());
          String format = bo.getDefaultFormat(getContext());
          FileList files = bo.getFiles(getContext(), format);
          bo.checkoutFiles(getContext(), false, format,
    getContext().getWorkspacePath());
          FileItr itr = new FileItr(files);
          while (itr.next())
          {
             File file = itr.obj();
             getContext().showDocument(file.getName().true);
          }
          bo.close(getContext());
          getContext().disconnect();
       }
       catch (MatrixException e)
       {
          error(e);
          return;
       }
       catch (MalformedURLException e)
       {
          error(new MatrixException(e.toString()));
          return;
       }
}
```

## Getting Information from the Database

Information is retrieved from the database as an object. In the examples included in this chapter, the information from the database is written to the Java console, but it can be treated as any other Java object.

### Creating a Context Object

Before you can retrieve any information from the database, you must create a context object, which defines the ENOVIA Live Collaboration server application name and host name. You must then set the context.

When creating a context object using the Applet Studio Customization Toolkit, you must pass two parameters: server and host. The syntax is:

```
Context c = new Context("SERVER", "HOST");
```

The server was originally meaningful for CORBA but the parameter must be passed for XML and when using RMI on the back end. So for all architectures, the first parameter passed should be ":SERVER_NAME", which by default is ":bos".

The host differs for each architecture.

#### XML

For XML, the host is http://HOST:PORT. So if the host is "raid" and you are using the default port for WebLogic, the command would look like this:

```
Context c = new Context(":bos", "http://raid:7001");
```

#### RMI

You can also create a context when using RMI on the back end. In this case the host parameter should be //HOST:PORT. If the host is "raid" and you are using the default port for RMI, the command would look like this:

```
Context c = new Context(":bos", "//raid:1099");
```

The following code sample shows how to create a context object and set the context.

#### Code sample - Context

```
try
   {
        // create the context object
        Context context = new Context(server, host);

        // XML example:
        // Context c = new Context(":bos", "http://raid:7001");

        // RMI example:
        // Context c = new Context(":bos", "//raid:1099");

        context.setUser(user);
        context.setPassword(password);
        context.setVault(vault);

        // set the context
```

```
            getContext().connect();
            getContext().disconnect();
         }
         catch (MatrixException e)
         {
         // unable to set context
       }
```

## Administrative Objects

You can use the Studio Customization Toolkit to retrieve information from the database about administrative objects, including type, person, group, role, vault, and relationship types.

This section contains segments of code which shows how to get lists of information relating to administrative objects.

### List of Persons

The following segment of code shows how to get a list of persons from the database and output the information to the Java console. This code is similar to other Java classes for getting lists of administrative objects. See the specific Java class for the information you need.

#### Code sample - Person List

```
try
{
   _context.connect();
   PersonList list =  Person.getPersons(_context);
   _context.disconnect();

   PersonItr itr = new PersonItr(list);
   while(itr.next())
   {
      Person person = itr.obj();
      System.out.println("person: " + person.getImageName());
   }
}
catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

### Hierarchy: Types, Groups and Roles

You can request hierarchical information from the ENOVIA Live Collaboration database. Examples of hierarchical information are types, groups, and roles.

When you retrieve a list of hierarchical information from the database, also included are the associated parents (from which privileges have been inherited) and children (which inherit privileges from the type, group or role from which they were created).

The following is a code sample that shows how to get a list of groups, including parent and child groups.

### Code sample - Group Hierarchy

```
try
{
   _context.connect();
   Group group = new Group(_str);
   group.open(_context);

   GroupList topLevelList = group.getTopLevelGroups(_context);
   GroupItr topLevelItr = new GroupItr(toplevellist);
   while (topLevelItr.next())
   {
      Group topLevel = topLevelItr.obj();
      System.out.println("top level group: " +
toplevel.getName());
   }

   if (group.hasChildren() == true)
   {
      GroupList childrenList = group.getChildren();
      GroupItr childrenListItr = new GroupItr(childrenlist);
      while (childrenListItr.next())
      {
         Group child = childrenListItr.obj();
         System.out.println("group child: " + child.getName());
      }
   }
   else
   {
      System.out.println(" does not have any Children");
   }

   String Parent = group.getParent();
   int retvalue = Parent.compareTo("");
   if ( retvalue == 0)
   {
      System.out.println(" does not have any Parent");
   }
   else
      System.out.println("group parent: " + Parent);

   group.close(_context);
   _context.disconnect();
}
catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

## List of Vaults

A vault is a grouping of objects that depends on the types of objects used, the relationships they have to one another, and the people who need access to them.

The following code demonstrates how to get a list of vaults.

### Code sample - Vault List

```
try
{
   getContext().connect();
   VaultList list = Vault.getVaults(getContext());
   _context.disconnect();

   VaultItr itr = new VaultItr(list);
   while(itr.next())
   {
      Vault vault = itr.obj();
      System.out.println("vault: " + vault.getName());
   }

   getContext().disconnect();
}
catch (MatrixException e)
{
   // Can’t load vaults
}
```

## List of Relationship Types

In defining relationships between objects, the *type* specifies the types of business objects that can be connected with the relationship. The business object must have defined at least one type at each end in order for the relationship to be valid.

The following code demonstrates how to get a list of relationship types.

### Code sample - Relationship Types

```
try
{
   _context.connect();

   RelationshipTypeList list =
RelationshipType.getRelationshipTypes(_context);
   RelationshipTypeItr itr = new RelationshipTypeItr(list);
   while (itr.next())
   {
      RelationshipType type = itr.obj();
      System.out.println("Relationship Types: " +
type.getName());
```

```
      }
      _context.disconnect();
   }
   catch (MatrixException e)
   {
      System.out.println("an error occurred");
   }
```

## Launching Wizards

A wizard can be launched from a Java program in the same way that you can use the appl wizard command to invoke a wizard from a program in ENOVIA Live Collaboration. Wizards can only be invoked from top level programs or methods (including the wizard code that is executed from the finish button), and never from within user-defined transaction boundaries. Invoking a wizard from another program or from within an active transaction will produce an error. For more information on this limitation, see the "Invoking a Wizard from a Program" section of the *Business Modeler Guide*.

Here's an example of code that launches a wizard:

```
// Display a wizard
public void wizardtest(String wizardName)
{
    try
    {
        if ( wizardName == null || wizardName.equals(""))
            throw (new MatrixException(command));

        // Initialize variables to run wizard named "Wizard1"
        wizardName = "Wizard1";
        Program wizardProg = null;
        BusinessObject bus = null;
        StringList argList = new StringList(0);
        Relationship relation = null;
        boolean relationTo = false;

        // Create an argument as a tcl list for easy parsing
        // in the wizard's tcl code
        argList.addElement("{ {item1} {item2} {item3} }");

        // Find the program in the db with name=wizardName
        ProgramList programList =
Program.getPrograms(getContext());
        ProgramItr programItr = new ProgramItr(programList);
        while (programItr.next())
        {
            Program program = programItr.obj();
            if ( wizardName.equals(program.getName()) ) {
                wizardProg = new Program(program);
                break;
            }
        }
```

```
            // If we found a wizard, create/launch the dialog
            if ( wizardProg != null )
            {
                WizardDialog dialog =
                    new WizardDialog(getContext(), this,
                                     wizardProg,
                                     bus,
                                     argList,
                                     relation, relationTo);
                dialog.setVisible(true);
                dialog.requestFocus();
            }
        }
        catch (MatrixException e)
        {
            error(e);
        }
    }
```

## Getting Data Type Information

These get functions, added for version 9.5.3.0, let programmers obtain datatype information so an application can perform the appropriate formatting on the data.

```
Attribute.java
/**
* Get a data type for this attribute
*
* @return int that is the data type of this attribute
* @since ADK 9.5.3.0
*/
public short getType()

BusinessObjectWithSelect.java
/**
* Get a data type
*
* @return data type for the key
* @since ADK 9.5.3.0
*/
public short getSelectDataType(String key)

RelationshipWithSelect.java
/**
* Get a data type
*
* @return int that is the data type for the key of
target data
* @since ADK 9.5.3.0
*/
```

```
public short getTargetSelectDataType(String key)
/**
* Get a data type
*
* @return data type for the key
* @since ADK 9.5.3.0
*/
public short getSelectDataType(String key)

ExpansionWithSelect.java
/**
* Get a root data type
*
* @return int that is the data type for the key
* @since ADK 9.5.3.0
*/
public short getRootSelectDataType(String key)
```

## Business Object Information

Included in the sample code in Appendix A is a class called BusinessObjectDb.java which illustrates the use of various methods in the BusinessObject class, including how to get information related to a particular business object, how to promote a business object with signature approval, and how to create a new business object.

Shown below are methods demonstrating how to get information from the database about business objects.

### Basic Information

The basic information about a business object includes the type, name, revision, owner, policy name, description, creation date, and last modified date.

The code to obtain any of this information is fundamentally the same. Check the code in *Sample 1: BusinessObject Class* in Appendix  for particulars about the basic information you need.

The following method demonstrates how to get the name of the business object.

```
System.out.println(_object.getName());
```

The output from the execution of this method would look similar to this:

```
object.Drawing.50461_cln.A
```

### isOpen method

The isOpen method returns a Boolean that indicates if the object is currently open on the server. This method is inherited by most Studio Customization Toolkit objects.

The following demonstrates how to check if a business object is currently open on the server:

```
if (BUSINESS_OBJECT.isOpen()) {
   BUSINESS_OBJECT.open(context);
```

## Revisions

Revisions are copies of a business object in a changed state. Each revision is stored in ENOVIA Live Collaboration as a distinct business object, making available to you the current revision, or any prior revisions of the object.

The following code demonstrates how to get a list of revisions for an object.

### Code sample - Revisions

```
try
{
   getContext().connect();
   _object.open(getContext());
   BusinessObjectList list = _object.getRevisions(getContext());
   BusinessObjectItr itr = new BusinessObjectItr(list);
   while(itr.next())
   {
      BusinessObject bo = itr.obj();
      System.out.println(bo.getTypeName() + " " +
                         bo.getName() + " " +
                         bo.getRevision());
   }
   _object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
{
    // Can't load revisions
}
```

## History

The history of a business object details information about each activity that has taken place for the object from the time of its creation to its current state.

The following code demonstrates how to get historical information about an object.

### Code sample - History

```
try
{
   getContext().connect();
   _object.open(getContext());

   StringList list = _object.getHistory(getContext());
   StringItr itr = new StringItr(list);
   while (itr.next())
   {
      System.out.println((String) itr.value());
   }

   _object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
{
   // unable to get history
}
```

## List of Attributes for Business Object

The following is a segment of code shows how to extract a list of attributes for the specified business object from the database. Returned is a list including the attribute name, description, value, attribute type and image name.

### Code sample - Attributes

```
try
{
   BusinessObject _object =
new BusinessObject(_type,_name,_revision);
   _context.connect();
   _object.open(_context);
   System.out.println("description: " +
_object.getDescription());

   // Get the list of attributes from the business object
   AttributeList list = _object.getAttributes();
   AttributeItr itr = new AttributeItr(list);

   while (itr.next())
   {
      Attribute attribute = itr.obj();
      System.out.println("name: " + attribute.getName());
      System.out.println("value: " + attribute.getValue());
      AttributeType type_object = attribute.getAttributeType();
      System.out.println("imagename: " +
type_object.getImageName());
   }

   _object.close(_context);
   _context.disconnect();
}

catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

## States and Signatures

A business object generally has a life cycle consisting of a series of connected states, each of which represents a stage in the life of the object. Often, for an object to move from one state to another, someone needs to approve the promotion to another state. This is accomplished with approval signatures.

The following example demonstrates how to get the names of the states and the signatures for the business object.

**Code sample - States and Signatures**

```
try
{
   _context.connect();
   _object.open(_context);

   StateList stateList = _object.getStates(_context);
   StateItr stateItr = new StateItr(stateList);
   while (stateItr.next())
   {
      State state = stateItr.obj();
      System.out.println("state name: " + state.getName());

      StateBranchList branchList = stateItr.obj().getBranches();
      StateBranchItr branchItr = new StateBranchItr(branchList);
      while (branchItr.next())
      {
         Branch branch = branchItr.obj();
         System.out.println("branch name: " + branch.getName());

         SignatureList signatureList =
branchItr.obj().getSignatures();
         SignatureItr signatureItr = new
SignatureItr(signatureList);
         while (signatureItr.next())
         {
            Signature signature = signatureItr.obj();
            System.out.println("signature name: " +
signature.getName());
         }
      }
   }

   _object.close(_context);
   _context.disconnect();
}
catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

## Relationships for the Business Object

The following code demonstrates how to get a list of relationships for the given business object.

**Code sample - Relationships**

```
try
{
      _context.connect();
      _object.open(_context);

      // Get the getTorelationship for the given business object
      RelationshipList relToList =
   _object.getToRelationship(_context);
      RelationshipItr relToItr = new RelationshipItr(relToList);
      while (relToItr.next())
      {
         Relationship relTo = relToItr.obj();
        System.out.println("relationship to: " + relTo.getName());
      }

      // Get the getFromRelationship for the given business object
      RelationshipList relFromList =
   _object.getFromRelationship(_context);
      RelationshipItr relFromItr =
   new RelationshipItr(relFromList);
      while (relFromItr.next())
      {
         Relationship relFrom = relFromItr.obj();
         System.out.println("relationship from: " +
   relFrom.getName());
      }

      _object.close(_context);
      _context.disconnect();
   }
   catch (MatrixException e)
   {
      System.out.println("an error occurred");
   }
```

## List of Formats and Files

Files are stored in ENOVIA Live Collaboration associated with a particular business object and format. The format defines the application with which you can view or modify the files.

The following code samples demonstrate how to get a list of formats and how to get a list of files.

### Code sample - Format List

```
   try
   {
      getContext().connect();
      _object.open(getContext());
      FormatList list = _object.getFormats(getContext());
      FormatItr itr = new FormatItr(list);
      while (itr.next())
      {
         Format format = itr.obj();
         System.out.println("format:" + format.getName());
      }
      _object.close(getContext());
      getContext().disconnect();
   }
   catch (MatrixException e)
   {
      // Can't load formats
   }
```

### Code sample - File List

```
   try
   {
      _context.connect();
      _object.open(_context);

      FileList files = _object.getFiles(_context,_format);
      FileItr fileItr = new FileItr(files);
      while (fileItr.next())
      {
         File fileObject = fileItr.obj();
         System.out.println("name: " + fileObject.getName());
      }

      _object.close(_context);
      _context.disconnect();
   }
   catch (MatrixException e)
   {
      //Can’t load files
   }
```

## Personal Objects

### Query

ENOVIA Live Collaboration provides the ability to search for business objects using a specified set of search criteria. This function is called a *query*. The criteria can include vaults, types, names, revisions, owners and attributes. Once you have established the search criteria to use, you can save the query for later use.

Included in the sample code in Appendix A is a class called QueryDb.java which illustrates the use of various methods in the Query class, including how to use an existing query to return business objects.

#### Programming Notes for Queries

### Default .Finder Query Read by Default

By default, all users have the query named .finder, which contains the search criteria of the last query performed by that user. Thus, when a program calls getQueries(Context), at least this one query is returned.

A query constructed using the Studio Customization Toolkit constructor new Query() always references the default ".finder" query. Having constructed such a query:

- the open method reads the definition of .finder from the database and must be overridden by explicitly setting each field.

  When the open method of the query class is executed, the client executes the method on the corresponding server object, which opens the existing query in the database. The query fields (name pattern, type pattern, where pattern, etc.) are then communicated back to the client, and the client must either accept what is stored in the database or override it. This is similar to the mql modify query command, where only fields that are specified are changed on the server. Therefore, in order to assure expected results, Studio Customization Toolkit users must set query fields explicitly every time they open a query.
- the update method will write the current fields in the Java Query object back to the .finder definition in the database.
- the close method will release the Query object without updating the .finder definition in the database (nor clearing it).

It's possible to create a very temporary query by passing the emptystring as a name to the Query constructor:

```
Query q = new Query("")'
```

In this case:

- the open method initializes all fields to default values.
- the update method will error.
- the close method will release the Query object.

Creating unnamed queries in this manner is recommended because it creates a completely in-memory query that cannot conflict with any previously-existing queries in the database.

### Query Errors Reported in ClientTaskList, not MatrixException

Query errors are not be reported via the standard MatrixException. This allows some data to be returned even if a subset of the interfaces are unavailable. You must check the Context object's ClientTaskList for error messages after the query and throw a MatrixException of your own if needed.

Here is sample code that shows how to check the ClientTaskList after evaluating the query and throw a MatrixException if errors are found. The vault Scott is an adapted vault on Windows machine with the OracleTnsListener80 service stopped:

```
try
{
// create a new query
q.setVaultPattern(^Scott^);
q.setBusinessObjectType(^*^);
q.setBusinessObjectName(^*^);
q.setBusinessObjectRevision(^*^);
q.setOwnerPattern(^*^);
q.setWhereExpression(^^);
q.setObjectLimit((short)256);
q.setExpandType(true);
q.create(context);

// Evaluate the query (using the default visuals)
list = q.evaluate(context,context.getDefaultVisuals());
q.close(context);

System.out.println(^ Found bus=^ + list.size());

// Check client task list to see if there were any errors.
ClientTaskList tasks = context.getClientTasks();
System.out.println(^tasks.size() = ^ + tasks.size());
String errors = ^^;
for (int i = (tasks.size() - 1); i >= 0; i--) {
ClientTask task = (ClientTask) tasks.elementAt(i);
if (task.getReason() == ClientTask.reasonErrorMsg) {
tasks.removeElementAt(i);
if (errors != ^^)
errors += ^
^;
errors += task.getTaskData();
}
}
if (errors != ^^)
throw new MatrixException(errors);
} catch(MatrixException me) {
System.out.println(^an error occurred during query^);
me.printStackTrace();
}
```

**Creating a New Query**

The following example demonstrates how to create a new query and specify search criteria, then save the query.

### Code sample - New Query

```
try
{
   _query = new Query(_str);
   _context.connect();

   try
   {
      _query.open(_context);
   }
   catch (MatrixException e)
   {
   }

   _query.setVaultPattern(_vault);
```

```
      _query.setBusinessObjectType(_type);
      _query.setBusinessObjectName(_name);
      _query.setBusinessObjectRevision(_revision);
      _query.setOwnerPattern(_owner);
      _query.setWhereExpression(_whereclause);
      _query.setExpandType(true);

      _query.update(_context);
      _query.close(_context);
      _context.disconnect();
   }
   catch (MatrixException e)
   {
      System.out.println("an error occurred");
   }
```

**Evaluating the Query**

Once the query is created and saved, you can run the query to extract specific information from the database. The following segment of code shows how to extract a list of business objects from the database.

### Code sample - Query Evaluation

```
   try
   {
      _query = new Query(_str);
      _context.connect();
      _query.open(_context);

      BusinessObjectList list = _query.evaluate(_context);
      // Iterates through a list of business objects
      BusinessObjectItr itr = new BusinessObjectItr(list);
      while(itr.next())
      {
         BusinessObject businessObject = itr.obj();
         System.out.println("type: " +
   businessObject.getTypeName());
         System.out.println("name: " + businessObject.getName());
         System.out.println("rev: " +
   businessObject.getRevision());
      }
   }
   catch (MatrixException e)
   {
      System.out.println("an error occurred");
   }
```

## List of Saved Sets

A number of business objects can be grouped together and saved as a set. The following example shows how to get a list of saved sets from the database.

**Code sample - List of Sets**

```
try
{
   _context.connect();

   SetList list = Set.getSets(_context);
   SetItr itr = new SetItr(list);
   while (itr.next())
   {
      Set set = itr.obj();
      System.out.println("Sets: " + set.getName());
   }

   _context.disconnect();
}

catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

# Modifying the Database

The Studio Customization Toolkit provides the means to make changes to the database, including adding a new business object, assigning signatures to an object, and checking files in or out.

## Adding Objects: New Business Object

Business objects can be added to the ENOVIA Live Collaboration database. When adding objects though the API, the owner of the object is defined by how the context is set. The following code segment demonstrates how to create a new business object.

### Code sample - New Business Object

```
try
{
    BusinessObject businessObject = new
BusinessObject(_typeText, _nameText, _revisionText,
_vaultText);
    // if the business object does not exist, then
    // catch the exception and create it.
    try
    {
        businessObject.exists(getContext());
        MatrixException e = new
MatrixException(StringResource.format(
                          MatrixErrorStringResource.getString(

MatrixErrorStringResource.AlreadyExists,
                             getContext().getLocale()),
                          "type", _typeText,
                          "name", _nameText,
                          "revision", _revisionText));
        e.addMessage(StringResource.format(
                          MatrixErrorStringResource.getString(
                          MatrixErrorStringResource.CantCreate,
                             getContext().getLocale()),
                          "type", _typeText,
                          "name", _nameText,
                          "revision", _revisionText));
        error(e);
    }
    catch (MatrixException e)
    {
        businessObject.create(getContext(), _policyText);
        businessObject.close(getContext());
    }
}
catch (MatrixException e)
{
    e.addMessage(StringResource.format(
                          MatrixErrorStringResource.getString(
                           MatrixErrorStringResource.CantCreate,
                             getContext().getLocale()),
                          "type", _typeText,
                          "name", _nameText,
                          "revision", _revisionText));
    error(e);
    return;
}
```

## Signatures

Each business object is governed by a policy which defines the object's life cycle. Generally, in order to promote the object from one state of its life cycle to the next, one or more signature approvals is required. A person whose signature is required can choose a rejection signature, alerting others that the conditions for the current state have not been met to promote the object to the next state. Sometimes a state may also have ignore signature defined, where a person can sign in place of the designated person.

This section includes code for both the signature approval and signature rejection.

### Promoting a Business Object

The following code segments demonstrate how to promote a business object. The first section of code demonstrates the signature approval, then the next code shows business object promotion.

#### Code sample - Signature Approval

```
try
{
   getContext().connect();
   _object.open(getContext());
   _object.approveSignature(getContext(), _signature, comment);
   _object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
{
}
```

#### Code Sample - Business Object Promotion

```
try
{
   getContext().connect();
   _object.open(getContext());
   _object.promote(getContext());
   _object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
```

```
{
   e.addMessage(StringResource.format(
                         MatrixErrorStringResource.getString(
                            MatrixErrorStringResource.CantPromote,
                              getContext().getLocale()),
                         "type",_object.getTypeName(),
                         "name",_object.getName(),
                         "revision",_object.getRevision()));
   error(e);
}
```

## Signature Rejection

If conditions, as defined in the policy, have not been met to move the business object to the next state, the person responsible for signing off on that particular state can select the Reject signature option. The following code segment demonstrates how to perform a signature rejection.

### Code sample - Signature Rejection

```
try
{
   getContext().connect();
   _object.open(getContext());
   _object.rejectSignature(getContext(), _signature, comment);
   _object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
{
}
```

## State Branching

A business object moves from state to state within its life cycle, as defined in the policy which governs it. A *branch* defines what the next state will be after a signature is applied. For each state, it is possible to have more than one branch. Which branch is taken depends on which signature is satisfied. The following code sample demonstrates how to get a list of branches for a particular state.

### Code Sample - State Branching

```
try
{
   getContext().connect();
   _object.open(getContext());
   StateList stateList = _object.getStates(getContext());
   _object.close(getContext());
   StateItr itr = new StateItr(stateList);
   while (itr.next())
   {
      State state = itr.obj();
```

```
            StateBranchList branches = state.getBranches();
            StateBranchItr branchItr = new StateBranchItr(branches);
            while (branchItr.next())
            {
               StateBranch branch = branchItr.obj();
               System.out.println("state = " + state.getName() + "
      branch = " + branch.getName());
            }
         }
         getContext().disconnect();
      }
      catch (MatrixException e)
      {
      }
```

## File Management

### Check Out Files

When you check a file out of ENOVIA Live Collaboration, it copies the file from the ENOVIA Live Collaboration database so you can modify it and check it back in. If you check a file out of the database, you should lock the business object to prevent anyone else with access privileges from attempting to make changes while you have the file checked out. Be sure to unlock the business object after you have checked the file back in.

```
// business object locked
   _object.lock(_context);
// business object unlocked
   _object.unlock(_context);
```

The following is a section of code that shows file checkout.

#### Code sample - File CheckOut

```
try
{
   getContext().connect();
   getContext().createWorkspace();
   object.open(getContext());
   object.checkoutFile(getContext(), _lock, checkoutFormat,
fileName, _path);
   object.close(getContext());
   getContext().disconnect();
}
catch (MatrixException e)
{
   System.out.println("an error occurred");
}
```

### Check In Files

The following is a section of code that shows file checkin.

### Code sample - File CheckIn

```
try
{
   getContext().connect();
   _object.open(getContext());
   _object.checkinFile(getContext(),
         _unlock,         // true to unlock
         _append,         // true to append files
         "",              // host
         _selectedFormat, // format
         _file,
         _path);
   _object.close(getContext());
}
catch (MatrixException e)
{
   throw e;
}
catch (SecurityException e)
{
   throw (new MatrixException(e.toString());
}
```

## Using the File Collaboration Server in a Custom Application

When ENOVIA Live Collaboration captured files need to be accessed over the internet, an alternative computer can be configured with a secondary file collaboration server that is used solely for file handling operations for the users at one site. This is particularly useful in a network where heavy traffic hinders file access with the primary collaboration server.

New constructors are available in CheckinDialog and CheckoutDialog that allow you to specify a context that represents the FileHost and FileServer of the secondary file server in a replicated network. These constructors are not strictly for checkin and checkout, but can also be used in routines that will handle open for view functionality. These new constructors were first available in version 9.0.2, and included in the on-line programmer's reference in the client-side Studio Customization Toolkit.

If you are writing an applet derived from MxApplet, there is no need to use the new constructors. The original constructors can be used since MxApplet automatically gets the context of the file server.

The following is the new constructor in the CheckinDialog:

```
public CheckinDialog(Context context,
         Context fileContext,
         DialogClient dialogClient,
         BusinessObject object)
     throws MatrixException, IllegalArgumentException
```

The following is the new constructor in the CheckoutDialog:

```
public CheckoutDialog(Context context,
         Context fileContext,
         DialogClient dialogClient,
         BusinessObjectList objects)
```

```
        throws MatrixException, IllegalArgumentException
```

*These constructors are not supported in CheckinServlet or CheckoutServlet, and so the file collaboration server cannot be used by JSP applications.*

Studio Customization Toolkit checkin/checkout can detect what context is being executed. When using the Studio Customization Toolkit *without* AppletXML (running in RMI), FCS is normally not used. You can make the Studio Customization Toolkit use FCS for checkin/checkout without AppletXML by setting the environment variable `MX_FCS_ENABLED` to `TRUE`. When set to FALSE, FCS is not used. When running *with* AppletXML, the Studio Customization Toolkit will use FCS.

For additional details, see the File Collaboration Server section in the Replicating Captured Stores chapter of the *System Manager Guide*.

### Disabling FCS for Any Studio Customization Toolkit Program

To disable FCS for any Studio Customization Toolkit program, do the following:

1. Create the context.
2. Include the following code:

```
Properties properties = context.getProperties();
if (properties != null)
  properties.setProperty("ematrix.fcsEnabled", "false");
```

## Appending an Object to the End of a Revision Chain

You can add an object to the end of an existing revision sequence, as long as the object is not already a member of another revision sequence. Attempts to add an object in the middle of an existing revision chain, or to add an object that is already part of a revision sequence will cause the Studio Customization Toolkit method to throw a MatrixException.

Creating new revisions in ENOVIA Live Collaboration has several side affects:

- **Float/Replicate rules.** Ordinarily the float/replicate rules for relationships cause relationships to be moved/copied to new revisions. These rules are ignored when using these interfaces; no relationships are added or removed to/from the inserted object, nor are any relationships added or removed to/from any of the previously existing members of the target sequence.
- **File Inheritance.** Ordinarily, a new revision of an existing object inherits all files checked into the original object. When using these interfaces, if the appended object already has checked in files, it does not inherit any files from the revision sequence. If the appended object has no files checked in, the behavior is controlled by the file keyword in MQL or the inheritFiles argument in Studio Customization Toolkit (if true, inherits files from previous revision; if false, no inheritance).
- **Triggers.** No triggers will fire.
- **History.** Revision history records will be recorded on both the new object and its previous revision (that is, both objects that are specified in the MQL command or Studio Customization Toolkit method).

Both of the following interfaces (methods on "this" business object) will insert newbus as the revision following "this" in the latter's revision sequence:

```
       public BusinessObject revise(Context context,
BusinessObject newbus,
String newvault)
            throws MatrixException
    {
        return (revise(context, (Visuals) null, newbus,
newvault));
    }



    public BusinessObject revise(Context context,
Visuals visuals,
BusinessObject newbus,
String newvault)
            throws MatrixException
```

### Example Code

```
    /**
     * Insert a revision of this BusinessObject
     *
     * @param context the context for this request
     * @param newbus the object to insert as a revision of this
     * @param newvault the vault to assign the revised
BusinessObject
     * @return a revision of this BusinessObject
     * @exception MatrixException when unable to retrieve data
from the collaboration server
     * @since ADK 6.2
     */
    public BusinessObject revise(Context context,
BusinessObject newbus,
String newvault)
            throws MatrixException
    {
        return (revise(context, (Visuals) null, newbus,
newvault));
    }

    /**
     * Insert a revision of this BusinessObject
     *
     * @param context the context for this request
     * @param visuals visuals to apply to this request
     * @param newbus the object to insert as a revision of this
     * @param newvault the vault to assign the revised
BusinessObject
     * @return a revision of this BusinessObject
     * @exception MatrixException when unable to retrieve data
from the collaboration server
     * @since ADK 7.0
     */
    public BusinessObject revise(Context context,
Visuals visuals,                           BusinessObject
newbus,
String newvault)
           throws MatrixException
```

# 6 Servlets

## ENOVIA Live Collaboration Servlet Summary

The ENOVIA Live Collaboration servlets are installed as part of the ENOVIA Live Collaboration Server (eMatrixServletXXX) and directly call the Studio Customization Toolkit, as is the case with the other parts of the server. The servlets are derived from the HttpServlet class in the standard javax.servlet package provided in the Java SDK (Servlet Development Kit).

The ENOVIA Live Collaboration servlets are summarized below. Some servlets require arguments and some have optional parameters. For a list of arguments and parameters and more information about how a servlet works, click on the servlet name in the left column.

*The ENOVIA Live Collaboration JavaDocs list more servlets than are listed below. The servlets that are not listed here are either internal servlets that should not be called directly or are in progress and not ready to be used.*

| ENOVIA Live Collaboration Servlet | Description | Class Name |
|---|---|---|
| BusinessObjectServlet | Loads business object ImageIcon gifs. | com.matrixone.servlet.BusinessObjectServlet |
| BusinessTypeServlet | Loads type gifs. | com.matrixone.servlet.BusinessTypeServlet |
| FileCheckinServlet | Checks in files. | com.matrixone.servlet.FileCheckinServlet |
| FileCheckoutServlet | Checks out files. | com.matrixone.servlet.FileCheckoutServlet |
| Framework | Initializes the properties for the application including server context, host name, user id, and password. | com.matrixone.servlet.Framework |
| FrameworkServlet | | com.matrixone.servlet.FrameworkServlet |
| LoginServlet | Authenticates users and connects to the ENOVIA Live Collaboration database. | com.matrixone.servlet.LoginServlet |
| LogoutServlet | Disconnects and end the user session so that subsequent Web pages are not connected and authenticated. | com.matrixone.servlet.LogoutServet |
| MatrixXMLServlet | Used with the Matrix Web Navigator applet. The XMLServlet is used with theLive Collaboration server. | com.matrixone.servlet.MatrixXMLServlet |
| ProgramServlet | Runs program objects. | com.matrixone.servlet.ProgramServlet |
| ResourceServlet | Delivers resource content (.gif, .wav, etc.) from Resource objects in your database or files in the server directory specified in web.xml. | com.matrixone.servlet.ResourceServlet |
| WizardServlet | Delivers wizard frames from the database, providing action buttons as appropriate. | com.matrixone.servlet.WizardServlet |

## Using Servlets

### Registering Servlets

In order to use a servlet, it must be registered with the J2EE application. If your implementation doesn't use a particular servlet, you need not register it. On the other hand, it does no harm to register servlets that are not used. The J2EE installation automatically registers the servlets in the web.xml file as shown below.

```
<!-- Define our servlets -->
<servlet id="Servlet_1">
         <servlet-name>Framework</servlet-name>
         <servlet-class>com.matrixone.servlet.Framework</servlet-class>
         <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet id="Servlet_2">
         <servlet-name>LogoutServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.LogoutServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_3">
         <servlet-name>ProgramServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.ProgramServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_4">
         <servlet-name>MatrixXMLServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.MatrixXMLServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_5">
         <servlet-name>ResourceServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.ResourceServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_6">
         <servlet-name>BusinessObjectServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.BusinessObjectServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_7">
         <servlet-name>MatrixExchangeServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.MatrixExchangeServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_8">
         <servlet-name>BusinessTypeServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.BusinessTypeServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_9">
         <servlet-name>FileCheckinServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.FileCheckinServlet</servlet-class>
      </servlet>
      <servlet id="Servlet_10">
         <servlet-name>LoginServlet</servlet-name>
         <servlet-class>com.matrixone.servlet.LoginServlet</servlet-class>
```

```
</servlet>
<servlet id="Servlet_11">
   <servlet-name>FileCheckoutServlet</servlet-name>
   <servlet-class>com.matrixone.servlet.FileCheckoutServlet</servlet-class>
</servlet>
<servlet id="Servlet_12">
   <servlet-name>WizardServlet</servlet-name>
   <servlet-class>com.matrixone.servlet.WizardServlet</servlet-class>
</servlet>
<servlet id="Servlet_13">
   <servlet-name>FrameworkServlet</servlet-name>
   <servlet-class>com.matrixone.servlet.FrameworkServlet</servlet-class>
</servlet>
<servlet id="Servlet_14">
   <servlet-name>MatrixAppletServlet</servlet-name>
   <servlet-class>com.matrixone.servlet.MatrixAppletServlet</servlet-class>
</servlet>
<servlet id="Servlet_15">
   <servlet-name>WorkspaceServlet</servlet-name>
   <servlet-class>com.matrixone.servlet.WorkspaceServlet</servlet-class>
</servlet>
<servlet id="Servlet_16">
   <servlet-name>fcs</servlet-name>
   <servlet-class>com.matrixone.fcs.fcs.FcsServlet</servlet-class>
</servlet>
<servlet id="Servlet_17">
   <servlet-name>audit</servlet-name>
   <servlet-class>com.matrixone.servlet.AuditServlet</servlet-class>
   <load-on-startup>2</load-on-startup>
</servlet>
```

## Servlet Arguments and Parameters

Some servlets accept arguments and parameters that are used to evaluate the servlet code. An argument is any required entry, and the order these are passed is important, since no argument name is passed with it. For example, many of the servlets expect to be passed a business object’s type, name, and revision, in that order. If one of the fields will not be provided, a wildcard character must be passed. Parameters, on the other hand, are generally optional, and are specified as “name=value”. The order that parameters are listed is irrelevant. These arguments and parameters are embedded in the URL used to invoke a servlet. The arguments and parameters for servlets are described in *Servlet Details*.

## Getting the URL for a Servlet

The servlets that are called from other servlets and JSPs have a `getURL` method that gets the URL, as defined in the alias, to invoke the servlet. Use the getURL method anywhere a URL is expected.

For example, here’s an example of the getURL method being used to invoke the LoginServlet as the action for a login form.

```
<form name="loginForm" method="post"
action="<%=LoginServlet.getURL()%>">
```

The getURL method for some servlets have parameters. For information about the parameters required for each getURL method, see the Javadoc for the servlets.

# Servlet Details

This section gives details about the parameters, arguments, and typical URLs for the ENOVIA Live Collaboration servlets.

## BusinessObjectServlet

This servlet gets the ImageIcon for a business object.

| | | |
|---|---|---|
| **Arguments** | TYPE | Business object type name. |
| | NAME | Business object name. |
| | REVISION | Business object revision. |
| | VAULT | Business object vault name. |
| **Parameters** | action | An action equal to *image* returns the ImageIcon for the given object as a gif. If the object has no ImageIcon, the type icon is returned instead. |
| | *Even though there is only one valid action value, this parameter is required.* | |

Assuming the alias for the servlet is servlet/object (as defined in the web.xml file), the URL to access this servlet would be:

```
/servlet/object/TYPE/NAME/REVISION/VAULT?action=image
```

## BusinessTypeServlet

The BusinessTypeServlet retrieves the type icon for a business object.

| | | |
|---|---|---|
| **Arguments** | TYPE | Business object type name. |
| **Parameters** | none | |

Assuming the alias for the servlet is servlet/type (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/type/TYPE
```

If the TYPE specified is not valid then the result will be a 404 status code, not found.

## FileCheckinServlet

The FileCheckinServlet lets users check in files to business objects.

| | | |
|---|---|---|
| **Arguments** | BUSID | The ID of the business object, such as 11748.18088.10698.61833.<br>Instead of the business object ID, you can use the TYPE, NAME, REVISION, and VAULT but the ID is preferable. |
| **Parameters** | format | Format name. The default format specified in policy is used if this parameter is not included. |
| | unlock | Whether to unlock the file to allow others to lock it for editing and check in new revisions. To unlock the file, set the parameter to true. The default is false. |
| | append | Whether the file should be appended to other files of the same format that are already checked in (true) or the file should replace any existing files of that format (false). The default is false. |

Assuming the alias for the servlet is servlet/checkin (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/checkin/
BUSID&format=format_name&append=false&unlock=true
```

The FileCheckinServlet uses an HTML form to checkin the file. It also uses a utility servlet called MultipartRequest. You might use the following HTML code to checkin a file:

```
<FORM ACTION="/ematrix/servlet/checkin/
11748.18088.10698.61833?format=ASCII&lock=true&append=true"ENCT
YPE="Multipart/form-data" METHOD=POST> file to checkin <INPUT
TYPE=FILE NAME=file><BR>
<INPUT TYPE=SUBMIT></FORM>
```

Files that are checked in using the servlet cannot include the following special characters in their names:

| |
|---|
| **; \ / ? < > + = : & @ $ , * \| "** |

*The Macintosh operating system (through OS 9.1) supports filenames up to, but not more than, 31 characters in length and uses colons as path separators.*

## FileCheckoutServlet

The FileCheckoutServlet lets users check out a file. It accepts the following arguments and parameters.

| | | |
|---|---|---|
| **Arguments** | BUSID | The ID of the business object, such as 11748.18088.10698.61833.<br>Instead of the business object ID, you can use the TYPE, NAME, REVISION, and VAULT but the ID is preferable. |
| | FILE_NAME | The name of the file that is being checked out. Note that the file name is needed in both the main URL and as a parameter because some browsers looks for the file name in the URL. |
| **Parameters** | format | Format name. If not specified, the default format or format that contains a file is used. |
| | file | The name of the file to be checked out. If not specified, an error is returned. |
| | lock | Whether to lock the file for edit or not. To lock the file, preventing anyone else from locking the file or checking in a new revision until it is unlocked, set to true. The default is false. |

Assuming the alias for the servlet is servlet/checkout (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/checkout/BUSID/file_name?file=file_name&format=form
at_name&lock=true
```

For example:

```
/servlet/checkout/11748.18088.10698.61833/
Document.doc?format=generic&lock=false&file=Document.doc"
```

If the file gets checked out successfully, the Web browser displays the file or prompts the user to save the file. Otherwise, the return status is 404 not found, with the text of the exception thrown.

## LoginServlet

The LoginServlet authenticates users and connects to the ENOVIA Live Collaboration database. The servlet logs in users by creating a matrix.db.Context object that is stored in the http session for the user. The servlet first checks to see if the user has already been authenticated by the Web/application server (for example, through LDAP). If pre-authentication has occurred, the servlet uses its username and password. Otherwise, a login page is presented to capture the username and password. The servlet gets the parameters required for creating a context object—server and host—from the web.xml file in place when the application is deployed. Once a context object is stored in the session, the user is considered logged in.

The servlet uses an HTML form to accept user input for login name and password, which prevents having to include the username and password in the URL as arguments and parameters. The HTML form must pass the username and password to the servlet using two fields:

`LoginServlet.FORM_LOGIN_NAME`

`LoginServlet.FORM_LOGIN_PASSWORD`

The user can log out explicitly (through the LogoutServlet) or may be logged out after some period of inactivity.

A sample login JSP that uses the LoginServlet is described in Chapter 4, *Creating a Login Page*.

## LogoutServlet

The LogoutServlet disconnects the current session by dropping the current Context object and invalidating the user session. To access the applications and database again, the user will be forced to login. The user may also be logged out by the Web server. (Most Web servers disconnect a user after some configurable amount of time.) No input arguments or parameters are required.

## ProgramServlet

Executes programs.

| | | | |
|---|---|---|---|
| **Arguments** | NAME | Program name. | |
| **Parameters** | action | A value of: | will: |
| | | *icon* (default) | return the icon for the given NAME as a gif. |
| | | *execute* | run program NAME. |

Assuming the alias for the servlet is servlet/program (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/program/NAME?action=execute
```

If no NAME is specified and the `ematrix.servlet.ui` property is set to true (in the web.xml file) then the user will be presented with a list of links to program objects. These links will only include those programs that are not methods, not wizards and to which the user has execute access. Clicking a link will execute the program. If the `ematrix.servlet.ui` property is set to false (its default value), the result will be a 501 status code, not implemented.

When the action is execute, if the program executes successfully, a status of 201, no content will be returned. Otherwise the return status will be 202, accepted with the text of the exception thrown.

## ResourceServlet

Delivers resource files (for example, GIF, WAV, or AVI files) by copying the files from your database or by retrieving files directly from the server directory specified in web.xml.

| | | |
|---|---|---|
| **Arguments** | NAME | Resource file name. |
| **Parameters** | none | |

Assuming the alias for the servlet is servlet/resource (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/resource/NAME
```

## WizardServlet

The Wizard servlet delivers wizard frames from the database, providing action buttons as appropriate. The web.xml file contains three settings that control the behavior of the servlet.

### Arguments and Parameters

The Wizard servlet accepts several arguments and parameters

| | | |
|---|---|---|
| **Arguments** | NAME | Name of the wizard object in the ENOVIA Live Collaboration database. |
| **Parameters** | action | Enter the value *execute* to run the specified wizard. If you don't specify the action parameter, the servlet supplies it automatically. |
| | oid | The object ID for a business object that you want to pass to the wizard. |
| | icon | Displays the icon for the given NAME as a gif. |

Assuming the alias for the servlet is servlet/wizard (as defined in the web.xml file), the URL to access the servlet would be:

```
/servlet/wizard/NAME?action=execute
```

### Considerations for Existing Wizards

The Wizard servlet has several limitations, most of which are a result of running wizards in an HTML environment. Make sure you test existing wizards and be prepared to redesign them as needed to work around these limitations.

- The width of buttons on wizards varies in some cases when they should be exactly the same.
- The Wizard servlet currently does not support `application` commands. Application commands call the dialog boxes, such as the Navigator browser or Checkin dialog box. The servlet does support MQL `notice`, `error`, and `warning` commands. To have the servlet display a message to warn you when a wizard uses an unsupported command, set the ematrix.frame.warn.unsupported property to true in the web.xml file.
- The width of some widgets (combo boxes, list boxes, and buttons) is too small or too large, depending on the width of the string. The browser controls the width of these objects.
- Combo boxes cannot be typed into to add to or edit the selections.
- The font sizes and background colors for widgets may not appear exactly as defined.

Several additional limitations associated with the Wizard servlet occur only when running with some browsers:

- The frame around some sets of controls, such as radio buttons and checkboxes, is not visible. This occurs when controls are displayed using an HTML table control.
- Text does not wrap when multiple lines of text are entered in text fields.
- If a frame contains many controls, some controls may not display at first. To make them display, resize the browser window.
- When a list box has a default item specified, the item is not scrolled into view, so it does not look selected.
- Read-only multiline fields can be typed into.

# 7 ODBC Reporting

## ODBC Driver

The ENOVIA Live Collaboration ODBC (Open Database Connectivity) driver can be used to access the ENOVIA Live Collaboration database from any ODBC-compliant application. Most often it will be used for formatting and reporting of data in applications such as Crystal Reports or Microsoft Office, but it can also be used to perform write operations on the database, using Visual Basic programs, for example.

Access to ENOVIA Live Collaboration through the ODBC driver is fully compliant with all ENOVIA Live Collaboration security mechanisms as well as distributed/ replicated/ federated ENOVIA Live Collaboration environments since the driver is really a special kind of ENOVIA Live Collaboration client application that works through ENOVIA Live Collaboration, rather than performing direct ODBC access to the backend stored ENOVIA Live Collaboration data.

### Installation and Access

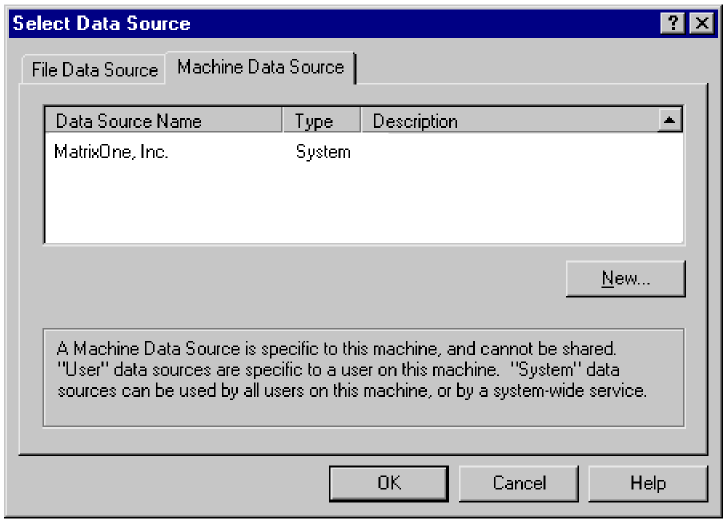
The ENOVIA Live Collaboration ODBC driver, mxodbc.dll, is automatically installed in the ..\bin\winnt directory as a standard part of the ENOVIA Live Collaboration installation process. The Windows registry is updated to include the new ENOVIA Live Collaboration data source, which can then be seen via the ODBC Administrative tool in the Windows Control Panel.

ODBC requires an ENOVIA Live Collaboration /Oracle database. (DB2 is not currently supported.) Refer to the Prerequisites chapter of the *ENOVIA Live Collaboration Installation Guide* for hardware and software version requirements.

*Currently, installation of the ODBC driver in a LAN based server configuration is not supported.*

*The ODBC driver uses the Matrix-r bootstrap file in the ENOVIA_INSTALL/studio directory to locate the ENOVIA Live Collaboration database. You cannot select an alternate bootstrap file.*

When you set up a data source using the ENOVIA Live Collaboration ODBC driver, you are asked for the Data Source name (DSN Name). Select the Machine Data Source tab and select **ENOVIA** For example:



| Data Source Name | Type | Description |
|---|---|---|
| MatrixOne, Inc. | System | |

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

You will also be asked for a username and password. Use the same username and password that you normally use when setting context in ENOVIA Live Collaboration. This ensures the same levels of security that are available when accessing ENOVIA Live Collaboration directly.

*The user interface for selecting the Data Source and context may vary from application to application.*

For a detailed example of using ODBC to extract data from the ENOVIA Live Collaboration database, see *Importing Data into MS Access Example*.

## Table Interfaces

The following sections describe the table interfaces exposed by the ENOVIA Live Collaboration ODBC interface. With these table definitions, any application using an ODBC connection can have full read/write access to the ENOVIA Live Collaboration database. Access through the ODBC driver is fully compliant with all ENOVIA Live Collaboration security mechanisms as well as distributed/ replicated/federated ENOVIA Live Collaboration environments.

The most important thing to realize with the ODBC interface is that the tables being manipulated by the ODBC application don't actually exist anywhere. The tables are views of data that are constructed internally by ENOVIA Live Collaboration at runtime from ENOVIA Live Collaboration objects.

The ODBC interface should be able to optimize queries to use any indices available in the ENOVIA Live Collaboration schema. For instance, the SQL statement

```
select * from PERSON where NAME='bob'
```

should recognize that an index is available on the NAME column of the PERSON table and act accordingly. The interface tables have been structured to allow most ENOVIA Live Collaboration queries to be performed in a single `select` statement.

There are four kinds of tables that have been defined within the ENOVIA Live Collaboration ODBC structure:

1. Administration data—definitions of object structure
2. Instance data—objects defined using structure contained in Administration data
3. Workspace data - objects defining preferences or settings owned by a single user
4. Utility tables - other tables providing direct APIs into ENOVIA Live Collaboration's object model

The definition of these tables is hard-wired in the ODBC interface implementation. There is no need for an application to run DDL (Data Description Language) statements (create table, create index, etc.).

## Instance Tables

Instance tables contain objects defined using structure contained in Administration data tables.

*The Created, Modified, and History date/time attributes for business objects cannot be edited.*

| Table Name | Table Columns |
|---|---|
| BUSINESS_OBJECT | OID char(64); |
| | VAULT char(128); |
| | TYPE char(128); |
| | NAME char(128); |
| | REVISION char(128); |
| | POLICY char(128); |
| | STATE char(128); |
| | OWNER char(128); |
| | LOCKER char(128); |
| | CREATED date; |
| | MODIFIED date; |
| | PREVIOUS_REVISION char(64); |
| | NEXT_REVISION char(64); |
| BUSINESS_OBJECT_LONG STRING_ATTRIBUTE | OID char(64); |
| | ATTRIBUTE_NAME char(128); |
| | VALUE long; |
| BUSINESS_OBJECT_ATTRIBUTE_${TYPE} (one table per business type) | OID char(64); |
| | ATTRIBUTE1 type1; |
| | ATTRIBUTE2 type2; |

| Table Name | Table Columns |
|---|---|
| BUSINESS_OBJECT_STATE | OID char(64); |
| | STATE char(128); |
| | ENABLE boolean; |
| | DISABLE boolean; |
| | OVERRIDE boolean; |
| | SIGNATURE char(128); |
| | SIGNER char(128); |
| | APPROVE boolean; |
| | REJECT boolean; |
| | IGNORE boolean; |
| | COMMENT long; |
| RELATIONSHIP | OID char(64); |
| | TYPE char(128); |
| | FROM char(64); |
| | TO char(64); |
| RELATIONSHIP_LONG_STRING_ATTRIBUT E | OID char(64); |
| | ATTRIBUTE_NAME char(128); |
| | VALUE long; |
| RELATIONSHIP_ATTRIBUTE_${TYPE} (one table per relationship type) | OID char(64); |
| | ATTRIBUTE1 type1; |
| | ATTRIBUTE2 type2; |
| HISTORY | OID char(64); |
| | DATE date; |
| | KIND int; |
| | HISTORY long; |

## Administration Tables

Administration tables contain definitions of object structure.

| Table Name | Table Columns |
|---|---|
| VAULT | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | TABLE_SPACE(128); |
| | INDEX_SPACE(128); |
| | SERVER(128); |
| STORE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | TYPE int; |
| | VERSION int; |
| | TABLE_SPACE(128); |
| | INDEX_SPACE(128); |
| | HOST char(128); |
| | PATH char(128); |
| | PERMISSION int; |
| | FTP_USER(128); |
| | FTP_PASS(128); |
| PERSON | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | FULLNAME char(255); |
| | ADDRESS char(255); |
| | PHONE char(40); |
| | FAX char(40); |
| | EMAIL char(128); |
| | PASSWORD char(128); |
| | HOME_VAULT char(128); |
| | HOME_SITE char(128); |
| | ACCESS_MASK int; |

| Table Name | Table Columns |
|---|---|
| GROUP | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | PARENT char(128); |
| ROLE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | PARENT char(128); |
| ASSIGNMENT | PERSON char(128); |
| | GROUP char(128); |
| | ROLE char(128); |
| ATTRIBUTE_TYPE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | DATA_TYPE int; |
| | DEFAULT long; |
| RANGE | ATTRIBUTE char(128); |
| | TYPE int; |
| | LOW_INCLUSIVE boolean; |
| | LOW_VALUE long; |
| | HIGH_INCLUSIVE boolean; |
| | HIGH_VALUE long; |
| FORMAT | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | VERSION char(128); |
| | SUFFIX char(128); |
| PROGRAM | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | NEEDS_CONTEXT boolean; |
| | IS_MQL boolean; |
| | CODE long; |

| Table Name | Table Columns |
|---|---|
| BUSINESS_TYPE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | PARENT char(128); |
| BUSINESS_TYPE_INFO | TYPE char(128); |
| | KIND int; |
| | OBJECT char(128); |
| RELATIONSHIP_TYPE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | PARENT char(128); |
| | ALL_TYPE_FROM boolean; |
| | ALL_TYPE_TO boolean; |
| | MEANING_FROM char(128); |
| | MEANING_TO char(128); |
| RELATIONSHIP_TYPE_INFO | TYPE char(128); |
| | KIND int; |
| | OBJECT char(128); |
| POLICY | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | ALL_TYPES boolean; |
| | ALL_FORMATS boolean; |
| | SEQUENCE long; |
| | DEFAULT_FORMAT char(128); |
| | STORE char(128); |
| POLICY_INFO | POLICY char(128); |
| | KIND int; |
| | OBJECT char(128); |

| Table Name | Table Columns |
|---|---|
| STATE_REQUIREMENT | POLICY char(128); |
| | STATE char(128); |
| | STATE_DESCRIPTION long; |
| | STATE_ICON raw; |
| | PUBLIC_ACCESS int; |
| | OWNER_ACCESS int; |
| STATE_USER_ACCESS; | POLICY char(128); |
| | STATE char(128); |
| | USER char(128); |
| | USER_ACCESS int; |
| POLICY_STATE_SIGNATURE | POLICY char(128); |
| | STATE char(128); |
| | SIGNATURE char(128); |
| | SIGNATURE_TYPE int; |
| | SIGNATURE_BRANCH char(128); |
| | SIGNATURE_FILTER long; |
| REPORT | NAME char(128); |
| | ROW_SIZE double; |
| | COLUMN_SIZE double; |
| | HEADER_SIZE double; |
| | FOOTER_SIZE double; |
| | LEFT_MARGIN double; |
| | RIGHT_MARGIN double; |
| | UNITS int; |

| Table Name | Table Columns |
|---|---|
| REPORT_FIELD | REPORT char(128); |
| | FIELD_NUMBER int; |
| | FIELD_TYPE int; |
| | FIELD_CALC_TYPE int; |
| | VALUE long; |
| | X_FONT double; |
| | Y_FONT double; |
| | FONT_NAME char(128); |
| | X double; |
| | Y double; |
| | XABS boolean; |
| | YABS boolean; |
| | HEIGHT double; |
| | WIDTH double; |
| | OUTPUT int; |
| | LABEL char(128); |
| | X_LABEL double; |
| | Y_LABEL double; |
| | XABS_LABEL boolean; |
| | YABS_LABEL boolean; |
| | HEIGHT_LABEL double; |
| | WIDTH_LABEL double; |
| | OUTPUT_LABEL int; |
| | FILTER long; |
| SERVER | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | USER char(128); |
| | PASSWORD char(128); |
| | CONNECT char(128); |

| Table Name | Table Columns |
|---|---|
| FORM | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | FOREGROUND char(128); |
| | BACKGROUND char(128); |
| | ROW_SIZE double; |
| | COLUMN_SIZE double; |
| | HEADER_SIZE double; |
| | FOOTER_SIZE double; |
| | LEFT_MARGIN double; |
| | RIGHT_MARGIN double; |
| | UNITS int; |
| FORM_FIELD | FORM char(128); |
| | FIELD_NUMBER int; |
| | FIELD_TYPE int; |
| | VALUE long; |
| | LABEL long; |
| | FOREGROUND char(128); |
| | BACKGROUND char(128); |
| | X double; |
| | Y double; |
| | HEIGHT double; |
| | WIDTH double; |
| | MIN_HEIGHT double; |
| | MIN_WIDTH double; |
| | X_FONT double; |
| | Y_FONT double; |
| | FONT_NAME char(128); |
| SITE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | LOCATION location(128); |

| Table Name | Table Columns |
|---|---|
| LOCATION | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | HOST char(128); |
| | PATH char(128); |
| | PERMISSION int; |
| | FTP_USER(128); |
| | FTP_PASS(128); |
| RELATIONSHIP_RULE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | ALL_TYPES boolean; |
| RELATIONSHIP_RULE_INFO | RULE char(128); |
| | KIND int; |
| | OBJECT char(128); |
| ACCESS_RULE | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | PUBLIC_ACCESS int; |
| | OWNER_ACCESS int; |
| ACCESS_RULE_USER_ACCESS | RULE char(128); |
| | USER char(128); |
| | USER_ACCESS int; |

## Workspace Tables

Workspace tables contain objects defining preferences or settings owned by a single user.

| Table Name | Table Columns |
|---|---|
| $TABLE: | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | FOREGROUND char(128); |
| | BACKGROUND char(128); |
| | ROW_SIZE double; |
| | COLUMN_SIZE double; |
| | HEADER_SIZE double; |
| | FOOTER_SIZE double; |
| | LEFT_MARGIN double; |
| | RIGHT_MARGIN double; |
| | UNITS int; |
| TABLE_FIELD | FORM char(128); |
| | FIELD_NUMBER int; |
| | FIELD_TYPE int; |
| | VALUE long; |
| | LABEL long; |
| | FOREGROUND char(128); |
| | BACKGROUND char(128); |
| | X double; |
| | Y double; |
| | HEIGHT double; |
| | WIDTH double; |
| | MIN_HEIGHT double; |
| | MIN_WIDTH double; |
| | X_FONT double; |
| | Y_FONT double; |
| | FONT_NAME char(128); |

| Table Name | Table Columns |
|---|---|
| BUSINESS_OBJECT_QUERY: | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | NAME_PATTERN char(255); |
| | TYPE_PATTERN char(255); |
| | REVISION_PATTERN char(255); |
| | OWNER_PATTERN char(255); |
| | LATTICE_PATTERN char(255); |
| | WHERE_CLAUSE long; |
| BUSINESS_OBJECT_SET: | NAME char(128); |
| | MEMBER char(64); |
| TOOL_SET: | NAME char(128); |
| | PROGRAM char(128); |
| $VIEW: | NAME char(128); |
| | KIND int; |
| | ACTIVE boolean; |
| | OBJECT char(128); |
| VISUAL_CUE: | NAME char(128); |
| | DESCRIPTION long; |
| | ICON raw; |
| | NAME_PATTERN char(255); |
| | TYPE_PATTERN char(255); |
| | REVISION_PATTERN char(255); |
| | OWNER_PATTERN char(255); |
| | LATTICE_PATTERN char(255); |
| | WHERE_CLAUSE long; |
| | FONT char(128); |
| | LINE_STYLE char(128); |
| | FOREGROUND char(128); |
| | BACKGROUND char(128); |

## Utility Tables

Utility tables contain other tables that provide direct APIs into ENOVIA Live Collaboration's object model

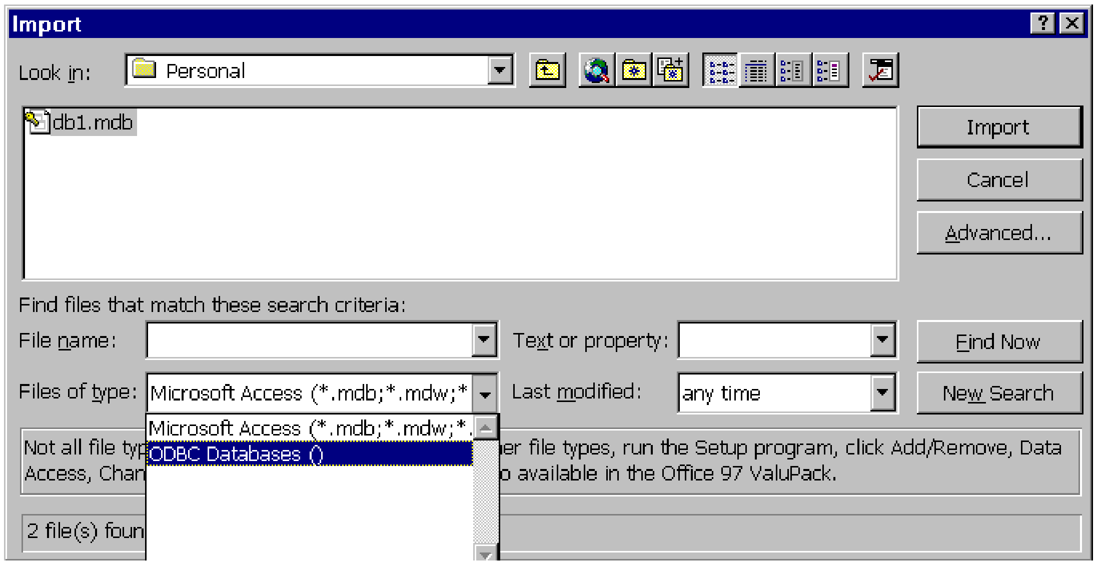
| Table Name | Table Columns |
|---|---|
| MQL_COMMAND<br>(interface to run arbitrary MQL commands) | INPUT long; |
| | OUTPUT long; |
| | ERROR long; |

# Examples

## Importing Data into MS Access Example

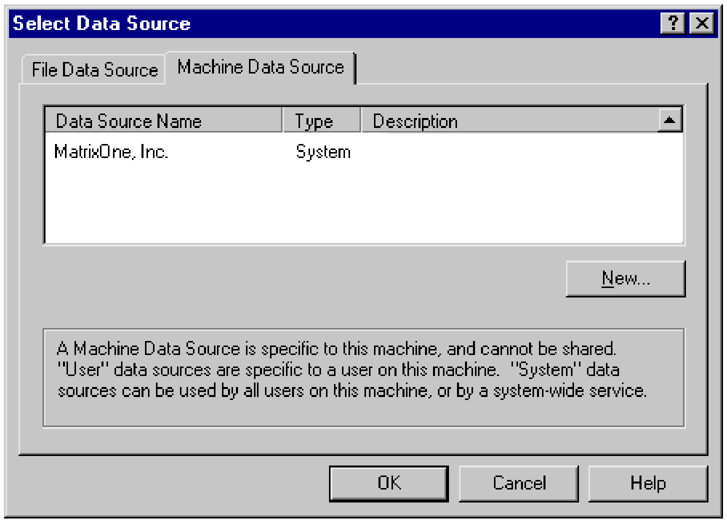
The following describes in detail how to import data from ENOVIA Live Collaboration into Microsoft Access using ODBC.

1. Start Microsoft Access.
2. Open a database or create a new one.
3. Select **File>Get External Data>Import**.
4. In the Import Dialog, from the **Files of type** list, select **ODBC Databases**.


Import
Look in:
Personal
db1.mdb
Import
Cancel
Advanced...
Find files that match these search criteria:
File name:
Text or property:
Find Now
Files of type:
Microsoft Access (*.mdb;*.mdw;*
Last modified:
any time
New Search
Microsoft Access (*.mdb;*.mdw;*.
ODBC Databases ()
Not all file typ
Access, Char
er file types, run the Setup program, click Add/Remove, Data
o available in the Office 97 ValuPack.
2 file(s) foun

The Select Data Source dialog is displayed.

5. In the Select Data Source dialog, select the Machine Data Source tab, and then select **ENOVIA**.

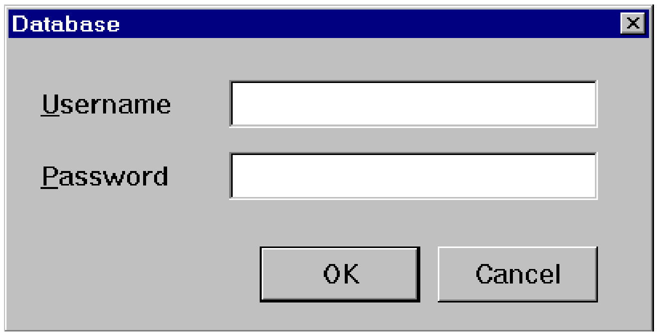


| Data Source Name | Type | Description |
|---|---|---|
| MatrixOne, Inc. | System | |

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

6. Click **OK**.

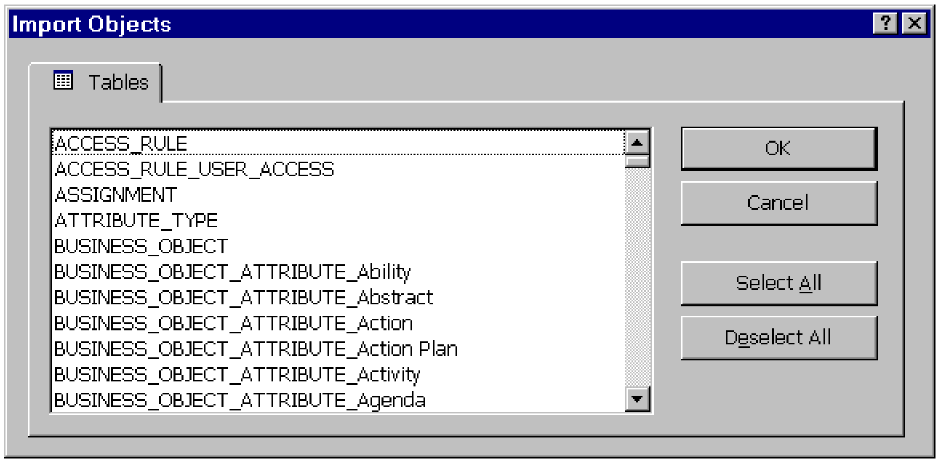
7. The Database dialog is displayed, where you can set the context for the session. Use the same username and password that you would use when setting context in ENOVIA Live Collaboration.



8. Click **OK**.

*If the username and password have been configured in the Windows ODBC setup (accessed from the Windows Control Panel), then the Database dialog will not be displayed.*

9. From the Import Objects dialog, select the table(s) that contain the data you want to import and click **OK**. (Use Ctrl-click to select multiple tables.)



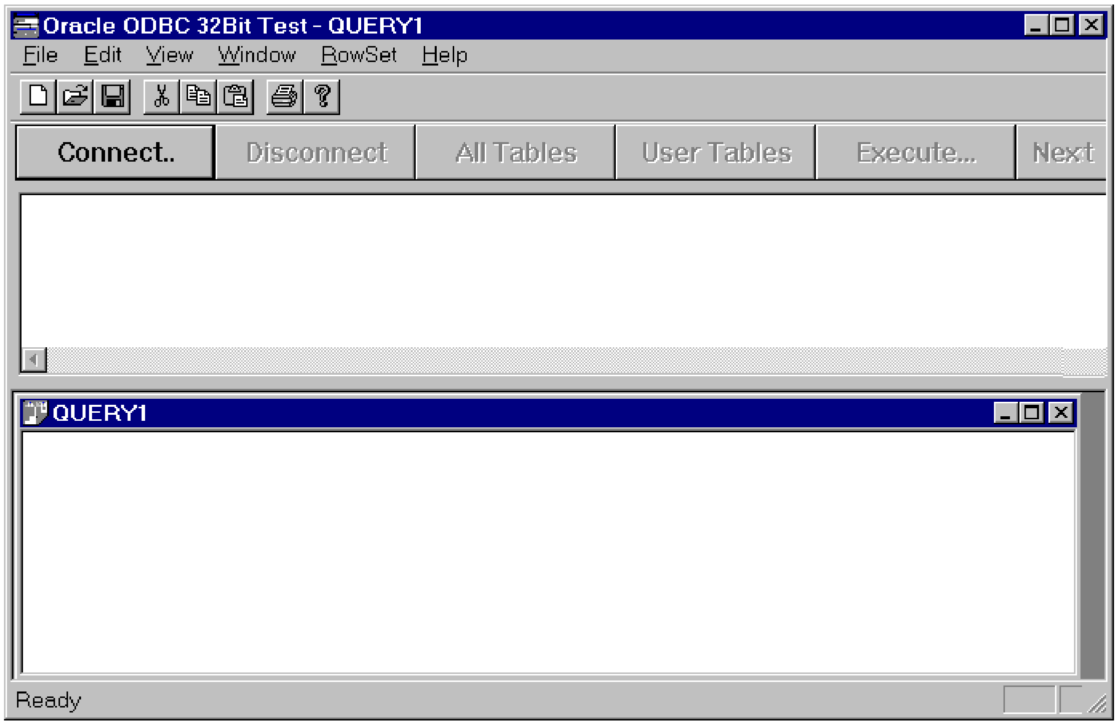
ACCESS_RULE
ACCESS_RULE_USER_ACCESS
ASSIGNMENT
ATTRIBUTE_TYPE
BUSINESS_OBJECT
BUSINESS_OBJECT_ATTRIBUTE_Ability
BUSINESS_OBJECT_ATTRIBUTE_Abstract
BUSINESS_OBJECT_ATTRIBUTE_Action
BUSINESS_OBJECT_ATTRIBUTE_Action Plan
BUSINESS_OBJECT_ATTRIBUTE_Activity
BUSINESS_OBJECT_ATTRIBUTE_Agenda

The information is imported into Microsoft Access.

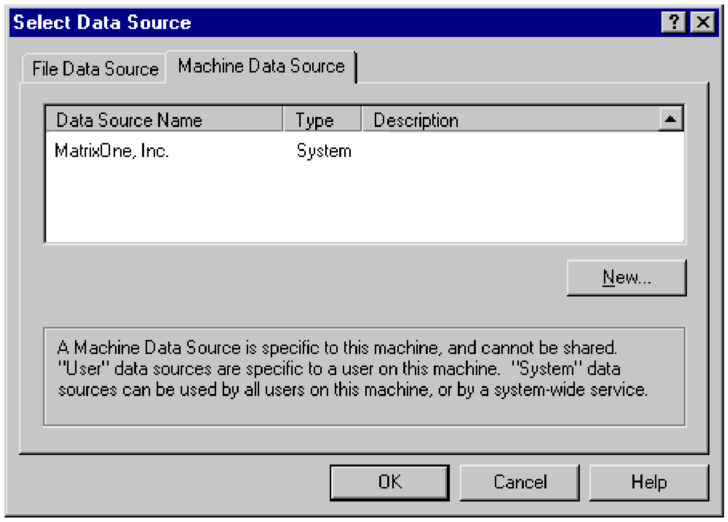
## Write Operation Example

The following is an Oracle “ODBC TEST” example of a write operation, to add a new object to the set BUSINESS_OBJECT:

1. Start ODBC TEST. The following window is displayed:



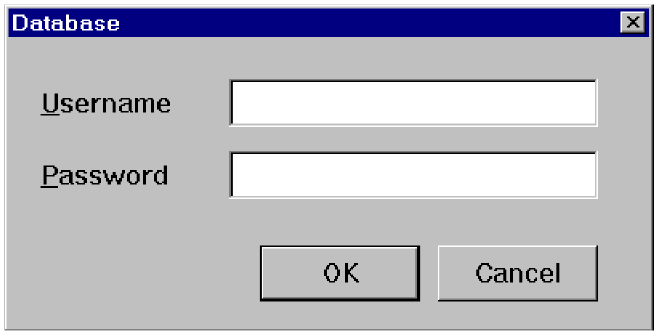
2. Click **Connect**.
3. In the Select Data Source dialog, select the Machine Data Source tab, and then click **ENOVIA.**



| Data Source Name | Type | Description |
|---|---|---|
| MatrixOne, Inc. | System | |

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

4. Click **OK**.
5. The Database dialog is displayed, where you can set the context for the session. Use the same username and password that you would use when setting context in ENOVIA Live Collaboration.



**6.** Click **OK**.

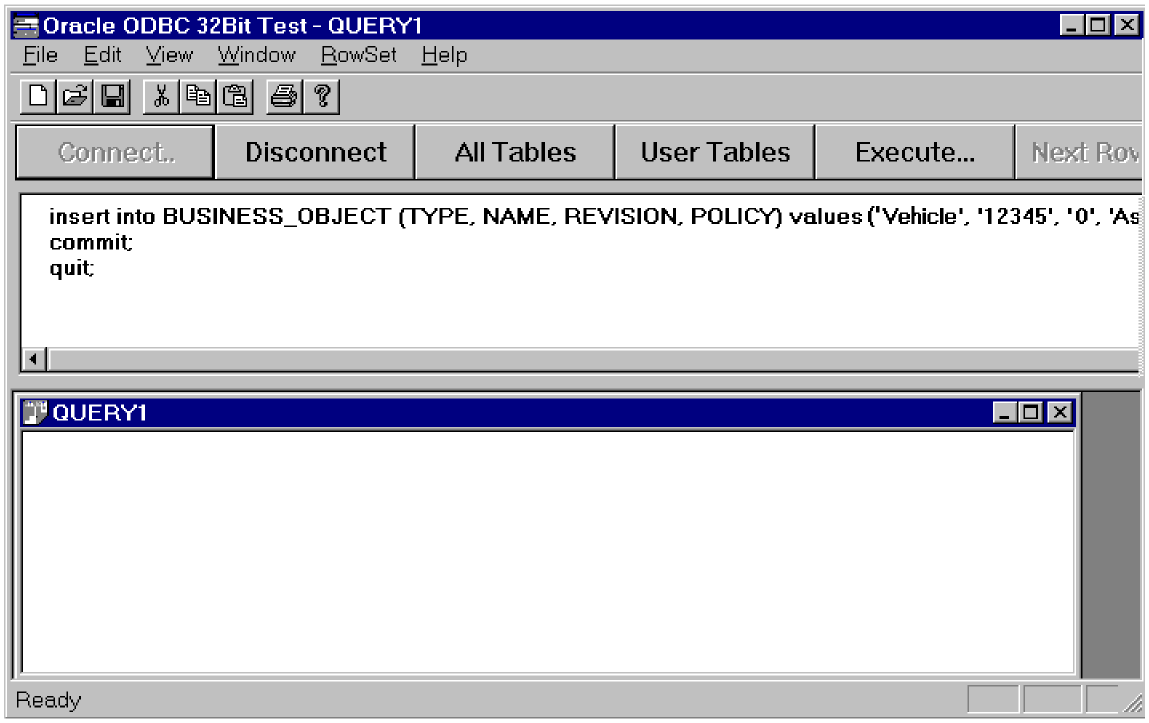
---

*If the username and password have been configured in the Windows ODBC setup (accessed from the Windows Control Panel), then the Database dialog will not be displayed.*

---

**7.** Add commands to the upper window. For example, to add a new object to the set BUSINESS_OBJECT, enter the following commands:

```
>insert into BUSINESS_OBJECT
(TYPE,NAME,REVISION,POLICY,VAULT) values
('Vehicle','12345','0','ASSEMBLY','CM');
>commit;
>quit;
```



```
insert into BUSINESS_OBJECT (TYPE, NAME, REVISION, POLICY) values ('Vehicle', '12345', '0', 'As
commit;
quit;
```

**8.** Click **Execute** to run the commands and modify the database.

---

*The Oracle test application doesn't deal well with multiple long columns in a table.*

---

## Supported API Calls

The ODBC conformance level of the driver is Level 1 API, minimum+ SQL grammar.

For information on SQL grammar, see *Supported SQL Grammar*.

The following table contains the API calls that are included in ENOVIA Live Collaboration ODBC support:

| Supported API Calls | | |
|---|---|---|
| SQLALLOCCONNECT | SQLALLOCENV | SQLALLOCSTMT |
| SQLBINDCOL | SQLCANCEL | SQLCOLATTRIBUTES |
| SQLCONNECT | SQLDESCRIBECOL | SQLDISCONNECT |
| SQLERROR | SQLEXECDIRECT | SQLEXECUTE |
| SQLFETCH | SQLFREECONNECT | SQLFREEENV |
| SQLFREESTMT | SQLGETCURSORNAME | SQLNUMRESULTCOLS |
| SQLPREPARE | SQLROWCOUNT | SQLSETCURSORNAME |
| SQLSETPARAM | SQLTRANSACT | SQLCOLUMNS |
| SQLDRIVERCONNECT | SQLGETCONNECTOPTION | SQLGETDATA |
| SQLGETFUNCTIONS | SQLGETINFO | SQLGETSTMTOPTION |
| SQLGETTYPEINFO | SQLPARAMDATA | SQLPUTDATA |
| SQLSETCONNECTOPTION | SQLSETSTMTOPTION | SQLSPECIALCOLUMNS |
| SQLSTATISTICS | SQLTABLES | SQLBROWSECONNECT |
| SQLCOLUMNPRIVILEGES | SQLDATASOURCES | SQLDESCRIBEPARAM |
| SQLEXTENDEDFETCH | SQLFOREIGNKEYS | SQLMORERESULTS |
| SQLNATIVESQL | SQLNUMPARAMS | SQLPARAMOPTIONS |
| SQLPRIMARYKEYS | SQLPROCEDURECOLUMNS | SQLPROCEDURES |
| SQLSETPOS | SQLSETSCROLLOPTIONS | SQLTABLEPRIVILEGES |
| SQLDRIVERS | SQLBINDPARAMETER | SQLCOLATTRIBUTE |

*SQL reserve words like GROUP may need to be double quoted in the SQL statement*

## Supported SQL Grammar

The ODBC conformance level of the driver is Level 1 API, minimum+ SQL grammar.

For information on which ODBC commands are supported, see *Supported API Calls*.

The following lists the SQL grammar supported by ENOVIA Live Collaboration  ODBC.

| SQL Grammar |
|---|
| `statement ::= CREATE create \| DROP drop \| SELECT select orderby \| INSERT insert \| DELETE delete \| UPDATE update \| passthroughSQL` |
| `passthroughSQL ::= any statement supported by the backend` |
| `create ::= TABLE tablename ( createcols ) \| INDEX indexname ON tablename ( indexcolumns )` |
| `indexcolumns ::= indexcolumn \| indexcolumn , indexcolumns` |
| `indexcolumn ::= columnname asc` |
| `createcols ::= createcol , createcols \| createcol` |
| `createcol ::= columnname datatype \| columnname datatype ( integer ) \| columnname datatype ( integer , integer )` |
| `drop ::= TABLE tablename \| INDEX indexname` |
| `select ::= selectcols FROM tablelist where groupby having` |
| `delete ::= FROM tablename where` |
| `insert ::= INTO tablename insertvals` |
| `update ::= tablename SET setlist where` |
| `setlist ::= set \| setlist , set` |
| `set ::= columnname = NULL \| columnname = expression` |
| `insertvals ::= ( columnlist ) VALUES ( valuelist ) \| VALUES ( valuelist ) \| ( columnlist ) VALUES ( SELECT select ) \| VALUES ( SELECT select )` |
| `columnlist ::= columnname , columnlist \| columnname` |
| `valuelist ::= NULL , valuelist \| expression , valuelist \| expression \| NULL` |
| `selectcols ::= selectallcols * \| selectallcols selectlist` |
| `selectallcols ::=  \| ALL \| DISTINCT` |
| `selectlist ::= selectlistitem , selectlist \| selectlistitem` |
| `selectlistitem ::= expression \| expression aliasname \| expression AS aliasname \| aliasname . *` |
| `where ::=  \| WHERE boolean` |
| `having ::=  \| HAVING boolean` |
| `boolean ::= and \| and OR boolean` |
| `and ::= not \| not AND and` |
| `not ::= comparison \| NOT comparison` |

| SQL Grammar |
|---|
| comparison ::= ( boolean ) \| colref IS NULL \| colref IS NOT NULL \| expression LIKE pattern \| expression NOT LIKE pattern \| expression IN ( valuelist ) \| expression NOT IN ( valuelist ) \| expression op expression\|EXISTS(SELECT select)\| expression op selectop ( SELECT select ) \| expression IN ( SELECT select ) \| expression NOT IN ( SELECT select ) \| expression BETWEEN expression AND expression \| expression NOT BETWEEN expression AND expression |
| selectop ::=  \| ALL \| ANY |
| op ::= > \| >= \| < \| <= \| = \| <> |
| pattern ::= string \| ? \| USER |
| expression ::= expression + times \| expression - times \| times |
| times ::= times * neg \| times / neg \| neg |
| neg ::= term \| + term \| - term |
| term ::= ( expression ) \| colref \| simpleterm \| aggterm \| scalar |
| scalar ::= scalarescape \| scalarshorthand |
| scalarescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) FN fn )*-- |
| scalarshorthand ::= { FN fn } |
| fn ::= functionname ( valuelist ) \| functionname ( ) \| POSITION ( expression IN expression ) \| EXTRACT ( expression FROM expression ) |
| aggterm ::= COUNT ( * ) \| AVG ( expression ) \| MAX ( expression ) \| MIN ( expression ) \| SUM ( expression ) \| COUNT ( expression ) |
| simpleterm ::= string \| realnumber \| ? \| USER \| date \| time \| timestamp |
| groupby ::= \| GROUP BY groupbyterms |
| groupbyterms ::= colref \| colref , groupbyterms |
| orderby ::= \| ORDER BY orderbyterms |
| orderbyterms ::= orderbyterm \| orderbyterm , orderbyterms |
| orderbyterm ::= colref asc \| integer asc |
| asc ::=  \| ASC \| DESC |
| colref ::= aliasname . columnname \| columnname |
| tablelist ::= tablelistitem , tablelist \| tablelistitem |
| tablelistitem ::= tableref  \| outerjoin |
| outerjoin ::= ojescape \| ojshorthand |
| ojescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) OJ oj )*-- |
| ojshorthand ::= { OJ oj } |
| oj := tableref LEFT OUTER JOIN tableref ON boolean \|  tableref LEFT OUTER JOIN oj ON boolean |

| SQL Grammar |
|---|
| tableref ::= tablename \| tablename aliasname |
| indexname ::= identifier |
| functionname ::= identifier |
| tablename ::= identifier |
| datatype ::= identifier |
| columnname ::= identifier |
| aliasname ::= identifier |
| identifier ::= an identifier (identifiers containing spaces must be enclosed in double quotes) |
| string ::= a string (enclosed in single quotes) |
| realnumber ::= a non-negative real number (including E notation) |
| integer ::= a non-negative integer |
| date ::= dateescape \| dateshorthand |
| dateescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) d dateval )*-- |
| dateshorthand ::= { d dateval } |
| dateval ::= a date in yyyy-mm-dd format in single quotes (for example, '1996-02-05') |
| time ::= timeescape \| timeshorthand |
| timeescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) t timeval )*-- |
| timeshorthand ::= { t timeval } |
| timeval ::= a time in hh:mm:ss format in single quotes (for example, '10:19:48') |
| timestamp ::= timestampescape \| timestampshorthand |
| timestampescape ::= --*(VENDOR(MICROSOFT),PRODUCT(ODBC) ts timestampval )* -- |
| timestampshorthand ::= { ts timestampval } |
| timestampval ::= a timestamp in yyyy-mm-dd hh:mm:ss[.ffffff] format in single quotes (for example, '1996-02-05 10:19:48.529') |

# 8 Embedding MQL: MQLIO and MatrixMQL

## Overview of MQLIO and MatrixMQL

### MQLIO

MQLIO is a library of functions used to embed MQL commands into programs so that they can communicate with an ENOVIA Live Collaboration database. MQLIO applications can be used with any ENOVIA Live Collaboration database.

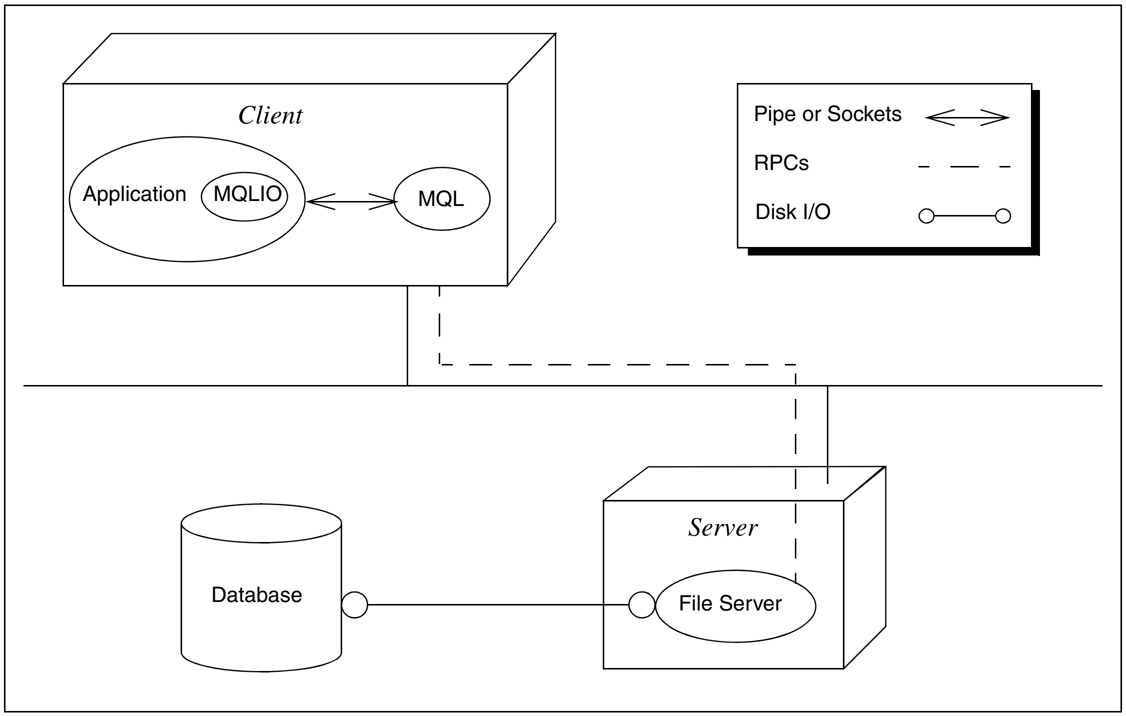
MQLIO manages a client MQL process to interface to the database. It also provides services to send commands to the MQL process and receive corresponding output.

### MatrixMQL

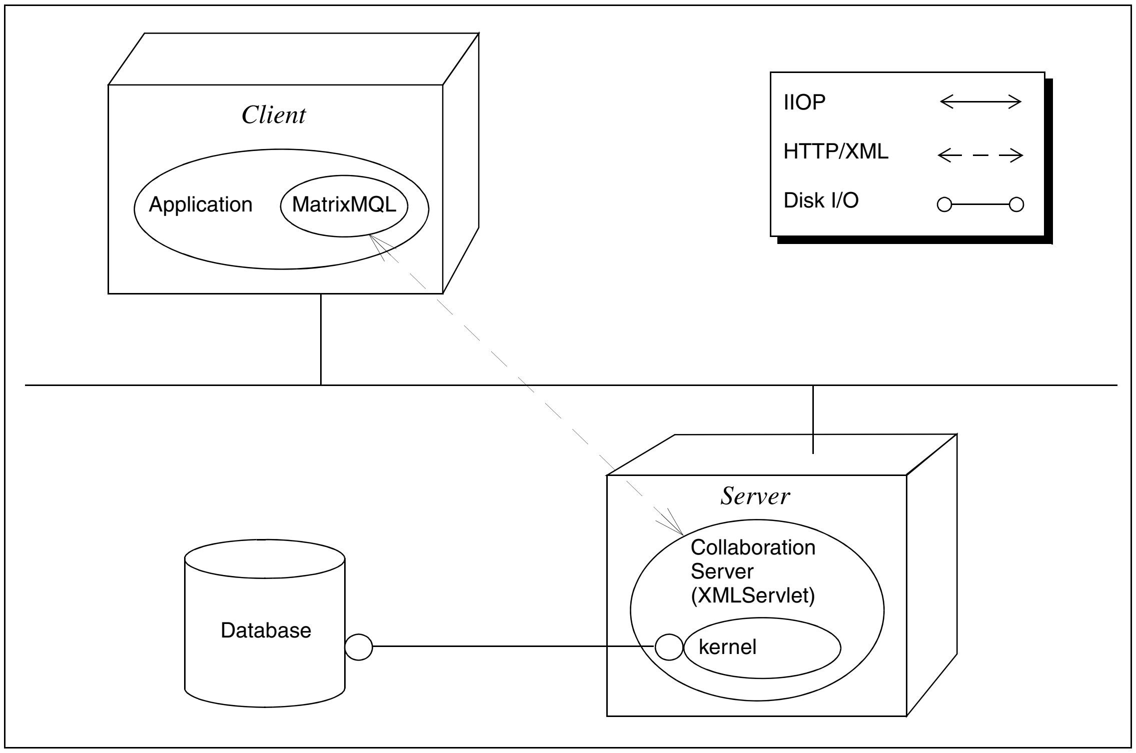
With MatrixMQL, an application sends MQL commands to the Live Collaboration Server, which accesses the ENOVIA Live Collaboration database and returns corresponding output.

*MatrixMQL is distributed with the ENOVIA Studio Customization Toolkit.*

**MQLIO ARCHITECTURE**



**MatrixMQL ARCHITECTURE**



## Requirements

In order to build applications with the MQLIO or MatrixMQL functions, you must have:

- An installed ENOVIA Live Collaboration database, from which to access data.
- The appropriate function library installed, MQLIO or MatrixMQL.

In addition, MatrixMQL requires:

- The Java 1.3.1 or greater Runtime must be installed
- The path of any calling program must include the jre/bin and jre/bin/class paths of Java
- The environment variable CLASSPATH must include the .jar file that contains the ENOVIA Collaboration Live Collaboration server classes.

For all, the minimum compiler version for Windows is Microsoft Visual C$^{++}$ 6.0 or higher.

It is important to note that prior to ENOVIA Live Collaboration version 10.8, ENOVIA products and MQLIO libraries were compiled with Microsoft Visual Studio 6. In ENOVIA Live Collaboration version 10.8 and after, Microsoft Visual Studio 2005 was used. The compiler version of the MQLIO libraries must be the same as the compiler version of the ENOVIA product calling them. That means MQLIO libraries compiled with Microsoft Visual Studio 6 can only be called by ENOVIA products compiled with Microsoft Visual Studio 6.

See the *ENOVIA Live Collaboration Installation Guide* for details.

## Comparison of MQLIO and MatrixMQL

The differences between MQLIO and MatrixMQL are shown in the following table:

| MQLIO | MatrixMQL |
|---|---|
| For non-multi-user applications | For multi-user applications |
| Uses Sockets under Microsoft Windows, or named pipes under UNIX | Uses XML wrapped by HTTP |
| Supports both 16-bit and 32-bit compilers | Supports 32-bit compilers only |
| Communicates with MQL.EXE running on a client machine | Communicates with Collaboration Server running on a server |

## Known Issues

### Do Not Use mqlOpen Function More than Once

Using the `mqlOpen` function more than once in a program will cause your program to fail. Do not use `mqlOpen` more than once even if you use `mqlClose` before calling the second `mqlOpen`. To prevent the need for calling `mqlOpen` a second time, don't use `mqlClose` until your program has no further business with ENOVIA Live Collaboration. For example, call `mqlOpen` on initialization and `mqlClose` on termination of the entire Studio Customization Toolkit program.

# C++ Studio Customization Toolkit

Installing the ENOVIA Live Collaboration Applet XML Studio Customization Toolkit with the C++ option installs MatrixMQL libraries and example source code in the INSTALL_DIR/cxx directory structure. MatrixMQL is a C++ XML client that can interact with the Live Collaboration server running on the server.

Refer to the sections that follow for additional requirements for creating and using custom C++ programs.

## C++ Studio Customization Toolkit Library Structure

The C++ Studio Customization Toolkit libraries should be installed within a cxx subdirectory of the existing Studio Customization Toolkit installation directory.

```
cxx
    | - - -src
          |- - - include              non platform dependent includes
          |     | - - -winnt          Windows-dependent includes
          |     | - - -hp9000s800     HP-dependent includes
          |     | - - -solaris4       Solaris-dependent includes
          |     | - - -ibmrs6000      AIX-dependent includes
          |- - - mql                  simple MQL program sample source
          |- - - ematrixmql           MatrixMQL sample source
|         bin
          |     | - - -winnt          contains Windows executable
          |     | - - -hp9000s800     contains HP executable
          |     | - - -solaris4       contains Solaris executable
          |     | - - -ibmrs6000      contains AIX executables
    | - - -lib
          |     | - - -winnt          contains Windows client cxx library
          |     | - - -hp9000s800     contains HP/UX client cxx library
          |     | - - -solaris4       contains Solaris client cxx library
          |     | - - -ibmrs6000      contains AIX client cxx library
    | - - -etc
          |     | - - -test           **Contains include files and programs that can be used as a
                - - - -               test harness for invoking the MatrixMQL libraries
                | - - -|   winnt      executables for winnt
                | - - -|   hp9000s800 executables for HP
                | - - -|   solaris4   executables for Solaris
                | - - -|   ibmrs6000  executables for AIX
```

## Setting Up a Client Machine to Run a MatrixMQL Application

This section describes how to set up a client machine to run an application that was compiled with MatrixMQL. It assumes:

- the client machine does not have the Live Collaboration server, Studio Customization Toolkit, or any other ENOVIA Live Collaboration component installed
- the custom application was compiled on a separate server machine that has the Collaboration Server and the C++ Studio Customization Toolkit installed.

The instructions below list the main steps for setting up a Windows client machine for a MatrixMQL application built on a Windows server machine using the Live Collaboration server Studio Customization Toolkit. The steps are the same for UNIX except:

- the directory references should be / instead of \
- instead of “winnt”, the subdirectories within the Studio Customization Toolkit installation directories are named according to the platform: solaris4, hp9000s800, or ibmrs6000
- The MatrixMql shared library is called eMatrixMql.so for Solaris and AIX, and eMatrixMql.sl for HP/UX.

#### To configure a client machine to run a MatrixMQL application

1. Copy the Matrix jar file, eMatrixServletRMI.jar from the server machine and paste it to the client machine. You can find the jar file in ENOVIA_INSTALL\server\java\classes.

   Make sure the jar file is the exact same version as the Studio Customization Toolkit used to compile the program.
2. On the client machine, add the path and filename of the ENOVIA Live Collaboration jar file to CLASSPATH.

   The specific name of the file is needed in addition to the full path. Adding only the path produces the error message “Can't find com.matrixone.jdl.MatrixSession”.
3. Copy eMatrixMql.dll from the ADK_INSTALL\cxx\lib\winnt directory to the client machine directory that contains the MatrixMQL application.
4. Make sure the client machine has the correct version of the JDK or the Java Runtime Environment (JRE). Refer to the *ENOVIA Live Collaboration Installation Guide* or readme for supported patch levels.
5. Add the path that includes the Java jvm.dll to the PATH.
6. Now you can run your custom application. For initial testing, you can run the test application that comes with the MatrixMQL installation. See “*Running the MatrixMQL Test Application*”.

## C++ Studio Customization Toolkit Files

### MatrixMql Files (Live Collaboration server only)

In the following, [platform] = winnt, hp9000s800, AIX (Live Collaboration server only), or solaris 4.

#### Client C++ header definitions

```
cxx\src\include\[platform]\bos.h
cxx\src\include\[platform]\eMatrixMql.h
```

#### eMatrixMql code sample, DLL and Static libraries

```
cxx\src\eMatrixMql\BOAImpl.C
cxx\src\eMatrixMql\BOAImpl.h
cxx\src\eMatrixMql\bos.h
cxx\src\eMatrixMql\bosClient.C
cxx\src\eMatrixMql\bosCommon.C
cxx\src\eMatrixMql\bosServer.C
cxx\src\eMatrixMql\eMatrixMql.C
cxx\src\eMatrixMql\eMatrixMql.h
cxx\src\eMatrixMql\eMatrixMqlUtils.h
cxx\src\eMatrixMql\List.C
cxx\src\eMatrixMql\MatrixSerializable.C
```

```
cxx\src\eMatrixMql\MatrixSerializable.h
cxx\src\eMatrixMql\MatrixSession.C
cxx\src\eMatrixMql\MatrixSession.h
cxx\src\eMatrixMql\MqlString.C
cxx\src\eMatrixMql\MqlString.h
cxx\src\eMatrixMql\mx.h
cxx\src\eMatrixMql\mxMutex.C
cxx\src\eMatrixMql\mxString.C
cxx\src\eMatrixMql\mxThread.C
cxx\src\eMatrixMql\StringMacros.h
```

### eMatrixMql DLL and Static libraries (Windows only)

```
cxx\lib\winnt\eMatrixMql.dll(DLL)
cxx\lib\winnt\eMatrixMql.lib(DLL stub library)
cxx\lib\winnt\eMatrixMqlU.dll(Unicode DLL)
cxx\lib\winnt\eMatrixMqlU.lib(Unicode DLL stub library)
cxx\lib\winnt\eMatrixMqlS.lib(Static library)
cxx\lib\winnt\eMatrixMqlSU.lib(Unicode static library)
```

### eMatrixMql DLL and Static libraries (UNIX only)

```
cxx/lib/hp9000s800/eMatrixMqlS.o(Static library)
cxx/lib/solaris4/eMatrixMqlS.o(Static library)
```

### eMatrixMql Test Executables

```
cxx/etc/test/eMatrixMqlTest.h
cxx/etc/test/eMatrixMqlTest.C
cxx/etc/test/[platform]/eMatrixMQLTest32.exe(Test program)
cxx/etc/test/[platform]/eMatrixMQLTest32u.exe(Unicode test
program)
```

You must also have access to the following Galaxy dll’s via the PATH setting:

```
vgal7u.dll
vgal_aux.dll
vgalaxy7.vr
vgalx16.dll
```

## Porting an MQLIO Application to a MatrixMQL-based Application

MatrixMQL contains all the functionality in MQLIO, except the metric functions. Every MatrixMQL function and its parameters are identical to MQLIO, to allow for an easy migration.

These are the steps needed to convert your Mqlio-based application to use MatrixMQL:

1. Change all references of mqlio.h to eMatrixMQL.h in your code.
2. Add mqlSetContext (host, user ID, password, lattice), prior to the `mqlOpen()` call. This sets the information needed to login to the Collaboration Server.

   | | | |
   |---|---|---|
   | `host` | = | host name of server running the Collaboration Server |
   | `user ID` | = | ENOVIA Live Collaboration User ID |
   | `password` | = | ENOVIA Live Collaboration Password |
   | `lattice` | = | Lattice to set context in (optional) |

3. Link the MatrixMQL static library or DLL stub library in place of the MQLIO library.

   Do NOT link in `cxx\lib\libclient.lib`; this library is already included in MatrixMQL (otherwise you will get numerous link errors).
4. Install and set up the Collaboration Server on a server.
5. Set up the client side as outlined in *Running the MatrixMQL Test Application*.

## Running the MatrixMQL Test Application

The MatrixMQL installation includes sample executables (one for unicode and one for non-unicode environments) that you can use as a test harness for invoking the MatrixMQL libraries. The include files and exe files needed for the test application are in cxx\etc\test\PLATFORM, where PLATFORM is winnt, solaris 4, or hp9000s800. This section describes how to set up and run the test application.

### Setting Up to Run the Test Application

This section lists the main steps for setting up a Windows system to run the test MatrixMQL applications. The steps are the same for UNIX except the directory references should be / instead of \.

**To install and configure a server to run the test application**

1. Install the ENOVIA Live Collaboration Server.
2. Install the Applet Studio Customization Toolkit and choose the C++ Studio Customization Toolkit. Make sure the Studio Customization Toolkit version matches the version of the Live Collaboration Server that you installed.
3. Copy eMatrix.dll from ENOVIA_INSTALL\server\bin\winnt to the directory that contains the test applications, which for Windows is cxx\etc\test\winnt.

   Note that the system can find the dll if it's located in any directory that's in the operating system's search path for exe files. But placing the dll in the directory with the test exe files ensures it picks up the correct version of the dll, in case you have other versions of the dll.
4. Copy eMatrixMql.dll from cxx\lib\winnt to the directory that contains the test cxx\etc\test\winnt.
5. Set CLASSPATH=\matrix\java\classes\eMatrixServletRMI.jar.
6. Set PATH=\jre\1.4\bin\classic.

Now run the test application using the steps in the following section.

### Running the Test Application

**To run the test application for MatrixMQL**

1. Create a text file in the cxx\etc\test\winnt directory with whatever MQL commands you want to run. This is the script you will pass to the test application.

   For example, you can create a file with the simple command "`version;`" (which works on any database) and save it as version.mql. The resulting cxx\etc\test\winnt directory would now contain these files:

   - eMatrix.dll
   - eMatrixMql.dll
   - eMatrixMqlTest32.exe
   - eMatrixMqlTest32u.exe
   - version.mql

2. In that same cxx\etc\test\winnt directory, issue the command to run the test application eMatrixMqlTest32.exe. The exe accepts the following arguments:

| Command Argument | Description | Example |
|---|---|---|
| eMatrixMqlTest32 <host server name> | Name of the host server, prefixed with // and appended with a colon and the port number. Make sure the port number is the Live Collaboration server port (for example, 1099) for the Live Collaboration server. | //hostname:1099 |
| -s <mqlscript> | Name of the mql script to pass to the program (stdin). | -s version.mql |
| -l <outputlog> | Name of log file to send output to (stdout). | -l version.log |
| -d <level> | Set level of MatrixMQL diagnostics: 0,1,2; default=1. | -d 1 |
| -h | Display help | -h |
| -p | Use pending vs. DDE callbacks (=default) | -p |
| -u <username>[/<pswd>] | Set context to the specified username and password. | -u creator |
| -v | Turn on verbose output. | -v |

For example, to run the version.mql script, you could issue the command:

```
eMatrixMqlTest32 //hostname:1099 -s version.mql -l version.log
-d 2 -u creator -v
```

The output window will look as follows:

```
Calling mqlSetVerbose()
Calling mqlSetTrace()
Calling mqlInit()
Calling mqlSetContext() initially
Calling mqlSetContext() again
Calling mqlOpen()
Calling mqlOpenLog()
Calling mqlCommand()
Calling mqlCallback()
Calling mqlOutputLine() through callback
Calling mqlErrors()
Calling mqlCommand()
Calling mqlCallback()
Calling mqlErrors()
Calling mqlCommand()
Calling mqlCallback()
Calling mqlErrors()
Calling mqlCloseLog()
Calling mqlClose()
eMatrixMqlTest Completed.
```

3. If the application runs successfully, you should find the output log file in the cxx\etc\test\winnt directory. Open the file to check its content.

   For example, the version.log file would contain something similar to this:

```
Input:version;
Output:10.5-Global
Input:
Input:
```

## Troubleshooting

Use this troubleshooting table to help resolve problems with running the test or a custom application.

| Error | Remedy |
|---|---|
| Can’t find or load eMatrix.dll | Copy ematrix.dll from ENOVIA_INSTALL\server\bin\winnt to the same directory as the calling program. |
| Unable to load eMatrixMql.dll | Copy ematrix.dll from ADK_INSTALL\cxx\lib\winnt to the same directory as the calling program. |
| Class not found errors | Make sure the ematrix.jar file is in the CLASSPATH. |
| Can’t find jvm.dll | The system environment PATH variable is pointing to the wrong jvm.dll. Make sure the JDK or JRE directory that contains the jvm.dll is in your PATH. For example, for JDK 1.3.3, the dll is in the hotspot directory, \jre\bin\hotspot.<br>Make sure the jdk/jre/REV/bin/classic is in your CLASSPATH. |
| Can’t create Java vm | The CLASSPATH is pointing to the wrong jvm.dll. |
| Can't find com.matrixone.jdl.MatrixSession | Make sure the CLASSPATH is defined and includes the full pathname and filename, for example "Set CLASSPATH=\PATHNAME\eMatrixServletRMI.jar" |
| Dr. Watson errors running the eMatrixMQL application | Be sure that the JRE or JDK is correctly installed and that the system environment PATH variable points to the correct Java binary path. |
| File checkin / checkout does not work | Need to add the keyword "client" to the checkin and “server” for the checkout command to specify where the operation will occur. For checkin the default is server and for checkout the default is client.<br>Also, the checkout command requires that a target directory specified. Refer to the *MQL Guide* for more information. |

## Starting a Session

A session is started using the `mqlOpen` command. The `mqlOpen` command will open communications between your client application and MQL or the server. For MQLIO, pipes are used under the UNIX Operating System, while a Socket interface is used under Windows. MatrixMQL uses XML wrapped by HTTP.

In order to track any errors that may occur, it is recommended that a log file be opened as well, using the `mqlOpenLog` command after `mqlOpen` is called. The log file will contain all input to and output from MQL or the server.

A session might begin as follows:

*NOTE: For MatrixMQL, the `mqlSetContext` command must be called before the `mqlOpen` command. For MQLIO, the session would begin with the `mqlOpen` command, as shown below.*

```
    if (mqlSetContext(“WebServer”,”JSmith”,”xyzzy”,”Engineering”) == MQLERROR)
    {
       fprintf(stderr, ”ERROR: Cannot Set Context....”);
    }
```

```
    if (mqlOpen() == MQLERROR)
    {
       fprintf(stderr, ”ERROR: Cannot Open MQL Interface\n”);
       exit(-1);
    }
```

`mqlOpen` returns one of the following:

- **MQLOK (0)** The session has been established.
- **MQLERROR (-1)** The session was unable to be established.

```
if (mqlOpenLog(“example.log”) == MQLERROR)
{
   fprintf(stderr, ”ERROR: Cannot Open Log File\n”);
   exit(-1);
}
```

`mqlOpenLog` returns one of the following:

- **MQLOK (0)** The log file was created without any problems.
- **MQLERROR (-1)** The log file could not be created.

See *mqlOpen*, *mqlClose*, *mqlOpenLog*, *mqlCloseLog* for further information.

## Issuing a Command

*There is an MQL command length limit of 8K.*

Commands are issued to the MQL process using the `mqlCommand` routine. The parameters used to format the commands are just like the C Language `printf` routine, in that it takes multiple parameters and uses format strings to control the layout of the command.

```
if (mqlCommand( “print businessobject \’%s\’ \’%s\’ \’%s\’ “, type. name.
revision) == MQLERROR)
{
   return FAILURE;
}
```

Assuming that type is equal to ‘Assembly’ and name is equal to ‘Engine’ and revision is equal to ‘A’, this is equivalent to issuing a `print businessobject ‘Assembly’ ‘Engine’ ‘A’` command from the MQL prompt.

*In the above example, the return of MQLERROR from `mqlCommand` does NOT mean that the `print businessobject` command failed, but that the `mqlCommand` function failed. Some reasons for failure are (a) the interface has not been initialized or (b) an input error occurred.*

`mqlCommand` returns one of the following:

- **MQLOK (0)** The sent command was successfully invoked.
- **MQLERROR (-1)** The sent command was not successfully invoked.

*If a callback function is not used when issuing a command, output should be checked with the `mqlPending` command.*

*The C++ Studio Customization Toolkit interface mqlCommand requires multibyte input to be in Unicode format.*

## Processing Output with Callbacks

In most programming situations, an application program calls the application programming interface (API), and then the API executes the function called by the application program and returns control, and possibly a value, to the calling program.

In some instances, however, it is necessary for the API to call the application program, either to request additional information or to allow the application program to process some data. The calling of an application program by an API is referred to as a *callback* (thus named because the API is called by the application program and then *calls back*).

Callbacks are often used to handle output or error conditions. To set up a callback, the application program usually *registers*, that is passes to the API, a pointer to a user-defined function. The API generally provides a registration function that accepts a function address as one of its parameters for this purpose. When the specific condition occurs, the API calls the user-defined function (through the registered function pointer) and passes control to that function for execution.

The `mqlCallback` function of MQLIO/MatrixMQL is a flexible way for a programmer to extract data from the ENOVIA Live Collaboration database using MQL commands. The `mqlCallback` function requires four (4) parameters:

- A pointer to a function to handle output
- A pointer to an output data area
- A pointer to a function to handle error output
- A pointer to an error output data area

The functions to handle output data and error output data are supplied by the programmer. Simple examples of functions to print data to standard output are given below. They use the MQLIO/MatrixMQL commands *mqlOutputLine* and *mqlErrorLine*..

```
/* outputCallback.c -- function to handle output callback. */
/****************************************************/
#include <stdio.h>
#include "outputCallback.h"                    /* Function prototype */
#ifdef MQLIO
#include "mqlio.h"
#else
#ifdef EMATRIXMQL
#include "eMatrixMQL.h"
#endif
/****************************************************/
#define LINEBUFFERSIZE 8192
/****************************************************/
int outputCallback(void *outData)
/* Function to handle output callback.
** input: none.
** output: Data from MQL output.
** returns: MQLOK....output callback was handled.
** returns: MQLERROR output callback could not be handled.
*/
{
   char buffer[LINEBUFFERSIZE];                    /* Buffer */
   if (mqlOutputLine(buffer,LINEBUFFERSIZE))
   {
      fprintf(stdout,"%s",buffer);
   }
   return MQLOK;
}
```

```
/* errorCallback.c -- function to handle error output callback. */
/***********************************************************/
#include <stdio.h>
#include "errorCallback.h"                      /* Function prototype */
#ifdef MQLIO
#include "mqlio.h"
#else
#ifdef EMATRIXMQL
#include "eMatrixMQL.h"
#endif
/**********************************************************/
#define LINEBUFFERSIZE 8192
/************************************************************/
int errorCallback(void *errData)
/* Function to handle error output callback.
** input: none.
** output: Data from MQL error output.
** returns: MQLOK.......error output callback was handled.
** returns: MQLERRORerror output callback could not be handled.
*/
{
   char buffer[LINEBUFFERSIZE];                        /* Buffer */
   if (mqlErrorLine(buffer,LINEBUFFERSIZE))
   {
      fprintf(stderr,"%s",buffer);
   }
   return MQLOK;
}
```

The `mqlCallback` call would be:

```
int iStat;
iStat = mqlCallback(outputCallback, NULL, errorCallback, NULL);
```

`mqlCallback` returns one of the following:

- **MQLOK (0)** The callback was successful.
- **MQLERROR (-1)** The callback failed.

See also *mqlCallback*.

## Processing Pending Output

MQL output can also be processed using `mqlPending` to determine if output is available for processing.

```
/************************************************/
int status;
for (;;)
{
   status = mqlPending();
   while (status == MQLOK)
   {
      sleep(1);
      status = mqlPending();
   }
   switch (status)
   {
      case MQLOUTPUTPENDING:
         mqlOutputLine(buffer,size);
                   :
                   :
         break;
      case MQLERRORPENDING:
         mqlErrorLine(buffer,size);
                   :
                   :
         break;
      default:
         break;
   }
}
/************************************************/
There are four possible returns:
```

- **MQLERRORPENDING (2)** There is error output pending.
- **MQLOUTPUTPENDING (1)** There is pending output.
- **MQLOK (0)** There is no pending output.
- **MQLERROR (-1)** The process timed out.

See also *mqlPending*.

## A Small Conversation

Here are two examples of a simple *conversation*, the first using `mqlCallback`, the second using `mqlPending`. The conversation requests the list of vaults using the MQL command `list vault`. All output will be printed to standard out or standard error.

```
/***************************************************/
/* mqlCallback */

   char cmd = "list vault";

   if (mqlCommand(cmd)== MQLERROR)
   {
      fprintf(stderr,"ERROR: mqlCommand() routine has failed\n");
      return FAILURE;
   }

   if (mqlCallback(outputCallback, NULL, errorCallback, NULL) == MQLERROR)
   {
      fprintf(stderr,"ERROR: mqlCallback() routine has failed\n" );
      return FAILURE;
   }

   if (mqlErrors())
   {
      fprintf( stderr, "ERROR: MQL Command %s failed\n", cmd );
      return FAILURE;
   }
```

See the example in the *Processing Pending Output* Section, above, for the `outputCallback` and `errorCallback` function definitions.

```
/* mqlPending */

   char cmd = "list vault";
   int errorCnt = 0;
   int outputCnt = 0;
   int status;
   int done;

   if (mqlCommand(cmd) == MQLERROR)
   {
      fprintf(stderr, "ERROR: mqlCommand() routine has failed\n" );
      return FAILURE;
   }
   for (;;)
   {
      status = mqlPending();
      while (status == MQLOK)
      {
         if (mqlEOF())
         {
            done = TRUE;
            break;
         }
         sleep(1);
         status = mqlPending();
      }
      if (done)
         break;

      switch (status)
      {
         case MQLOUTPUTPENDING:
                   mqlOutputLine(buffer,size);
                   fprintf( stdout, "%s\n", buffer );
                   outputCnt++;
                   break;
         case MQLERRORPENDING:
                   mqlErrorLine(buffer,size);
                   fprintf( stderr, "%s\n", buffer );
                   errorCnt++;
                   break;

         default:
                   break;
      }
   }

   if (errorCnt)
   {
      fprintf( stderr, "MQL Command %s failed\n", cmd );
      return FAILURE;
   }
/**************************************************/
```

## Dialog Scripting Language

Programs can be written that can pop up many ENOVIA Live Collaboration dialogs through the use of the `application` command. The application command provides programmers with a graphical user interface alternative to Tk, and one which provides ENOVIA Live Collaboration familiarity. Since the dialogs must be accessible from the launching process, these programs can be:

- An MQL or Tcl program object run either within ENOVIA Live Collaboration (via a tool, method, or trigger) or with the “matrix -mql” command line option. (`matrix -mql -c “execute program NAME;”`).
- An MQL script run with the “matrix -mql” command line option (`matrix -mql -c “run FILE.mql;”`).
- Commands passed by an external program or application which is talking to ENOVIA Live Collaboration -mql through an MQLIO connection that was opened using the mqlopen2 interface, passing `matrix.exe` as the ‘executable’ argument.

Refer to the *Command Line Execution* in Chapter 2 for more information on the first two options.

For details on the `application` command, refer to *Dialog Scripting Language* appendix.

## Library of Functions

### Library Listing

The following files are distributed for use with MQLIO applications:

| File | Usage | Use With | Calling Convention |
|---|---|---|---|
| mqlio16.dll | 16-bit DLL | VC-16 or VB | FAR PASCAL |
| mqlio16.lib | 16-bit DLL Stub Library | VC-16 | FAR PASCAL |
| mqlio32.dll | 32-bit DLL | VC-32 | C |
| mqlio32.lib | 32-bit DLL Stub Library | VC-32 | C |
| mqlVB32.dll | 32-bit DLL | VB-32 | STD CALL |
| mqlVB32.lib | 32-bit DLL Stub Library | VB-32 | STD CALL |
| mqlio32.o | 32-bit Library | Unix | C |
| mqlio32s.lib | 32-bit Static Library | VC-32 | C |

The following files are the distributed for use with MatrixMQL applications:

| File | Usage |
|---|---|
| eMatrixMQL.h | Header File |
| eMatrixMQL.dll | Windows DLL |
| eMatrixMQL.lib | Windows DLL stub library (this gets linked to your application vs. the DLL) |
| eMatrixMQLs.def | Windows DLL Function Export List |
| eMatrixMQLS.lib | Windows Static Library |
| mqlio32.o | UNIX Static Library |

### Library of MQLIO and MatrixMQL Functions

This section lists and describes the public functions in the MQLIO and MatrixMQL libraries. Syntax and output, as well as additional notes where applicable, are provided in the following section.

| **Function** | **Description** |
|---|---|
| mqlCallback | This function is a setup routine to handle MQL or the Collaboration Server output and errors. The user passes in a pointer to a function that returns an integer and a storage location for any results. This is done for both normal and error output. |
| mqlClose | Ends an open session by terminating the MQL process or the Collaboration Server connection. |
| mqlCloseLog | Closes the log file opened using mqlOpenLog. |
| mqlCommand | Sends an MQL command to MQL or the Collaboration Server, using printf style syntax. |
| mqlDisableLog | Disables writing to the log file. |
| mqlEnableLog | Enables writing to the log file. |
| mqlEOF | Checks for the MQL prompt to determine whether or not the MQL process is ready for input from MQLIO. (This function is not needed for MatrixMQL.) |
| mqlError | Used by mqlCallback to receive error data from MQL or the Collaboration Server. |
| mqlErrorLine | Used by mqlCallback to receive a line of error data from MQL or the Collaboration Server. It breaks on new line. |
| mqlErrorMessage | Prints an error message. |
| mqlErrors | Maintains a counter of the number of times the user-supplied error Callback routine has been called. |
| mqlExecute | Sends an MQL command to the MQL process or the Collaboration Server session for execution. |
| mqlInit | For MatrixMQL only, resets all static string variables for the host, user ID, password, and vault to blank values. |
| mqlInput | Used by mqlCommand to send the contents of the input buffer to MQL or the Collaboration Server. |
| mqlOpen | Starts a session and establishes communications with MQL or the Collaboration Server. MQLIO uses DDE on Windows or named pipes on UNIX; MatrixMQL uses HTTP/XML. |
| mqlOpenLog | Opens a log file filename for the current session. It reads the input and output buffers and keeps a record of input to MQL or the Collaboration Server and output from MQL or the Collaboration Server. |
| mqlOutput | Used by mqlCallback to receive data from MQL or the Collaboration Server. |
| mqlOutputLine | Used by mqlCallback to receive a line of data from MQL or the Collaboration Server. It breaks on new line. |
| mqlOutputs | Maintains a counter of the number of times the user-supplied outputCallback routine has been called, by recording the number of outputs from mqlCallback. |

| Function | Description |
|---|---|
| mqlPending | Checks for any output data that is waiting to be processed. It is useful for commands that generate a large amount of output, because it begins to process the output data as soon as it is available. mqlCallback is the better function to use. |
| mqlSetContext | For MatrixMQL only, sets the context with User, Password and Vault parameters. It is the first MatrixMQL function your program calls. An additional parameter is required for the Collaboration Server host name. |
| mqlSetQuote | For MatrixMQL only, turns MQL quoting on. |
| mqlTimeout | Sets the sleep function: sleeps for x seconds. |
| mqlWriteToLogFile | Writes data to the log file. |

## Synopsis of MQLIO and MatrixMQL Functions

The tables that follow provide syntax and additional notes on the MQLIO and MatrixMQL functions.

| Function | **mqlOpen** |
|---|---|
| Synopsis | `int mqlOpen(void);` |
| Description | This function starts a session and establishes communications with MQL or the Collaboration Server. MQLIO uses DDE on Windows or named pipes on UNIX; MatrixMQL uses HTTP/XML. |
| Returns | **MQLOK** means that the session has been established.<br>**MQLERROR** means that session has not been established. |

| Function | **mqlClose** |
|---|---|
| Synopsis | `int mqlClose(void);` |
| Description | This function ends an open session by terminating the MQL process. |
| Returns | **MQLOK** means the session was shutdown successfully.<br>**MQLERROR** means the session shutdown encountered a problem. |
| Note | Once you have opened a session using mqlOpen, you must call mqlClose to end the session before exiting your program. When using MQLIO, if you exit without calling mqlClose, the MQL process will remain open, taking up system resources. |

.

| | |
|---|---|
| **Function** | **mqlnit** |
| **Synopsis** | `void mqlInit(void);` |
| **Description** | For MatrixMQL only, this function resets all static string variables for the host, user ID, password, and vault to blank values. |
| **Returns** | void |

| | |
|---|---|
| **Function** | **mqlSetContext** |
| **Synopsis** | `void mqlSetContext(MQLBOS_STR_PTR host, MQLBOS_STR_PTR user, MQLBOS_STR_PTR password, MQLBOS_STR_PTR vault);` |
| **Description** | For MatrixMQL only, this function sets the context variables with the User, Password and Vault parameters. It is the first MatrixMQL function your program calls. The additional parameter, MQLBOS_STR_PTR host, provides the Live Collaboration Server host name. |
| **Returns** | void |

| | |
|---|---|
| **Function** | **mqlOpenLog** |
| **Synopsis** | `int mqlOpenLog(char * filename);`<br>`char * filename` is a pointer to a character string containing the path and name of the log file to be opened. |
| **Description** | This function opens a log file *(filename)* for the current session. It reads the input and output buffers and keeps a record of input to MQL and the Collaboration Server and output from MQL and the Collaboration Server. |
| **Returns** | **MQLOK** means the log file was opened successfully.<br>**MQLERROR** means the log file could not be opened. |

| | |
|---|---|
| **Function** | **mqlCloseLog** |
| **Synopsis** | `int mqlCloseLog(void);` |
| **Description** | This function closes the log file opened using mqlOpenLog. |
| **Returns** | **MQLOK** means the log file was closed successfully.<br>**MQLERROR** means the log file was not properly closed. |

| | |
|---|---|
| **Function** | **mqlEOF** |
| **Synopsis** | `int mqlEOF(void);` |
| **Description** | For MQLIO only, this function checks for the MQL prompt to determine whether or not the MQL process is ready for input from MQLIO. |
| **Returns** | **TRUE (1)** means MQL is ready.<br>**FALSE (0)** means MQL is not ready. |

| | |
|---|---|
| **Function** | **mqlPending** |
| **Synopsis** | `int mqlPending (void);` |
| **Description** | This function checks for any output data that is waiting to be processed. It is useful for commands that generate a large amount of output, because it begins to process the output data as soon as it is available. mqlCallback is the better function to use. |
| **Returns** | **MQLERRORPENDING (2)** means that error output is pending.<br>**MQLOUTPUTPENDING (1)** means output is pending.<br>**MQLOK (0)** means that no output is pending.<br>**MQLERROR (-1)** means that a time-out has occurred. |

| | |
|---|---|
| **Function** | **mqlTimeout** |
| **Synopsis** | `int mqlTimeout (int seconds);` |
| **Description** | Sets the sleep function: sleeps for `int` seconds. |
| **Returns** | **MQLOK** means the timeout was set successfully. |

| **Function** | **mqlCallback** |
|---|---|
| **Synopsis** | `int mqlCallback (int (*outputCallback) (void * outData),`<br>`void * outputData, int (*errorCallback) (void * errData),`<br>`void * errorData);`<br>`int (*outputCallback) (void * outData)` is a pointer to a user-supplied output handling function that returns an integer and has outData as a parameter.<br>`void * outputData` is an output data storage location passed to mqlCallback from the output handling function.<br>`int (*errorCallback) (void * errData)` is a pointer to a user-supplied error handling function that returns an integer and has errData as a parameter.<br>`void * errorData` is the error data storage location passed to mqlCallback from the error handling function. |
| **Description** | This function is a setup routine to handle output and errors. The user passes in a pointer to a function that returns an integer and a storage location for any results. This is done for both normal and error output. |
| **Returns** | **MQLOK** means that callback completed successfully.<br>**MQLERROR** means that callback failed. |
| **Note** | The return values for mqlCallback do not reflect the status of the MQL command for which output is being processed; they reflect the execution status of the mqlCallback function itself. |

| **Function** | **mqlOutputs** |
|---|---|
| **Synopsis** | `int mqlOutputs (void);` |
| **Description** | This function maintains a counter of the number of times the user-supplied outputCallback routine has been called, by recording the number of outputs from mqlCallback. |
| **Returns** | The number of times that mqlCallback called the outputCallback routine. |
| **Note** | The counter is reset each time mqlCallback is called. |

| | |
|---|---|
| **Function** | **mqlErrors** |
| **Synopsis** | `int mqlErrors (void);` |
| **Description** | This function maintains a counter of the number of times the user-supplied errorCallback routine has been called. |
| **Returns** | The numbers of times that mqlCallback called the error callback routine. |
| **Note** | The counter is reset each time mqlCallback is called. |

| | |
|---|---|
| **Function** | **mqlInput** |
| **Synopsis** | `int mqlInput (char * buffer, int size);` |
| **Description** | This function is used by mqlCommand to send the contents of the input buffer to MQL or the Collaboration Server.<br>`char * buffer` is the input buffer.<br>`int size` is a buffer size. |
| **Returns** | The size of the buffer written or MQLERROR for failure. |

| | |
|---|---|
| **Function** | **mqlOutput** |
| **Synopsis** | `int mqlOutput(char * buffer, int limit);` |
| **Description** | This function is used by mqlCallback to receive data from MQL or the Collaboration Server.<br>`char * buffer` is a pointer to the output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | The size of the buffer received or MQLERROR on failure. |

| | |
|---|---|
| **Function** | **mqlOutputLine** |
| **Synopsis** | `char * mqlOutputLine(char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback to receive a line of data from MQL or the Collaboration Server. It breaks on new line.<br>`char * buffer` is a pointer to the output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | The size of the buffer received or MQLERROR on failure. |

| **Function** | mqlError |
|---|---|
| **Synopsis** | `int mqlError(char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback to receive error data from MQL or the Collaboration Server.<br>`char * buffer` is a pointer to the error output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | Number of bytes successfully read or MQLERROR on failure. |

| **Function** | **mqlErrorLine** |
|---|---|
| **Synopsis** | `char * mqlErrorLine(char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback to receive a line of error data from MQL or the Collaboration Server. It breaks on new line.<br>`char * buffer` is a pointer to the error output buffer.<br>`int limit` is a buffer size limit. |
| **Returns** | A line of data or NULL if nothing could be read. |

| **Function** | **mqlErrorMessage** |
|---|---|
| **Synopsis** | `void mqlErrorMessage(MQLBOS_STR_FAR_PTR message, const MQLBOS_STR_FAR_PTR segment);` |
| **Description** | Prints formatted error message to stdout. |
| **Returns** | void |

| **Function** | **mqlCommand** |
|---|---|
| **Synopsis** | `int mqlCommand(MQLBOS_STR_FAR_PTR format, ...);` |
| **Description** | This function writes an MQL command to MQL or the Collaboration Server using printf style syntax. |
| **Returns** | **MQLOK** means that the sent command(s) was successfully invoked.<br>**MQLERROR** means that the sent command(s) could not be invoked. |
| **Note** | Do not put a semi-colon (;) at the end of the format string. The mqlCommand function appends a semi-colon automatically. The extra semi-colon will cause mqlCallback to be invoked a second time, resetting the mqlErrors and mqlOutputs counters.<br>When using mqlCommand without a callback function, output should be checked with the mqlPending command |

| **Function** | **mqlExecute** |
|---|---|
| **Synopsis** | `int mqlExecute(MQLBOS_STR_FAR_PTR format);` |
| **Description** | Sends an MQL command to the MQL or the Collaboration Server for execution. |
| **Returns** | **MQLOK** means that the command executed successfully.<br>**MQLERROR** means that the command execution failed. |

| **Function** | **mqlDisableLog** |
|---|---|
| **Synopsis** | `int mqlDisableLog(void);` |
| **Description** | This function turns off the log function started with `mqlOpenLog`. Use this to prevent your log file from becoming too large, for example, if you are listing 50,000 objects. |
| **Returns** | 0 indicates that the log function has been turned off.<br>-1 indicates that the log function has not been turned off. |

| **Function** | **mqlEnableLog** |
|---|---|
| **Synopsis** | `int mqlEnableLog(void);` |
| **Description** | This function turns on the log function again, after a mqlDisableLog function has been called. |
| **Returns** | 0 indicates that the log function has been turned on.<br>-1 indicates that the log function has not been turned on. |

| **Function** | **mqlWriteToLogFile** |
|---|---|
| **Synopsis** | `int mqlWriteToLogFile(int dataType, MQLBOS_STR_FAR_PTR buffer, int size);` |
| **Description** | This function writes data (i.e., errors) to the log file. |
| **Returns** | **MQLOK** means that data was successfully written to the log file.<br>**MQLERROR** means that data was not written to the log file. |

## Library of MQLIO2 Functions

The MQL*2 functions allow simultaneous connections to multiple MQL or Matrix -mql processes.

| mqlio2 Functions | Description |
|---|---|
| mqlCallback2 | This function is a setup routine to handle MQL or the Collaboration Server output and errors. The user passes in a pointer to a function that returns an integer and a storage location for any results. This is done for both normal and error output. This function takes a session ID as argument. |
| mqlClose2 | Ends an open session by terminating the MQL process or the Collaboration Server connection. This function takes a session ID as argument. |
| mqlCommand2 | Sends an MQL command to MQL or the Collaboration Server, using printf style syntax. This function takes a session ID as argument. |
| mqlEOF2 | Checks for the MQL prompt to determine whether or not the MQL process is ready for input from MQLIO. This function takes a session ID as argument. (This function is not needed for MatrixMQL.) |
| mqlError2 | Used by mqlCallback2 to receive error data from MQL or the Collaboration Server. |

| mqlio2 Functions | Description |
|---|---|
| mqlErrorLine2 | Used by mqlCallback2 to receive a line of error data from MQL or the Collaboration Server. It breaks on new line. This function takes a session ID as argument. |
| mqlExecute2 | Sends an MQL command to the MQL process or the Collaboration Server session for execution. This function takes a session ID as argument. |
| mqlInput2 | Used by mqlCommand2 to send the contents of the input buffer to MQL or the Collaboration Server. This function takes a session ID as argument. |
| mqlOpen2 | Starts a session and establishes communications with MQL or the Collaboration Server. MQLIO uses DDE on Windows or named pipes on UNIX; MatrixMQL uses HTTP/XML. This function takes 3 arguments: directory path, name of program, and bootstrap file. |
| mqlOutput2 | Used by mqlCallback2 to receive data from MQL or the Collaboration Server. This function takes a session ID as argument. |
| mqlOutputLine2 | Used by mqlCallback2 to receive a line of data from MQL or the Collaboration Server. It breaks on new line. This function takes a session ID as argument. |
| mqlPending2 | Checks for any output data that is waiting to be processed. It is useful for commands that generate a large amount of output, because it begins to process the output data as soon as it is available. This function takes a session ID as argument. mqlCallback2 is the better function to use. |

## Synopsis of MQLIO2 Functions

The tables that follow provide syntax and additional notes on the MQLIO2 functions.

| | |
|---|---|
| **Function** | **mqlCallback2** |
| **Synopsis** | `mqlCallback2(SESSION, int (*outputCallback) (void * outData),void * outputData, int (*errorCallback) (void * errData),void * errorData);` |
| **Description** | This function is a setup routine to handle output and errors. The user passes in a pointer to a function that returns an integer and a storage location for any results. This is done for both normal and error output.<br>`SESSION` is the session id which identifies the open connection.<br>`int (*outputCallback) (void * outData)` is a pointer to a user-supplied output handling function that returns an integer and has outData as a parameter.<br>`void * outputData` is an output data storage location passed to mqlCallback from the output handling function.<br>`int (*errorCallback) (void * errData)` is a pointer to a user-supplied error handling function that returns an integer and has errData as a parameter.<br>`void * errorData` is the error data storage location passed to mqlCallback from the error handling function. |
| **Returns** | **MQLOK** means that callback completed successfully.<br>**MQLERROR** means that callback failed. |
| **Note** | The return values for mqlCallback do not reflect the status of the MQL command for which output is being processed; they reflect the execution status of the mqlCallback function itself. |

| | |
|---|---|
| **Function** | **mqlClose2** |
| **Synopsis** | `int mqlClose2(SESSION);` |
| **Description** | This function takes a session id as an argument and issues a quit command to the MQL process that corresponds to the id. It also closes any logfile that MQLIO currently has open.<br>`SESSION` is the session id which identifies the open connection. |
| **Returns** | **MQLOK** means the session was shutdown successfully.<br>**MQLERROR** means the session shutdown encountered a problem. |
| **Note** | Once you have opened a session using mqlOpen2, you must call mqlClose2 to end the session before exiting your program. When using MQLIO, if you exit without calling mqlClose2, the MQL process will remain open, taking up system resources. |

| | |
|---|---|
| **Function** | **mqlCommand2** |
| **Synopsis** | `mqlCommand2(SESSION, MQLBOS_STR_FAR_PTR format, ...);` |
| **Description** | This function writes an MQL command to MQL or the Collaboration Server using printf style syntax.<br>`SESSION` is the session id which identifies the open connection. |
| **Returns** | **MQLOK** means that the sent command(s) was successfully invoked.<br>**MQLERROR** means that the sent command(s) could not be invoked. |
| **Note** | Do not put a semi-colon (;) at the end of the format string. The mqlCommand2 function appends a semi-colon automatically. The extra semi-colon will cause mqlCallback2 to be invoked a second time, resetting the mqlOutput2 counters. When using mqlCommand2 without a callback function, output should be checked with the mqlPending2 command |

| | |
|---|---|
| **Function** | **mqlEOF2** |
| **Synopsis** | `int mqlEOF2(SESSION);` |
| **Description** | For MQLIO only, this function checks for the MQL prompt to determine whether or not the MQL process is ready for input from MQLIO.<br>`SESSION` is the session id which identifies the open connection. |
| **Returns** | **TRUE (1)** means MQL is ready.<br>**FALSE (0)** means MQL is not ready. |

| | |
|---|---|
| **Function** | mqlError2 |
| **Synopsis** | `int mqlError2(SESSION, char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback to receive error data from MQL or the Collaboration Server.<br>`SESSION` is the session id which identifies the open connection.<br>`char * buffer` is a pointer to the error output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | Number of bytes successfully read or MQLERROR on failure. |

| **Function** | **mqlErrorLine2** |
|---|---|
| **Synopsis** | `char * mqlErrorLine2(SESSION, char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback2 to receive a line of error data from MQL or the Collaboration Server. It breaks on new line.<br>`SESSION` is the session id which identifies the open connection.<br>`char * buffer` is a pointer to the error output buffer.<br>`int limit` is a buffer size limit. |
| **Returns** | A line of data or NULL if nothing could be read. |

| **Function** | **mqlExecute2** |
|---|---|
| **Synopsis** | `int mqlExecute2(SESSION, MQLBOS_STR_FAR_PTR format);` |
| **Description** | Sends an MQL command to the MQL or the Collaboration Server.<br>`SESSION` is the session id which identifies the open connection. |
| **Returns** | **MQLOK** means that the command executed successfully.<br>**MQLERROR** means that the command execution failed. |

| **Function** | **mqlInput2** |
|---|---|
| **Synopsis** | `int mqlInput2(SESSION, char * buffer, int size);` |
| **Description** | This function is used by mqlCommand to send the contents of the input buffer to MQL or the Collaboration Server.<br>`SESSION` is the session id which identifies the open connection.<br>`char * buffer` is the input buffer.<br>`int size` is a buffer size. |
| **Returns** | The size of the buffer written or MQLERROR for failure. |

| | |
|---|---|
| **Function** | **mqlOpen2** |
| **Synopsis** | `int mqlOpen2(PATH, PROGRAM, BOOTFILE);` |
| **Description** | This function starts a session and establishes communications with MQL or the Collaboration Server. This function takes three arguments.<br>`PATH` is the directory path to the program with which MQLIO communicates. (MQLIO can communicate with either `mql.exe`, or with `matrix.exe` called with the `-mql` command option.)<br>`PROGRAM` is the name of the program used, that is, either `matrix.exe` or `mql.exe`.<br>`BOOTFILE` indicates the bootstrap file used to open the MQL or Matrix -mql session. |
| **Returns** | Integer identifier for each MQLIO session opened. |

| | |
|---|---|
| **Function** | **mqlOutput2** |
| **Synopsis** | `int mqlOutput2(SESSION, char * buffer, int limit);` |
| **Description** | This function is used by mqlCallback2 to receive data from MQL or the Collaboration Server.<br>`SESSION` is the session id which identifies the open connection.<br>`char * buffer` is a pointer to the output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | The size of the buffer received or MQLERROR on failure. |

| | |
|---|---|
| **Function** | **mqlOutputLine2** |
| **Synopsis** | `char * mqlOutputLine2(SESSION,char * buffer,int limit);` |
| **Description** | This function is used by mqlCallback2 to receive a line of data from MQL or the Collaboration Server. It breaks on new line.<br>`SESSION` is the session id which identifies the open connection.<br>`char * buffer` is a pointer to the output buffer.<br>`int limit` is the buffer size limit. |
| **Returns** | The size of the buffer received or MQLERROR on failure. |

| | |
|---|---|
| **Function** | **mqlPending2** |
| **Synopsis** | `int mqlPending2(SESSION);` |
| **Description** | This function checks for any output data that is waiting to be processed. It is useful for commands that generate a large amount of output, because it begins to process the output data as soon as it is available. mqlCallback2 is the better function to use.<br>`SESSION` is the session id which identifies the open connection. |
| **Returns** | **MQLERRORPENDING (2)** means that error output is pending.<br>**MQLOUTPUTPENDING (1)** means output is pending.<br>**MQLOK (0)** means that no output is pending.<br>**MQLERROR (-1)** means that a time-out has occurred. |

## Terminating a Session

A session is terminated using the `mqlClose` command. `mqlClose` issues a `quit` command to the MQL process or server session started with `mqlOpen`. Any open log files should also be closed using `mqlCloseLog`.

```
    if (mqlCloseLog() == MQLERROR)
    {
       fprintf(stderr, “ERROR: Can not Close MQL Log File\n”);
       exit(-1);
    }
```

```
    if (mqlClose() == MQLERROR)
    {
       fprintf(stderr, “ERROR: Can not Close MQL Interface\n”);
       exit(-1);
    }
```

*For MQLIO only: It is VERY important to call `mqlClose` before exiting the MQLIO application. If the application exits due to an error condition and `mqlClose` is not called, a disconnected MQL process will be left in the process status block. If this occurs on a regular basis, system resource problems will occur.*

`mqlClose` and `mqlCloseLog` have the same possible returns:

- **MQLOK (0)** The session or log file was closed properly.
- **MQLERROR (-1)** The session or log file encountered a problem when attempting to close.

See also *mqlClose* and *mqlCloseLog*.

## Debugging MQLIO Programs

The MQL window can optionally be displayed when running MQLIO by setting the `mqlwindowdisplay` variable in the initialization file (matrix.ini or ematrix.ini). This is used to diagnose and debug mqlio-based programs and applications. The variable can be set as follows:

- When `mqlwindowdisplay=FALSE`, the MQL session appears as a button on the Windows task bar for mqlio, matrix -wizard, matrix -c, mql -c -d, and openedit, but cannot be opened. (Note: use of `mql -c` without '-d' does display the window)
- When `mqlwindowdisplay=TRUE`, the window can be opened for debugging.

## Conversational Strategies

There are many different strategies a programmer can use to retrieve data from MQL or the server. For example, a list of character pointers can be allocated to hold the data from a command and expand as needed.

```
/***************************************************/
#define ALLOCATIONBLOCK 64
/* List. */
typedef struct stringList stringList;

struct stringList
{
   int size;
   int allocated;
   int iter;
   char **base;
};
/***************************************************/
stringList * createStringList(void)

/* Function to create string list.
** input: none.
** output: none.
** returns: stringList
*/
{
   stringList *list = NULL;
   char **base = NULL;
   if (base = (char **)malloc(ALLOCATIONBLOCK*sizeof(char *)))
   {
      if (list = (stringList *)malloc(sizeof(stringList)))
      {
         list->size = 0;
         list->allocated = ALLOCATIONBLOCK;
         list->iter = -1;
         list->base = base;
      }
      else
      {
         if (base) free((void *)base);
      }
   }
   return list;
}
Functions to append to the stringList can also be added.
/*****************************************************/

int appendStringList(stringList * list,char * string)

/* Function to append item to string list.
** input: list.....list pointer.
** input: string.....string pointer.
** output: none.
** returns: 0......item was successfully appended.
** returns: -1.....item could not be appended.
```

```
*/

{

   int item;
   char **newbase = NULL;

      if (list && list->base && string)
   {
      if (list->size >= list->allocated)
      {
         if (newbase = (char **)malloc((list->allocated
+ALLOCATIONBLOCK)*sizeof(char *)))
         {
            memcpy((char *)newbase,(char *)list->base, list-
allocated*sizeof(char *));
            free((void *)list->base);
            list->base = newbase;
            list->allocated += ALLOCATIONBLOCK;
         }
      }
      item = list->size;

      if(list->base[item]=(char *)malloc(strlen(string)+1))
      {
         strcpy(list->base[item],string);
         list->size++;
         return 0;
      }
   }
   return -1;
}

/*****************************************************/
An output function for use with mqlCommand would look like this:
/* stringListOutputCallback.c -- function to handle string list output
** callback. */

/********************************************************/

#include <string.h>
#include "stringListOutputCallback.h"
#include "stringList.h"
#ifdef MQLIO
#include "mqlio.h"
#else
#ifdef EMATRIXMQL
#include "eMatrixMQL.h"
#endif

/********************************************************/

#define LINEBUFFERSIZE 8192

/********************************************************/

int stringListOutputCallback(void * list)

/* Handle string list output callback.
```

```
** input: list......string list.
** output: none.
** returns: MQLOK.........output was successfully handled.
** returns: MQLERROR.......output could not be handled.
*/

{

   char buffer[LINEBUFFERSIZE];                  /* Buffer */
   char *eol;          /* End of line */

   if (mqlOutputLine(buffer,LINEBUFFERSIZE))
   {
      if (eol = strchr(buffer,' \n'))
      *eol = '\0';
         if (strlen(buffer))
            appendStringList((stringList *)list,buffer);
   }
   return MQLOK;

}
```

## Modular Packaging

The use of MQL commands through the MQLIO or MatrixMQL interface leads to easy modularization of functions. Any MQL sequence can be placed in a *wrapper* function for easy re-use. It is recommended that the `mqlClose` function be placed in a function that would be used in place of `exit()`.

```
/**************************************************/
void ExitApplication( int status )
{

   if (mqlCloseLog() == MQLERROR)
   {
      fprintf(stderr,"ERROR: Can Not Close Log File \n");
      exit(status);
   }
   if (mqlClose() == MQLERROR)
   {
      fprintf(stderr,"ERROR: Can Not Close MQL Interface\n");
      exit(status);
   }

   exit(status);
}

In the application, an error condition could then be handled easily.
if (strcmp(objectType, "Assembly") != 0 )
{
   fprintf(stderr, "Can't find an Assembly!\n");
   ExitApplication( NO_ASSEMBLY );

}
```

Another common MQL usage is to test the existence of an object.

```
/* testObject.c -- function to test if an eMatrix object exists.*/

/*******************************************************/
#include <stdio.h>
#include "testObject.h"
#ifdef MQLIO
#include "mqlio.h"
#include "metric.h"
#else
#ifdef EMATRIXMQL
#include "eMatrixMQL.h"
#endif
/*******************************************************/
#define SUCCESS (0)
#define FAILURE (1)
#define ERROR (-1)
#define LINEBUFFERSIZE 8192
int ignoreOutputCallback(void * outData);
int ignoreErrorCallback(void * errData);
/*******************************************************/

int testObject(char * type, char * name, char * revision, char * user)
```

```
/* Test for object in Matrix.
** input: type..........object type.
** input: name..........object name.
** input: revision......object revision.
** output: none.
** returns: SUCCESS.......object already exists.
** returns: FAILURE.......object does not exist.
** returns: ERROR.........error.
*/
{
   if (mqlCommand("print bus \'%s\' \'%s\' \'%s\' select name dump",  type,
name, revision)
         ==  MQLERROR)
   {
      return ERROR;
   }
   if (mqlCallback( ignoreOutputCallback, NULL,  ignoreErrorCallback, NULL)
      == MQLERROR )
   {
      return ERROR;
   }
   /* If mqlErrors returns zero (0) the object exists return success. */
   if (mqlErrors() == 0)
   {
      return SUCCESS;
   }
   else
   {
      return FAILURE;
   }
}
/********************************************************/
int ignoreOutputCallback(void * outData)

/* Function to handle output callback.
** input: none.
** output: none.
** returns: MQLOK..output callback was handled.
** returns: MQLERROR..output callback could not be handled.
*/

{
   char buffer[LINEBUFFERSIZE];                  /* Buffer */

   if (mqlOutputLine(buffer, LINEBUFFERSIZE))
   {
       /* ignore any output */
   }

   return MQLOK;
}
/********************************************************/

int ignoreErrorCallback(void * errData)

/* Function to handle error callback.
** input: none.
** output: none.
** returns: MQLOK..error callback was handled.
** returns: MQLERROR..error callback could not be handled.
*/

{
   char buffer[LINEBUFFERSIZE];                   /* Buffer */

   if (mqlErrorLine(buffer, LINEBUFFERSIZE))
   {
      /* ignore any error output */
   }

   return MQLOK;

}
The function could be used as follows:
int status;

status = testObject("Assembly", "AAAA", "1", "Fred");

switch (status)
{
   case FAILURE :
      createObject("Assembly", "AAAA", "1", "Fred");
      break;

   case SUCCESS :
      fprintf( stderr, "Object: %s %s %s exists\n",
         "Assembly", "AAAA", "1", "Fred");
      break;

   case ERROR :
      #ifdef MQLIO
      fprintf( stderr, "mqlio Failure\n" );
      #else
      #ifdef EMATRIXMQL
      fprintf( stderr, "eMatrixMQL Failure\n" );
      #endif
      break;

   default :
      fprintf( stderr, "Unknown Return Code %d\n", status );
      break;
}
```

## Working with a DLL

Using a DLL is a convenient method of accessing MQL from another application, since you can perform runtime linking with your C or C++ program, or call it externally from another program. There are several DLLs available:

| DLL | Format | Use With | Protocol |
|---|---|---|---|
| mqlio16.dll | 16-bit | MQLIO | Sockets |
| mqlio32.dll | 32-bit | MQLIO | Sockets |
| mqlVB32.dll | 32-bit | MQLIO, Visual Basic 4 | Sockets |
| eMatrixMQL.dll | 32-bit | MatrixMQL | HTTP/XML |

### 16-bit DLL

mqlio16.dll exports its functions using the FAR PASCAL calling convention. This is to provide compatibility with applications such as Visual Basic Version 3.0 or lower, that support the use of external DLLs, and require the FAR PASCAL calling convention.

### 32-bit DLL

Using mqlio32.dll on a PC is similar to using MQLIO on UNIX platforms. However, compiling and linking with the 32-bit DLL should be performed as shown in the examples in the section *Compiling and Linking*.

### Using Visual Basic with mqlio16.dll

Since Visual Basic 3.0 supports only the FAR PASCAL calling convention when talking with DLLs, you must use the 16-bit version of MQLIO with this version of Visual Basic.

To use mqlio16.dll with Visual Basic 3.0, you need to declare the functions before being able to use them within your program. The function declarations need to be placed in the GENERAL section of your form. The following example shows some typical declarations.

```
Declare Function mqlOpen   Lib "mqlio16.dll" () As Integer
Declare Function mqlExecute   Lib "mqlio16.dll" (ByVal mqlcmd$)
As Integer
Declare Function mqlOpenLog    Lib "mqlio16.dll" (ByVal
FileName$) As Integer
Declare Function mqlCloseLog    Lib "mqlio16.dll" () As Integer
Declare Function mqlTimeOut    Lib "mqlio16.dll" (ByVal
Seconds%) As Integer
Declare Function mqlPending    Lib "mqlio16.dll" () As Integer
Declare Function mqlErrors    Lib "mqlio16.dll" () As Integer
Declare Function mqlOutputs    Lib "mqlio16.dll" () As Integer
Declare Function mqlInput    Lib "mqlio16.dll" (ByVal Buffer$,
ByVal Size%) As Integer
Declare Function mqlOutput    Lib "mqlio16.dll" (ByVal Buffer$,
ByVal Limit%) As Integer
Declare Function mqlError    Lib "mqlio16.dll" (ByVal Buffer$,
ByVal Limit%) As Integer
Declare Function mqlClose    Lib "mqlio16.dll" () As Integer
```

After the functions are declared, you can use them in a Visual Basic script. For example:

```
Sub B_Execute_Click ()

Dim strIn$, strLength, strOut$, status
Dim iPending

strIn$ = ""
strIn$ = W_Input.Text
strOut$ = String(8192, " ")
strLength = 8192

status = mqlExecute(strIn$)

iPending = mqlPending()
Select Case iPending
   Case 1
      status = mqlOutput(strOut$, strLength)
      W_Results.Text = strOut$
   Case 2
      status = mqlError(strOut$, strLength)
      W_Results.Text = strOut$
   Case Else
      W_Results.Text = "No data was returned from this query"
End Select

End Sub
```

Because of the way it handles strings, when using any function that uses a pointer to a string buffer (such as `mqlOutput`), Visual Basic requires that space be allocated for the buffer, allowing it to be modified by the DLL.

For example:

```
Sub GetOutputStreamButton_Click ()
   Dim Value, Limit
   Dim OutVal$
   Limit = 8192 ‘Sets the maximum returned bytes
   OutVal$ = String(8192, 0) ‘Initialize a 8K block for the string
   Value = mqlOutput(OutVal$, Limit) ‘Call the DLL function
   MsgBox OutVal$
End Sub
```

## Using Visual Basic with mqlVB32.dll

mqlVB32.dll is for use with Visual Basic 4.0 or higher which uses the STD calling convention. Functions need to be declared, in a manner similar to using the mqlio16.dll.

## Using DLLs with C or C$^{++}$

When using mqlio16.dll or mqlio32.dll with a C or C$^{++}$ program, you can use the `LoadLibrary()` function to access any of the exported functions, or link in the DLL Stub Library (recommended) as long as the appropriate DLL is in the same path as the executable for your program. When statically linking, `mqlio.h` (`eMatrixMQL.h`, if using MatrixMQL) must be *included*, and the .lib files must appear in the project.

## Include Files

- The include file for MatrixMQL is eMatrixMQL.h.
- The include file for MQLIO is mqlio.h

# Compiling and Linking

## Compiling a Custom Application

### To Compile Using Visual C++

1. Create a new project as “Win32 Console Application” in Visual studio.
2. Add the custom *.cpp source code file(s)
3. Link with eMatrixMql.lib. Under Project Settings -> Link tab, add the library to the end of the line “Object/library modules”.
4. Add the directory where your header files are located under “preprocessor, additional include directories”.
5. Under Project Settings -> C/C++ tab, set **Category** drop down list to **Preprocessor**, and add the path to the header file(s)
6. Make sure the eMatrixMql.dll is available to the compiler, which might mean to place it in the default directory

## Distributing a Custom Application

### To run your test application from the client machine

1. Be sure that the full JRE (or JDK) distribution is installed, and that the system environment PATH variable points to the location of the Java binaries, for example:
   ```
   Set PATH=%PATH%;\JDKINSTALLDIR\bin\hotspot
   ```
2. Be sure that all Components needed are on the client computer:
   - the program you made (eg. "test.exe")
   - eMatrixMQL.dll : in the same directory as the above program, or in other location (but be sure to set your PATH env variable to point to it)
   - eMatrixServletRMI.jar : put this anywhere, and be sure to define your CLASSPATH var to point to it, including full path and filename, (set CLASSPATH=/programs/eMatrixServletRMI.jar)
   - You do not need the "eMatrix.dll" kernel on the client machine.

## Compiling and Linking with UNIX

Makefiles will vary slightly on each UNIX platform. The following is an example Makefile for an HP9000 workstation:

```
##########################################################

CC = cc

CFLAGS = -g -Aa -D_HPUX_SOURCE -I/home/matrix/src/mqlBos

LIBS = /home/matrix/hp9000s800/lib/libmqlBos.o -lm

SRCS = example1.c \
       testObject.c \
       errorCallback.c \
       ignoreErrorCallback.c \
       ignoreOutputCallback.c
```

```
OBJS = example1.o \
       testObject.o\
       errorCallback.o \
       ignoreErrorCallback.o\
       ignoreOutputCallback.o

all: example1

example1: $(OBJS)
       cc -o example1 $(LDFLAGS) $(CFLAGS) $(OBJS) $(LIBS)

clean:
       rm -f $(OBJS) example1

#########################################################
```

*Note: The library supplied for the MQLIO or MatrixMQL interface is in object format (.o). Using the -l option will NOT work. Use the full path name instead.*

## Compiling and Linking on Windows

When using any of the MQLIO dll's or the eMatrixMQL.dll, you can link any of the MQLIO/MatrixMQL functions using dllimport, as shown below.

```
#include <windows.h>

#ifdef MatrixMQL
__declspec (dllimport) int mqlSetContext(char *, char *, char *,
char *);
#endif

__declspec (dllimport) int mqlOpen(void);
__declspec (dllimport) int mqlClose(void);
__declspec (dllimport) int mqlExecute(char *);
__declspec (dllimport) int mqlPending(void);
__declspec (dllimport) int mqlOutput(char *,int);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrev
Instance, LPSTR
lpszCmdLine, int nCmdShow)
{
   int nResult, status;
   char * retbuf = "";
   char * host = "WebServer";
   char * userID = "JSmith";
   char * password = "xyzzy";
   char * vault = "Engineering";

#ifdef EMATRIXMQL
mqlSetContext(host, userID, password, vault);
#endif

   nResult = mqlOpen();
   mqlExecute("set context user creator");
   status = mqlPending();
   mqlExecute("list vault");
   mqlOutput (retbuf, 1000);
   MessageBox (NULL, retbuf, "Title", MB_OK);
   mqlClose();

   return 1;
}
```

## Sample Code

Sample code for MQLIO and MatrixMQL is provided on the distributed CDs. The filenames and locations are listed below.

### MQLIO Sample Code

The MQLIO sample code can be found on the Matrix core distributed CD. The following files are included:

- MqlioTest.C
- MqlioTest.h

### MatrixMQL Sample Code

The MatrixMQL sample code can be found on the Studio Customization Toolkit distributed CD. The following files are included:

- eMatrixMQL.C
- eMatrixMQL.h

### Example makefile for Studio Customization Toolkit mql.C

The following is an example using C++ Studio Customization Toolkit classes to create a command line interface to access the database via the collaboration server.

This example is for the HP platform.

```
#
# make.mql - example makefile for Matrix ADK mql.C example.
#
#  Usage : make -f make.mql
#

ADKHOME=/opt/docroot/matrix/cxx
PLATFORM=hp9000s800

CC=/bin/CC
HPFLAGS= -Wl,+s -Wl,+b/usr/lib:/usr/matrix/lib/${PLATFORM}
-lITinimt -lcma
CXXFLAGS=-c -O +eh -I${ADKHOME}/src/include -I${ADKHOME}/src/
include/corba -I${ADKHOME}/src/include/${PLATFORM}
-I${ADKHOME}/src/mqlBos
LDFLAGS=-o mql +eh -L${ADKHOME}/lib/${PLATFORM} ${ADKHOME}/lib/
${PLATFORM}/libclient.o ${HPFLAGS}

all: mql

mql:
   ${CC} ${CXXFLAGS} mql.C
   ${CC} ${CXXFLAGS} MxString.C
   ${CC} ${LDFLAGS} mql.o MxString.o
```

# A Dialog Scripting Language

## Dialog Scripting Language

Programs can be written that can call ENOVIA Live Collaboration dialogs through the use of the `application` command. The application command provides programmers with a graphical user interface alternative to Tk, and one that provides ENOVIA Live Collaboration familiarity. Since the dialogs must be accessible from the launching process, these programs can be:

- An MQL or Tcl program object run either within ENOVIA Live Collaboration (via a tool, method, or trigger) or with the "matrix -mql" command line option. (`matrix -mql -c “execute program NAME;”`).
- An MQL script run with the "matrix -mql" command line option (`matrix -mql -c “run FILE.mql;”`).
- Commands passed by an external program or application that is talking to `matrix -mql` through an MQLIO connection that was opened using the mqlopen2 interface, passing `matrix.exe` as the 'executable' argument.

Refer to *Command Line Execution* for more information on the first two options. MQLIO is documented in *Embedding MQL: MQLIO and MatrixMQL*.

Special considerations must be made when writing code for programs that will be run on the Web. See *Dialog Scripting on the Web* for details.

## Application Command

The `application` command is used by specifying the dialog type and the full specification of a business object, if required. The syntax is:

```
appl DIALOGTYPE [bus OBJECTID] [program PROGRAM_NAME {ARG}];
```

where:

`OBJECTID` is the Type Name Revision of the business object. Most DIALOGTYPEs require that a business object is specified.

`PROGRAM_NAME` allows a program to be launched when the application dialog is dismissed. A dialog is dismissed when a user clicks the Close or Cancel button, a dialog-specific commit button (such as Modify or Checkin), or the X in the upper right corner of the dialog. This “dismiss program” can be used, for example, to allow a wizard sequence to be customized to include an application dialog. Any variables that need to be maintained can be passed to the specified program as arguments. When a program is launched in this manner, the `INVOCATION` macro is set to “program.” (See the Wizard Command in the programming reference section of the *MQL Guide*.)

`ARG` is zero or more space-delimited strings that are passed to the program as arguments.

The `program` clause with all its arguments must be at the end of the `appl` command. Everything after the program name is assumed to be arguments to the program.

`DIALOGTYPE` can be:

| | | |
|---|---|---|
| attributes | basic | change |
| checkin | checkout | clone |
| details**** | edit*** | files |
| find* | formats | forms |
| history | icons**** | indented |
| indentedtable | methods | navigator |
| notify** | paginatedtable | print*** |
| reassign | relationshipattributes | relationshiphistory |
| revision | revisions | route |
| sendmail* | star | states |
| view*** | wizard | . |

A few points need to be made about some of the DIALOGTYPEs:

- * Does not require that a business object is specified.
- ** Notify is not a dialog type, but is used when a business object has been modified to update the display of the object in whatever windows it resides in. See *Updating Browsers*.
- *** A format and/or a file can also be specified when `appl` is used with `edit`, `print` or `view`, as follows:

  ```
  appl | edit  | bus OBJECTID [format FORMAT [file FILENAME]];
       | print |
       | view  |
  ```

  When any of these forms of the `appl` command is used, an ENOVIA Live Collaboration dialog is not displayed, but the format’s edit, view or print program is launched.

- ****The details and icons dialog types let you add objects or sets to ENOVIA Live Collaboration browsers in Details or Icon mode. See *Adding an Object or a Set to Specified Browser* for additional options.

*DIALOG values are CASE SENSITIVE regardless of the system casesensitive setting. Also, they CANNOT be abbreviated.*

For example, the following command could be used to call the attributes dialog for whatever object the program passes:

```
appl attributes bus OBJECTID
```

## Additional Functionality

The `application` command provides several additional functions:

### Adding an Object or a Set to Specified Browser

An object or set can be added to an ENOVIA Live Collaboration primary browser, in either Icons or Details mode. In addition, several options exist for the programmer to specify in which primary browser they should be added.

To add an object or set to a primary browser, the following commands are available:

```
    appl[ication] | icons   | | bus OBJECTID |[dialog WHICHDIALOG];
                  | details|  | set SETNAME   |
```

where:

`OBJECTID` is the `OID` or Type Name Revision of a business object.

`SETNAME` is the name of a valid set in the context user's workspace.

`WHICHDIALOG` indicates the primary browser to which object(s) will be added. It can have one of the following values:

- **`first`**—the first primary browser that was opened in the session
- **`all`**—all existing primary browsers
- **`DIALOGID`**— the primary browser that was opened through a previous application scripting command and returned the specified DIALOGID to the calling program
- **`new`**—meaning a new primary browser will be opened and loaded with the specified object(s)

If the `dialog WHICHDIALOG` clause is not used, the specified object or set will use the default and be loaded into a `new` primary browser. When a particular dialog is specified, its mode (icons/details) can be changed with this command.

For example, to add `MySet` to all primary browsers and switch them all to Details mode, use:

```
appl details set MySet dialog all;
```

To add the Release Notes Manual object to a new primary browser use:

```
appl icons bus Manual “Release Notes” 1 dialog;
```

### Specifying a View and/or a Role

The appl navigator commands provide the option to specify a view, allowing a set of visuals (tips, filters, cues, and a default table) to be enabled. You can also specify a role, either with or without a view (if with a view, the role must be part of the view, and the current user must belong to the role). If a role is specified without a view, the View menu in the new window to be initialized shows the visuals owned by the named role rather than those owned by the current context user. The default value for rolename is "Personal." The following commands can optionally include a view and/or a role specification:

```
appl indented bus OBJECTID [view VIEWNAME] [role ROLENAME]
appl indentedtable bus OBJECTID [view VIEWNAME] [role ROLENAME]
appl star bus OBJECTID [view VIEWNAME] [role ROLENAME]
appl details bus T N R role [view VIEWNAME][ROLENAME]
appl icons set N role [ROLENAME]

appl navigator bus OBJECTID [view VIEWNAME] [role ROLENAME]
```

(This last one looks at the user's "browse by" preference to display the preferred relationship browser.)

*Role-based visuals with appl commands are supported with Matrix Web Navigator only. (Desktop Navigator is not supported.)*

Where:

`OBJECTID` is the `OID` or Type Name Revision of the business object.

`VIEWNAME` is the name of a pre-defined view. Performance is optimized since the view is applied before the object is expanded.

`ROLE` is the name of a pre-defined role.

For example::

```
    appl details bus Manual Spec 1.0 view StandardView role
    Employee;
```

### Updating Browsers

When modifications are made to an object from either the Attributes or Change dialogs, or from a form, all visuals active in any existing Primary or Navigator browsers that contain that object are updated automatically. Object state changes are also tracked and updated. In addition, when these types of changes are made via a program (method, trigger, or wizard), the following command can be incorporated into that program to reapply the visuals:

```
appl notify bus OBJECTID modified;
```

Similarly, the following command can be used to update browsers to reflect newly-created objects:

```
appl notify bus OBJECTID created;
```

The notification of such change is sent only to the dialogs/browsers that are controlled by the SAME process as the one doing the notifying. For example, if you launch one Matrix Navigator session (Session1), and another Matrix Navigator session (Session2), the `appl notify` command typed into Session2 would NOT affect any of the dialogs/browsers of Session1.

### Checkin Parameters

It is possible to hide some of the check boxes on the checkin dialog and set the value to a specific value. For example:

```
appl checkin flags FLAG_NAME;
```

`FLAG_NAME` can be any of the following:

- `append` or `replace`
- `unlock` or `notunlock`
- `delete` or `notdelete`

### Checking Out Sets

Entire sets can be checked out with the following command:

```
appl checkout set NAME;
```

The checkout dialog is displayed, listing the objects contained in the named set. The user can then check out the files contained in all objects in the set.

### Displaying Forms

Specific forms can be displayed for the passed business object using the following command:

```
appl forms bus ${OBJECT} [FORM];
```

If a FORM is specified, it is displayed against the object passed. Without a named form, the form chooser is presented to the user for a selection.

### Reassigning a Set

The contents of an entire set can be reassigned to a single user by popping up the reassign dialog:

```
appl reassign set NAME;
```

The reassign dialog is displayed listing all objects contained in the set.

### Displaying Relationship Attributes and History

The attributes dialog of a connection can be displayed using the following application command:

```
appl relationshipattributes CONNECTIONID;
```

where `CONNECTIONID` is the identification number of the connection.

This is helpful when you are programmatically connecting objects, and you want to get user input for the attribute values.

The history dialog of a connection can be displayed using the following application command:

```
appl relationshiphistory CONNECTIONID;
```

where `CONNECTIONID` is the identification number of the connection.

### Sending a Set

Entire sets can be sent via IconMail using the following command:

```
appl sendmail set NAME
```

The IconMail dialog is displayed with all objects contained in the named set listed. The message is sent once, but contains all members of the named set.

## Application Help

Help on the syntax of the `application` command is available, but *not* through the MQL `help application` command. Since `application` is not really an MQL command, only being available when ENOVIA Live Collaboration is run in MQL mode, you must be in this mode to get help as well.

To get help, start ENOVIA Live Collaboration with the `-mql` argument. On Windows machines, this means modifying the properties of the matrix.exe shortcut. Add `-mql` to the target area of the shortcut tab. Start the application and then type:

```
applic help;
```

## Using the Application Command

When the `application` command is passed, the dialog is displayed on the user's screen and the dialog ID# is returned to the calling process. By default, dialogs are invoked non-modally, which means that the calling program continues to operate whether or not it received input from the user. This might be useful, for example, if the calling program wanted to give the user the opportunity to update attributes by presenting the attributes dialog, without needing to know what changes the user made. In this case the calling program could use `appl attributes bus OBJECTID` to cause the attributes dialog to pop up. Control would return immediately to the calling program, which could continue, ignorant of whether or not the user edited the attributes or just cancelled the attributes dialog.

### Wait and Modal Clauses

In other cases, it is important for a calling program to know what modifications the user may have made in ENOVIA Live Collaboration through an `appl` dialog. In these cases, the calling program should follow the dialog invocation with a call to `appl wait`. This command causes execution of the program to pause until the user has disposed of the specified dialog (or all dialogs, for `appl wait 0`). Once the user does that, execution of the program resumes at the statement just following the `appl wait` command. Note that the appl wait command does not freeze ENOVIA Live Collaboration. A user can still use ENOVIA Live Collaboration even though they have not disposed of the invoked dialog.

The `wait` keyword causes the calling program to be suspended until the specified dialog has been closed. The syntax is:

```
appl wait DIALOGID;
```

where `DIALOGID` is the number output by the `appl DIALOGTYPE` command which popped up the dialog that you want to wait for.

To have the ENOVIA Live Collaboration application freeze (be unusable) until the user has finished with the invoked dialog, the calling program should follow the dialog invocation with a call to `appl modal`. For this reason, dialogs that have child windows should not be made modal.

When the `modal` clause is used, the specified dialog becomes modal. Modal dialogs stay in the foreground until they are closed.

```
appl modal DIALOGID;
```

where `DIALOGID` is the number output by the `appl DIALOGTYPE` command which popped up the dialog that you want to be modal.

Note that `appl modal` is intended to be used only with a subset of the `appl` commands. Specifically, it should NOT be used with any dialogs which themselves can invoke additional dialogs. For example, the Find dialog has a button to get to the Query dialog and the Formats dialog has menus for file access, edit, view, etc.; these dialogs should not be invoked in a modal state.

*`Appl modal` is intended to be used ONLY with standalone dialogs. It is not supported on ENOVIA Live Collaboration Applet.*

The following table identifies which dialogs can (OK) or cannot (NO) be used in conjunction with `appl modal`:

| | |
|---|---|
| `appl attributes bus Assemby 12345 1` | OK |
| `appl basic bus Assemby 12345 1` | OK |
| `appl change bus Assemby 12345 1` | NO |
| `appl checkin bus Assemby 12345 1` | OK |
| `appl checkout bus Assemby 12345 1` | OK |
| `appl checkout set Assemblies` | OK |
| `appl clone bus Assemby 12345 1` | OK |
| appl details `bus Assemby 12345 1` | OK |
| `appl edit bus Assemby 12345 1` | OK |
| `appl edit bus Assemby 12345 1 format Word` | OK |
| `appl edit bus Assemby 12345 1 format Word file nospace.doc` | OK |
| `appl files bus Assemby 12345 1` | OK |
| `appl find` | NO |
| `appl formats bus Assemby 12345 1` | NO |
| `appl forms bus Assemby 12345 1` | NO |
| `appl history bus Assemby 12345 1` | NO |
| `appl icons bus Assemby 12345 1` | OK |
| `appl indented bus Assemby 12345 1` | NO |
| `appl indentedtable bus Assemby 12345 1` | NO |
| `appl methods bus Assemby 12345 1` | NO |
| `appl navigator bus Assemby 12345 1` | OK |
| `appl print bus Assemby 12345 1` | OK |
| `appl print bus Assemby 12345 1 format Word` | OK |

| | |
|---|---|
| `appl print bus Assemby 12345 1 format Word file nospace.doc` | OK |
| `appl reassign bus Assemby 12345 1` | OK |
| `appl reassign set app` | OK |
| `appl relationshipattributes connection 19.24.5.16` | OK |
| `appl revision bus Assemby 12345 1` | OK |
| `appl revisions bus Assemby 12345 1` | NO |
| `appl route bus Assemby 12345 1` | OK |
| `appl sendmail bus Assemby 12345 1` | OK |
| `appl sendmail set app` | OK |
| `appl star bus Assemby 12345 1` | NO |
| `appl states bus Assemby 12345 1` | NO |
| `appl view bus Assemby 12345 1` | OK |
| `appl view bus Assemby 12345 1 format Word` | OK |
| `appl view bus Assemby 12345 1 format Word file nospace.doc` | OK |

*Appl modal is not supported for the Web version of Matrix Navigator.*

The difference between `appl wait` and `appl modal` is only evident when running a Tcl/MQL script in an interactive `matrix -mql` session, in which case `appl wait` allows the user to continue to type ahead in the `matrix -mql` text window; `appl modal` does not.

## Output

Most dialogs output only the ID to the calling program, and this is done as soon as the window is opened. User input may affect the ENOVIA Live Collaboration database, but is generally not explicitly fed to the calling program. For example, if the attributes dialog is popped up and the user changes a value and clicks the edit button, the database is updated, but the new values are not automatically passed to the program. If the program needs to know the new values, it still has the object ID, and can execute its own MQL commands to retrieve them. However, with dialogs like change, clone, and revision, the program could lose the handle to the object if the type, name or revision is changed. In these cases, the object type, name, and revision are output to the program.

# B Studio Customization Toolkit

## ENOVIA Live Collaboration Database Package Methods

The following samples include a number of sections of code that illustrate server-side features added for Version 9.x.

### Set Select

The Java Studio Customization Toolkit class matrix.db.Set includes a `select` method that accepts a context object and select statements and returns the selected data for all objects in the set.

This code demonstrates running select clauses on a set or list.

```
// Instantiate the BusinessObject
      Set _object =  new Set("AllObjects");
      _object.open(_context);

      StringList selectStmts = new StringList(4);
```

```
              selectStmts.addElement("type");
              selectStmts.addElement("name");
              selectStmts.addElement("id");

              BusinessObjectWithSelectList _objectSelectList =
        _object.select(_context, selectStmts);
              int size = _objectSelectList.size();
              String ODI[] = new String[size];
              BusinessObjectWithSelectItr busItr = new
        BusinessObjectWithSelectItr(_objectSelectList);
              //StringItr strItr = new StringItr(selectStmts);
```

## List Select

The Java Studio Customization Toolkit class matrix.db.Context includes a `getSelectBusinessObjectData` method, which accepts a stringList of object IDs and a StringList of select statements (name, owner, etc.). The method returns a BusinessObjectWithSelectList, which provides details about all the business objects.

```
        // Getting Object IDs
              int i =0;
              while (busItr.next())
              {
                   BusinessObjectWithSelect bus = new
        BusinessObjectWithSelect(busItr.obj());
                   ODI[i]=(String)bus.getSelectData("id");
                   System.out.println(ODI[i]);
                   i++;

              }

         // getSelectBusinessObjectData()
                    BusinessObjectWithSelectList _busSel=
        _context.getSelectBusinessObjectData(ODI,selectStmts);
                    BusinessObjectWithSelectItr busItr1 = new
        BusinessObjectWithSelectItr(_objectSelectList);
                    StringItr strItr = new StringItr(selectStmts);
                    while (busItr1.next())
                    {
                         BusinessObjectWithSelect bus = new
        BusinessObjectWithSelect(busItr1.obj());
                         while (strItr.next())
                          {
                          System.out.print("  ");
                          System.out.print(strItr.obj());
                          System.out.print(" - ");

        System.out.println(bus.getSelectData(strItr.obj()));

                        }
                        strItr.reset();

        System.out.println("----------------------------");
```

```
            }
          RelationshipWithSelectList relList =
_context.getSelectRelationshipData(ODI,selectStmts);

            RelationshipWithSelectItr relItr = new
RelationshipWithSelectItr(relList);
            StringItr strItr2 = new StringItr(selectStmts);
            while (relItr.next())
            {
                RelationshipWithSelect relsel = relItr.obj();

                Hashtable hash1 = relsel.getRelationshipData();
                Hashtable hash2 = relsel.getTargetData();

               // System.out.print(relsel.getLevel()+"
"+hash1.get("name"));

                while (strItr2.next())
                {

                    System.out.print("
"+hash2.get(strItr2.obj()));
                }
                System.out.println();
                strItr.reset();
            }

//
----------------------------------------------------------------
----
          _object.close(_context);
      }
      catch (MatrixException e)
      {
        e.printStackTrace();
      }
    }
```

## Expand Select

The Java Studio Customization Toolkit class matrix.db.BusinessObject includes an `expandSelect` method that lets you specify one or more select clauses to be used on a business object expand. The method returns the relationships of the business object.

The following demonstrates how to expand the business object. It gives all relationships to the level you specify.

```
// Instantiate the BusinessObject.
       StringList selectBusStmts = new StringList(4);
       selectBusStmts.addElement("type");
       selectBusStmts.addElement("name");
       selectBusStmts.addElement("owner");
       selectBusStmts.addElement("vault");
       StringList selectRelStmts = new StringList(3);
       selectRelStmts.addElement("name");
       selectRelStmts.addElement("to");
       selectRelStmts.addElement("from");
       short sh = 2;

       BusinessObject _object = new
BusinessObject("MX-TYPE-1","TEST6","1","MX-VAULT-1");
       _object.open(_context);

// Getting the expansion
 ExpansionWithSelect expan =  _object.expandSelect
(_context,"*","*",selectBusStmts,selectRelStmts,true,true,sh);
//
-------------------------------------------------------------
//  _object.expandSelect(_context, - Java context object
//  "*",                           - relationship Pattern
//  "*",                           - type Pattern
//  selectBusStmts,               - selects for Business Objects
//  selectRelStmts,                - selects for Relationships
//  true,                          - get To relationships
//  true,                          - get From relationships
//  sh);                           - recursion level (0 = all)
//
-------------------------------------------------------------

// Get Task
    ClientTaskList  ctl =  expan.getTasks();
    ClientTaskItr ctlItr = new ClientTaskItr(ctl);

    while (ctlItr.next())
           {
               System.out.print("  ");
               System.out.print(ctlItr.obj());
               System.out.print(" - ");
              System.out.println((ctlItr.obj()).getTaskData());
           }

// Getting Root Object
       BusinessObject _objRoot = expan.getRoot();
       _objRoot.open(_context);
       StringItr strItr = new StringItr(selectBusStmts);
       BusinessObjectWithSelect bus = _objRoot.select(_context,
selectBusStmts);
       System.out.println("The Root Business Object for this
Relation is :");
           while (strItr.next())
           {
               System.out.print("  ");
               System.out.print(strItr.obj());
               System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
           }
           strItr.reset();
      _objRoot.close(_context);

// Getting Root Data
   Hashtable rtHash = expan.getRootData();
   Enumeration rtEnum = rtHash.keys();

System.out.println("------------------------------------------
----");
   System.out.println("The Root Data  is");

         while(rtEnum.hasMoreElements()){
             String str = (String)rtEnum.nextElement();
             System.out.println(str + " : " + rtHash.get(str));
         }

// Getting Relationships.
       RelationshipWithSelectList _relSelectList =
expan.getRelationships();

       RelationshipWithSelectItr relItr = new
RelationshipWithSelectItr(_relSelectList);

          while (relItr.next())
          {

          RelationshipWithSelect relSelect =relItr.obj();
          Hashtable relHash =relSelect.getRelationshipData();
          Hashtable tarHash =relSelect.getTargetData();
          Enumeration enumRel = relHash.keys();
          Enumeration enumTar = tarHash.keys();

 // Relationship Details
         System.out.println("The Relationship is");

         while(enumRel.hasMoreElements()){
         String str = (String)enumRel.nextElement();
         System.out.println(str + " : " + relHash.get(str));
         }

 // Target Details
         System.out.println("The Target BusinessObject is ");

         while(enumTar.hasMoreElements()){
         String strs = (String)enumTar.nextElement();
         System.out.println(strs + " : "+ tarHash.get(strs));
```

```
                }


    System.out.println("-----------------------------------------
    -----");
           }

          _object.close(_context);

          }
          catch (MatrixException e)
          {
          e.printStackTrace();

       //      End of Relationship
```

## Set Subset Select and Select Count

The Java Studio Customization Toolkit class matrix.db.Set includes a `count` method that accepts a context object as input and returns the number of objects the set contains. A `subsetSelect` method is also provided that returns selected data for a portion of a given set. For example, if a set has 100 objects, the user can retrieve the selected data for any n number of objects starting from any position.

The following prints the count of objects in a set and lists all objects in the set.

```
    //     Instantiate the BusinessObject
           Set _object =  new Set("AllObjects");
           _object.open(_context);
     //
    ---------------------------------------------------------------
    --
     //    public long count(Context context) throws MatrixException
     //
    ---------------------------------------------------------------
    --
           long l = _object.count(_context);
           System.out.println("The Count of the Set is :"+l);

           StringList selectStmts = new StringList(4);
           selectStmts.addElement("type");
           selectStmts.addElement("name");
           selectStmts.addElement("revision");
           selectStmts.addElement("id");
           selectStmts.addElement("owner");
           selectStmts.addElement("policy");
           selectStmts.addElement("vault");
           selectStmts.addElement("modified");

           BusinessObjectWithSelectList _busSel=
    _object.subsetSelect(_context,24,5,selectStmts);
           BusinessObjectWithSelectItr busItr = new
    BusinessObjectWithSelectItr(_busSel);
```

```
              StringItr strItr = new StringItr(selectStmts);
              while (busItr.next())
              {
                   BusinessObjectWithSelect bus = new
       BusinessObjectWithSelect(busItr.obj());
                   while (strItr.next())
                    {
                    System.out.print("  ");
                    System.out.print(strItr.obj());
                    System.out.print(" - ");

       System.out.println(bus.getSelectData(strItr.obj()));

                  }
                  strItr.reset();
                  System.out.println("-----------------------------");
              }
       //
       ---------------------------------------------------------------
       -----
             _object.close(_context);
        }
        catch (MatrixException e)
        {
          e.printStackTrace();
        }
```

## Print Select

The Java Studio Customization Toolkit includes a `select` method on a matrix.db.BusinessObject that lets you specify one or more select clauses to be used on a business object print. The method returns a list of business object IDs with a name/value pair for each selected item.

```
    public PrintBusinessObject ()
       {
            // Variables.
            Context _context = null;

         try
         {

           _context = new Context(":bos","rheia");

                _context.setUser("creator");
                _context.setPassword("");
                _context.setVault("");
                _context.connect();

           // Instantiate the BusinessObject.
```

```
            BusinessObject _object = new
BusinessObject("MX-TYPE-1", "TEST8", "1", "MX-VAULT-3");

            _object.open(_context);

            StringList selectStmts = new StringList(8);
            selectStmts.addElement("type");
            selectStmts.addElement("name");
            selectStmts.addElement("revision");
            selectStmts.addElement("id");
            selectStmts.addElement("owner");
            selectStmts.addElement("policy");
            selectStmts.addElement("vault");
            selectStmts.addElement("modified");

            StringItr strItr = new StringItr(selectStmts);
            BusinessObjectWithSelect bus =
_object.select(_context, selectStmts);
                while (strItr.next())
                {
                    System.out.print("  ");
                    System.out.print(strItr.obj());
                    System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
                }
                strItr.reset();

            _object.close(_context);
      }
      catch (MatrixException e)
      {
        e.printStackTrace();
      }
   }
```

## Query Select

The Java Studio Customization Toolkit class matrix.db.Query includes a `select` method, which returns the selected data for the result of a query. It assumes that a valid query exists in the database.

```
// Instantiate the Query.
            Query _object = new Query("z");

            _object.open(_context);

            StringList selectStmts = new StringList(3);

            selectStmts.addElement("type");
            selectStmts.addElement("name");
            selectStmts.addElement("revision");
```

```
          BusinessObjectWithSelectList _objectSelectList = new
BusinessObjectWithSelectList();

           _objectSelectList = _object.select(_context,
selectStmts);

           BusinessObjectWithSelectItr busItr = new
BusinessObjectWithSelectItr(_objectSelectList);
           StringItr strItr = new StringItr(selectStmts);
           int i = 1;
           while (busItr.next())
           {
               BusinessObjectWithSelect bus = new
BusinessObjectWithSelect(busItr.obj());

       //To get the S.No and Id for the printed resulting
Name-Value pair
               System.out.println("(" + i++ + ")" +
bus.getObjectId());
               while (strItr.next())
               {
                   System.out.print("  ");
                   System.out.print(strItr.obj());
                   System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
               }
               strItr.reset();
           }

           _object.close(_context);
```

## Save/Load Structured Sets

A structured set is a collection of business objects and relationships that result from a expand operation. Structured sets are stored as persistent objects in the database. Relative levels are maintained. The Java Studio Customization Toolkit class matrix.db.Structure contains a number of methods. This sample loads an existing structure object and saves into another structure.

Instructions:

```
Create a Structure object from MQL using the following command:
Expand bus "Type" "Name" "Revision" recurse to all
structure "StructureName";
Give this created structure name in the statement:
Structure _object = new Structure("structureName");
// Instantiate the Structure.
       Structure _object = new Structure("struct3");


       _object.open(_context);
       Visuals empty = new Visuals();

       Expansion objectExpand = _object.loadStructure(_context,
empty);

       _object.close(_context);

       Structure _objectNew = new Structure("struct3new");
       _objectNew.open(_context);

       // Build copy of _objectNew structure and save it.
       String topNodeId = objectExpand.getRoot().getObjectId();
       ExpandRelationshipList relList =
objectExpand.getRelationships();
       StructureList strList = new
StructureList(relList.size());
       ExpandRelationshipItr relItr = new
ExpandRelationshipItr(relList);

       int i = 0;
       while (relItr.next())
       {
           String name = Integer.toString(i++);
           Structure tmp = new Structure(name);
           tmp.setLevel(relItr.obj().getLevel());
           System.out.println(tmp.getLevel());
           tmp.setExpanded(true);
           System.out.println(tmp.getExpanded());

           BusinessObject to = relItr.obj().getTo();
           BusinessObject from = relItr.obj().getFrom();
           BusinessObject origin = relItr.obj().getOrigin();

           if (to == origin)
             tmp.setTargetId(from.getObjectId());
           else
             tmp.setTargetId(to.getObjectId());

           tmp.setRelationshipId(relItr.obj().getName());

           strList.addElement(tmp);
       }
       _objectNew.saveStructure(_context,topNodeId,strList);
 }
```

## Getting Data Type Information

These get functions, added for version 9.5.3.0, let programmers obtain datatype information so an application can perform the appropriate formatting on the data.

```
Attribute.java
/**
* Get a data type for this attribute
*
* @return int that is the data type of this attribute
* @since ADK 9.5.3.0
*/
public short getType()

BusinessObjectWithSelect.java
/**
* Get a data type
*
* @return data type for the key
* @since ADK 9.5.3.0
*/
public short getSelectDataType(String key)

RelationshipWithSelect.java
/**
* Get a data type
*
* @return int that is the data type for the key of
target data
* @since ADK 9.5.3.0
*/
public short getTargetSelectDataType(String key)
/**
* Get a data type
*
* @return data type for the key
* @since ADK 9.5.3.0
*/
public short getSelectDataType(String key)

ExpansionWithSelect.java
/**
* Get a root data type
*
* @return int that is the data type for the key
* @since ADK 9.5.3.0
*/
public short getRootSelectDataType(String key)
```

## Sample 1: BusinessObject Class

The following program (BusinessObjectDb.java) illustrates the use of various methods in the businessobject class.

The BusinessObjectDb.java program needs the following arguments to be passed:

```
hostname, server_name, Matrix user_name, BusinessObject_Type,
BusinessObject_Name, BusinessObject_Revision, Format,
error_logfile
```

### Comments List

The following list of comments from the code is included to provide easy access to specific points in the sample program:

// Set session context using the server and ENOVIA Live Collaboration user names

// Construct a new business object

// Get the business object owner

// Construct a new business object

// Get the business object policy

// Get the last modified date for the business object

// Get the created date

// Get the type name

// Get the revision

// Get the default format for the business object

// Get a list of states for the business object

// Get the getTorelationship for the given business object

// Get the getFromRelationship for the given business object

// Return a list of formats for the given business object type

// Return a list of files for the given format

## Code sample - BusinessObjectDb.java

```
//
/*  Desc of program: This program illustrates the use of various
methods in the BusinessObject class.
*/
import matrix.db.*;
import matrix.util.*;
import matrix.db.File;
import java.io.*;
import java.lang.*;
import java.util.*;
import java.awt.Frame;

public class BusinessObjectDb extends Frame {

    Context _context;
    String _host;
    String _server;
    String _user;
    String _type;
    String _name;
    String _rev;
    String _format;
    String _errfile;

public BusinessObjectDb (String host, String server, String
user, String type, String name, String rev,String format, String
err_file)
{
    _host = host;
    _server = server;
    _user = user;
    _type = type;
    _name = name;
    _rev = rev;
    _format = format;
    _errfile = err_File;
}
    public static void main( String args[])
    {
new BusinessObjectDb( args[0], args[1], args[2], args[3],
args[4], args[5], args[6], args[7]);
    }
```

// Set session context using the server and ENOVIA Live Collaboration user names

```
    public boolean setContext()
    {
       try
       {
          _context = new Context(_server,_host);
          _context.setUser(_user);
```

```
                          _context.setPassword("");
                          _context.setVault("");
                          _context.connect();
                          _context.disconnect();
                       }

                    catch (MatrixException e)
                    {
                       _context = null;
                       System.out.println("Can't set context");

                       ListItr itr = new ListItr(e.getMessages());
                       while(itr.next())
                       {
                          System.out.println((String) itr.value());
                       }
                    return false;
                    }
                    return true;
                    }
                 private void getBusinessObject()
                 {
                    try
                    {
                       _context.connect();

// Construct a new business object
                       BusinessObject _object = new
                 BusinessObject(_type,_name,_rev,_context.getVault.getName());
                       _object.open(_context);

                       AttributeList list = _object.getAttributes();
                       AttributeItr itr = new AttributeItr(list);
                       while (itr.next())
                       {
                          Attribute attribute = itr.obj();
                          System.out.println(“attribute name: “ +
                 attribute.getName());
                       }

                       System.out.println(“description: “ +
                 _object.getDescription());

                       String vault = _object.getVault();
                       System.out.println(“vault: “ + vault);

// Get the business object owner
                       User user  = _object.getOwner();
                       System.out.println(“owner: “ + user.getName());
```

```
// Get the business object policy
                              Policy policy = _object.getPolicy();
                              System.out.println(“policy: “ + policy.getName());

// Get the last modified date for the business object
                              System.out.println(“modified: “ + _object.getModified());

// Get the created date
                              System.out.println(“created: “ + _object.getCreated());

// Get the name of the business object icon
                              System.out.println(“image name: “ +
                        _object.getImageName());

// Get the type name
                              System.out.println(“type: “ + _object.getTypeName());

// Get the revision
                              System.out.println(“rev: “ + _object.getRevision());

// Get the default format for the business object
                              System.out.println(“format: “ +
                        _object.getDefaultFormat(_context));

// Get a list of states for the business object
                              StateList stateList = _object.getStates(_context);
                              StateItr stateItr = new StateItr(stateList);
                              while (stateItr.next())
                              {
                                 State state = stateItr.obj();
                                 System.out.println(“state: “ + state.getName());
                              }
// Get the getTorelationship for the given business object
                              RelationshipList relToList =
                        _object.getToRelationship(_context);
                              RelationshipItr relToItr = new
                        RelationshipItr(relToList);
                              while (relToItr.next())
                              {
                                 Relationship relTo = relToItr.obj();
                                 System.out.println(“relationship to: “ +
                        relTo.getName());
                              }
// Get the getFromRelationship for the given business object
                              RelationshipList relFromList =
                        _object.getFromRelationship(_context);
                              RelationshipItr relFromItr = new
                        RelationshipItr(relFromList);
                              while (relFromItr.next())
                              {
                                 Relationship relFrom = relFromItr.obj();
```

```
         System.out.println(“relationship from: “ +
relFrom.getName());
      }
```

// Return a list of formats for the given business object type

```
      FormatList formatList = _object.getFormats(_context);
      FormatItr formatItr = new FormatItr(formatList);
      while (formatItr.next())
      {
         Format format = formatItr.obj();
         System.out.println(“format: “ + format.getName());
      }
```

// Return a list of files for the given format

```
      FileList fileList = _object.getFiles(_context, _format);
      FileItr fileItr = new FileItr(fileList);
      while (fileItr.next())
      {
         File fileObject = fileItr.obj();
         System.out.println(“files: “ + fileObject.getName());
      }

      _object.close(_context);
         _context.disconnect();
   }
      catch(MatrixException e)
   {
         _context = null;
         ListItr itr = new ListItr(e.getMessages());
         while(itr.next())
         {
                     log_file(file1,(String) itr.value());
         }
      }
      if (itr.value() instanceof ErrorMessage)

System.out.println(((ErrorMessage)itr.value()).getMessage() );
      {
         System.out.println("Business Object  .....done
successfully");
      }
   }
}
```

## Sample 2: MQL Command Execution

The following program (MqlExecuteCmd.java) executes a simple MQL command and captures the result through the getResult() and the errors through the getError() methods. The mql statement used within the command line argument should be enclosed in quotes.

The MqlExecuteCmd.java program needs the following arguments to be passed:

```
hostname, server_name, business_administrator user_name, mql
statement, error_logfile
```

### Comments List

The following list of comments from the code is included to provide easy access to specific points in the sample program:

// Set session context using the server and ENOVIA Live Collaboration user names

// Construct a new MQLCommand object

// mqlcommand executed

// mqlcommand results fetched

// mqlcommand errors fetched

### Code sample - MqlExecuteCmd.java

```
//
/* Desc of the program: This program executes a simple MQL
command and captures the result through the getResult() and the
errors through the getError() methods. The mql statement used
within the command line argument should be enclosed in quotes.
*/

import java.awt.*;
import matrix.db.*;
import matrix.util.*;
import java.io.*;

public class MqlExecuteCmd extends Frame
{
   Context _context;
   String _host;
   String _server;
   String _user;
   String _command;
   String _errfile;

public MqlExecuteCmd (String host, String server, String user,
String cmd, String err_file)
   {
      _host = host;
      _server = server;
      _user = user;
      _command = cmd;
      _errfile = err_file;
   }

   public static void main( String args[])
   {
      new MqlExecuteCmd( args[0], args[1], args[2], args[3],
args[4], args[5]);
   }
```

// Set session context using the server and ENOVIA Live Collaboration user names

```
   public boolean setContext()
   {
      try
      {
         _context = new Context(_server,_host);
         _context.setUser(_user);
         _context.setPassword("");
         _context.setVault("");
         _context.connect();
         _context.disconnect();
      }
      catch (MatrixException e)
```

```
            {
               _context = null;
               System.out.println("Can't set context");
               ListItr itr = new ListItr(e.getMessages());
               while(itr.next())
               {
                           System.out.println((String) itr.value());
               }
                           return false;
            }
            return true;
         }
         private void executecmd()
         {
            int cursor = getCursorType();
            setCursor(WAIT_CURSOR);
            message("Executing Mql Command");
            try
            {
               _context.getContext();
               _context.connect();

// Construct a new MQLCommand object
               MQLCommand mqlcommand = new MQLCommand();

// mqlcommand executed
               mqlcommand.executeCommand(_context,_command);
               _context.disconnect();

// mqlcommand results fetched
               String result = mqlcommand.getResult();
               System.out.println(result);

// mqlcommand errors fetched
               String error1 = mqlcommand.getError();
               System.out.println(error1);
            }
            catch(MatrixException e)
            {
               _context = null;
               System.out.println("Can't set context");
               ListItr itr = new ListItr(e.getMessages());
               while(itr.next())
               {
                           System.out.println((String) itr.value());
                           error(e, "Mql command execute error");
               }
            }
            finally
            {
               message("MQL command Executed");
               setCursor(cursor);
            }
         }
      }
```

## Sample 3: Query Class Methods

This is a simple program (QueryDb.java) that uses various methods of the Query class. It uses an existing query and evaluates this query to return Business Objects. This program will work as long as a query with the given name is available within the ENOVIA Live Collaboration Database. For more information on programming queries, see *Programming Notes for Queries* in Chapter 5.

The QueryDb.java program needs the following arguments to be passed:

```
hostname, server_name, Matrix user name, vault, owner,
whereclause, error_logfile
```

### Comments List

The following list of comments from the code is included to provide easy access to specific points in the sample program:

// Set Session Context using user and server names

// Set the query

// Evaluate the query

// Iterate through a list of business objects

// Write the data to the Java console

### Code sample - QueryDb.java

```
//
/* Desc of Program: This program uses various methods of the
Query class.
   Uses an existing query and evaluates this query to return
Matrix Business Objects.
   This program will work as long as a query with the given name
is available within the Matrix Database.
*/
import java.awt.*;
import matrix.db.*;
import matrix.util.*;
import java.io.*;
import matrix.client.*;

public class QueryDb extends Frame {

   Context _context;
   String _host;
   String _server;
   String _user;
   Query _query;
   String _vault;
   String _type;
   String _name;
   String _revision;
   String _owner;
   String _whereclause;
   String _str;
   String _errfile;

public QueryDb (String host, String server, String user, String
str, String vault, String type, String name, String revision,
String owner, String whereclause, String err_file)
{

   _host = host;
   _server = server;
   _user = user;
   _vault = vault;
   _type = type;
   _name = name;
   _revision = revision;
   _owner = owner;
   _whereclause = whereclause;
   _str = str;
   _errfile = err_file;

   try
   {
      FileOutputStream file1 = new FileOutputStream(_errfile);
      setContext(file1);
      getQuery(file1);
   }
      catch (MatrixException e)
         {
         }
      catch (IOException e)
      {
      }

      }
   public static void main( String args[])
   {
   new QueryDb( args[0], args[1], args[2], args[3], args[4],
args[5], args[6], args[7], args[8], args[9], args[10], args[11],
args[12]);
   }
```

// Set Session Context using user and server names

```
   public boolean setContext()
   {
      try
      {
         _context = new Context(_server,_host);
         _context.setUser(_user);
         _context.setPassword("");
         _context.setVault("");
         _context.connect();
         _context.disconnect();
      }

      catch (MatrixException e)
      {
         _context = null;
         System.out.println("Can't set Context");
         ListItr itr = new ListItr(e.getMessages());
         while(itr.next())
         {
                     System.out.println((String) itr.value());
         }
         return false;
      }
      return true;
   }
```

// Set the query

```
   public void getQuery() throws MatrixException
   {
      try
      {
         _query = new Query(_str);
                   _context.connect();

                   _query.open(_context);
                   _query.setVaultPattern(_vault);
                   _query.setBusinessObjectType(_type);
                   _query.setBusinessObjectName(_name);
                   _query.setBusinessObjectRevision(_revision);
                   _query.setOwnerPattern(_owner);
                   _query.setWhereExpression(_whereclause);
                   _query.setExpandType(true);
                   _query.save(_context);

// Evaluate the query
      BusinessObjectList list = _query.evaluate(_context);

// Iterate through a list of business objects
      BusinessObjectItr itr = new BusinessObjectItr(list);
      while(itr.next())
      {
         BusinessObject bo = itr.obj();
         System.out.println("type: " + bo.getTypeName());
         System.out.println("name: " + bo.getName());
         System.out.println("rev: " + bo.getRevision());
      }

// Write the data to the Java console
      System.out.println("image name: " +
_query.getImageName());
      System.out.println("name: " +
_query.getBusinessObjectName();
      System.out.println("type: " +
_query.getBusinessObjectType());
      System.out.println("revision: " +
_query.getBusinessObjectRevision());
      System.out.println("owner: " + _query.getOwnerPattern());
      System.out.println("vault: " + _query.getVaultPattern());
      System.out.println("whereclause: " +
_query.getWhereExpression());

      boolean expandExpr = _query.getExpandTypes();
         Boolean boolObj = new Boolean(expandExpr);
      System.out.println("expand: " + boolObj.toString());

      QueryList queryList = _query.getQueries(_context);
      QueryItr queryItr = new QueryItr(queryList);
      while (queryItr.next())
      {
         Query query = queryItr.obj();
         System.out.println(“queries: “ + query.getImageName());
      }
      _query.close(_context);
         _context.disconnect();
   }
   catch(MatrixException e)
   {
      System.out.println("Can't load objects");
   }

      finally
      {
         System.out.println("Objects  .....loaded
successfully");
      }

   }
}
```

# Sample 4: ENOVIA Live Collaboration Database Package Methods

The following samples include a number of sections of code that illustrate features added as of version 9.x.

## Set Select

The Java Studio Customization Toolkit class matrix.db.Set includes a method that accepts a context object and select statements and returns the selected data for all objects in the set.

This code demonstrates running select clauses on a set or list.

### Code sample - Set Select

// Instantiate the BusinessObject.

```
      Set _object =  new Set("AllObjects");
      _object.open(_context);

      StringList selectStmts = new StringList(4);
      selectStmts.addElement("type");
      selectStmts.addElement("name");
      selectStmts.addElement("id");

      BusinessObjectWithSelectList _objectSelectList = 
_object.select(_context, selectStmts);
      int size = _objectSelectList.size();
      String ODI[] = new String[size];
      BusinessObjectWithSelectItr busItr = new 
BusinessObjectWithSelectItr(_objectSelectList);
      //StringItr strItr = new StringItr(selectStmts);
```

## List Select

The Java Studio Customization Toolkit class matrix.db.Context includes a method getSelectBusinessObjectData, which accepts a stringList of Object ID and select statements and returns a BusinessObjectWithSelectList, which provides details about all the business objects.

### Code sample - List Select

// Getting Object IDs

```
      int i =0;
      while (busItr.next())
      {
           BusinessObjectWithSelect bus = new 
BusinessObjectWithSelect(busItr.obj());
           ODI[i]=(String)bus.getSelectData("id");
           System.out.println(ODI[i]);
           i++;
```

```
        }
```

// getSelectBusinessObjectData()

```
            BusinessObjectWithSelectList _busSel=
_context.getSelectBusinessObjectData(ODI,selectStmts);
            BusinessObjectWithSelectItr busItr1 = new
BusinessObjectWithSelectItr(_objectSelectList);
            StringItr strItr = new StringItr(selectStmts);
            while (busItr1.next())
            {
                 BusinessObjectWithSelect bus = new
BusinessObjectWithSelect(busItr1.obj());
                 while (strItr.next())
                  {
                  System.out.print("  ");
                  System.out.print(strItr.obj());
                  System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));

                }
                strItr.reset();

System.out.println("-----------------------------");
            }
           RelationshipWithSelectList relList =
_context.getSelectRelationshipData(ODI,selectStmts);

            RelationshipWithSelectItr relItr = new
RelationshipWithSelectItr(relList);
            StringItr strItr2 = new StringItr(selectStmts);
            while (relItr.next())
            {
                RelationshipWithSelect relsel = relItr.obj();

                Hashtable hash1 = relsel.getRelationshipData();
                Hashtable hash2 = relsel.getTargetData();

               // System.out.print(relsel.getLevel()+"
"+hash1.get("name"));

                while (strItr2.next())
                {

                    System.out.print("
"+hash2.get(strItr2.obj()));
                }
                System.out.println();
                strItr.reset();
            }
```

```
//
--------------------------------------------------------------
----
          _object.close(_context);
      }
      catch (MatrixException e)
      {
        e.printStackTrace();
      }
    }
```

## Expand Select

The Java Studio Customization Toolkit class matrix.db.BusinessObject includes a method that allows the caller to specify one or more select clauses to be used on a business object expand. The method returns the relationships of the business object.

The following demonstrates how to expand the business object. It gives all relationships to the level you specify.

### Code sample - Expand Select

// Instantiate the BusinessObject.

```
       StringList selectBusStmts = new StringList(4);
       selectBusStmts.addElement("type");
       selectBusStmts.addElement("name");
       selectBusStmts.addElement("owner");
       selectBusStmts.addElement("vault");

       StringList selectRelStmts = new StringList(3);
       selectRelStmts.addElement("name");
       selectRelStmts.addElement("to");
       selectRelStmts.addElement("from");
       short sh = 2;

       BusinessObject _object = new
BusinessObject("MX-TYPE-1","TEST6","1","MX-VAULT-1");
       _object.open(_context);
```

// Getting the expansion

```
 ExpansionWithSelect expan =  _object.expandSelect
(_context,"*","*",selectBusStmts,selectRelStmts,true,true,sh);
//
-------------------------------------------------------------
//  _object.expandSelect(_context, - Java context object
//  "*",                           - type Pattern
//  "*",                           - relationship Pattern
//  selectBusStmts,               - selects for Business Objects
//  selectRelStmts,                - selects for Relationships
//  true,                          - get To relationships
//  true,                          - get From relationships
//  sh);                           - recursion level (0 = all)
```

```
//
--------------------------------------------------------------
```

// Get Task

```
    ClientTaskList  ctl =  expan.getTasks();
    ClientTaskItr ctlItr = new ClientTaskItr(ctl);

    while (ctlItr.next())
           {
               System.out.print("  ");
               System.out.print(ctlItr.obj());
               System.out.print(" - ");
              System.out.println((ctlItr.obj()).getTaskData());
           }
```

// Getting Root Object

```
       BusinessObject _objRoot = expan.getRoot();
       _objRoot.open(_context);
       StringItr strItr = new StringItr(selectBusStmts);
      BusinessObjectWithSelect bus = _objRoot.select(_context,
selectBusStmts);
       System.out.println("The Root Business Object for this
Relation is :");
           while (strItr.next())
           {
               System.out.print("  ");
               System.out.print(strItr.obj());
               System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
           }
           strItr.reset();
      _objRoot.close(_context);
```

// Getting Root Data

```
   Hashtable rtHash = expan.getRootData();
   Enumeration rtEnum = rtHash.keys();

System.out.println("------------------------------------------
----");
   System.out.println("The Root Data  is");

         while(rtEnum.hasMoreElements()){
             String str = (String)rtEnum.nextElement();
             System.out.println(str + " : " + rtHash.get(str));
         }
```

// Getting Relationships.

```
       RelationshipWithSelectList _relSelectList =
expan.getRelationships();
```

```
               RelationshipWithSelectItr relItr = new
        RelationshipWithSelectItr(_relSelectList);

                  while (relItr.next())
                  {

                  RelationshipWithSelect relSelect =relItr.obj();
                  Hashtable relHash =relSelect.getRelationshipData();
                  Hashtable tarHash =relSelect.getTargetData();
                  Enumeration enumRel = relHash.keys();
                  Enumeration enumTar = tarHash.keys();
```

// Relationship Details

```
                 System.out.println("The Relationship is");

                 while(enumRel.hasMoreElements()){
                 String str = (String)enumRel.nextElement();
                 System.out.println(str + " : " + relHash.get(str));
                 }
```

// Target Details

```
                 System.out.println("The Target BusinessObject is ");

                 while(enumTar.hasMoreElements()){
                 String strs = (String)enumTar.nextElement();
                 System.out.println(strs + " : "+ tarHash.get(strs));
                 }


        System.out.println("-----------------------------------------
        -----");
               }

              _object.close(_context);

              }
              catch (MatrixException e)
              {
              e.printStackTrace();

           //      End of Relationship
```

## Set Subset Select and Select Count

The Java Studio Customization Toolkit class matrix.db.Set includes a method that accepts a context object as input and returns the number of objects the set contains. Another method is provided that returns selected data for a portion of a given set. For example, if a set has 100 objects, the user can retrieve the selected data for any n number of objects starting from any position.

The following prints the count of objects in a set and lists all objects in the set.

**Code sample - Set Subset Select and Select Count**

//     Instantiate the BusinessObject

```
       Set _object =  new Set("AllObjects");
       _object.open(_context);
 //
---------------------------------------------------------------
--
 //    public long count(Context context) throws MatrixException
 //
---------------------------------------------------------------
--
       long l = _object.count(_context);
       System.out.println("The Count of the Set is :"+l);

       StringList selectStmts = new StringList(4);
       selectStmts.addElement("type");
       selectStmts.addElement("name");
       selectStmts.addElement("revision");
       selectStmts.addElement("id");
       selectStmts.addElement("owner");
       selectStmts.addElement("policy");
       selectStmts.addElement("vault");
       selectStmts.addElement("modified");

       BusinessObjectWithSelectList _busSel=
_object.subsetSelect(_context,24,5,selectStmts);
       BusinessObjectWithSelectItr busItr = new
BusinessObjectWithSelectItr(_busSel);
       StringItr strItr = new StringItr(selectStmts);
       while (busItr.next())
       {
            BusinessObjectWithSelect bus = new
BusinessObjectWithSelect(busItr.obj());
            while (strItr.next())
             {
             System.out.print("  ");
             System.out.print(strItr.obj());
             System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));

           }
           strItr.reset();
           System.out.println("-----------------------------");
       }
//
---------------------------------------------------------------
-----
      _object.close(_context);
 }
 catch (MatrixException e)
```

```
{
  e.printStackTrace();
}
```

## Print Select

The Java Studio Customization Toolkit includes a method on a matrix.db.BusinessObject that allows the caller to specify one or more select clauses to be used on a business object print. The method returns a list of business object IDs with a name/value pair for each selected item.

### Code sample - Print Select

```
public PrintBusinessObject ()
   {
       // Variables.
       Context _context = null;

     try
     {

       _context = new Context(":bos","rheia");

            _context.setUser("creator");
            _context.setPassword("");
            _context.setVault("");
            _context.connect();
```

// Instantiate the BusinessObject.

```
            BusinessObject _object = new
BusinessObject("MX-TYPE-1", "TEST8", "1", "MX-VAULT-3");

            _object.open(_context);

            StringList selectStmts = new StringList(8);
            selectStmts.addElement("type");
            selectStmts.addElement("name");
            selectStmts.addElement("revision");
            selectStmts.addElement("id");
            selectStmts.addElement("owner");
            selectStmts.addElement("policy");
            selectStmts.addElement("vault");
            selectStmts.addElement("modified");

            StringItr strItr = new StringItr(selectStmts);
            BusinessObjectWithSelect bus =
_object.select(_context, selectStmts);
                while (strItr.next())
                {
                    System.out.print("  ");
                    System.out.print(strItr.obj());
```

```
                                System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
                }
                strItr.reset();

            _object.close(_context);
      }
      catch (MatrixException e)
      {
        e.printStackTrace();
      }
   }
```

## Query Select

The Java Studio Customization Toolkit class matrix.db.Query includes the method select, which returns the selected data for the result of a query. It assumes that a valid query exists in the database.

**Code sample - Query Select**

// Instantiate the Query.

```
            Query _object = new Query("z");

            _object.open(_context);

            StringList selectStmts = new StringList(3);

            selectStmts.addElement("type");
            selectStmts.addElement("name");
            selectStmts.addElement("revision");

           BusinessObjectWithSelectList _objectSelectList = new
BusinessObjectWithSelectList();

            _objectSelectList = _object.select(_context,
selectStmts);

            BusinessObjectWithSelectItr busItr = new
BusinessObjectWithSelectItr(_objectSelectList);
            StringItr strItr = new StringItr(selectStmts);
            int i = 1;
            while (busItr.next())
            {
                BusinessObjectWithSelect bus = new
BusinessObjectWithSelect(busItr.obj());
```

//To get the S.No and Id for the printed resulting Name-Value pair

```
                System.out.println("(" + i++ + ")" +
bus.getObjectId());
                while (strItr.next())
                {
```

```
                                        System.out.print("  ");
                                        System.out.print(strItr.obj());
                                        System.out.print(" - ");

System.out.println(bus.getSelectData(strItr.obj()));
                                }
                                strItr.reset();
                        }

                        _object.close(_context);
```

## Save/Load Structured Sets

A structured set is a collection of business objects and relationships that result from a expand operation. Structured sets are stored as persistent objects in the database. Relative levels are maintained. The Java Studio Customization Toolkit class matrix.db.Structure contains a number of methods. This sample loads an existing structure object and saves into another structure.

Instructions:

1. Create a Structure object from MQL using the following command:
1. `Expand bus “Type” “Name” “Revision” recurse to all structure “StructureName”;`
2. Give this created structure name in the statement:
3. `Structure _object = new Structure(“structureName”);`

**Code sample - Save/Load Structured Sets**

// Instantiate the Structure.

```
        Structure _object = new Structure("struct3");

        _object.open(_context);

        Visuals empty = new Visuals();

       Expansion objectExpand = _object.loadStructure(_context,
empty);

        _object.close(_context);

        Structure _objectNew = new Structure("struct3new");
        _objectNew.open(_context);
```

// Build copy of _objectNew structure and save it.

```
        String topNodeId = objectExpand.getRoot().getObjectId();
        ExpandRelationshipList relList =
objectExpand.getRelationships();
        StructureList strList = new
StructureList(relList.size());
        ExpandRelationshipItr relItr = new
ExpandRelationshipItr(relList);
```

```
            int i = 0;
            while (relItr.next())
            {
                String name = Integer.toString(i++);
                Structure tmp = new Structure(name);
                tmp.setLevel(relItr.obj().getLevel());
                System.out.println(tmp.getLevel());
                tmp.setExpanded(true);
                System.out.println(tmp.getExpanded());

                BusinessObject to = relItr.obj().getTo();
                BusinessObject from = relItr.obj().getFrom();
                BusinessObject origin = relItr.obj().getOrigin();

                if (to == origin)
                  tmp.setTargetId(from.getObjectId());
                else
                  tmp.setTargetId(to.getObjectId());

                tmp.setRelationshipId(relItr.obj().getName());

                strList.addElement(tmp);
            }
            _objectNew.saveStructure(_context,topNodeId,strList);
      }
```

## Check Access

The Java Studio Customization Toolkit class matrix.db.BusinessObject includes the method checkAccess(), which checks access privileges for a specific person.

Instructions:

**1.** Modify the following:

  **a )** Set the host and server values.

  ```
  _context = new Context(host, server);
  ```

  **b )** Set the user and password.

  **c )** Change the business object to a valid one in your database.

  ```
  BusinessObject _object = new BusinessObject(Type,
  name, Revision, vault)
  ```

**2.** Save and compile this program.

### Code sample - Check Access

```
import java.io.*;
import java.util.*;
import matrix.client.*;
import matrix.db.*;
import matrix.util.*;
```

```
public class TestCheckAccess extends Object
{

public static final short ALLACCESS =
(short)bosAccessEn.CALLACCESS.value();
public static final short CHANGELATTICE =
(short)bosAccessEn.CCHANGElATTICE.value();
public static final short CHANGENAME =
(short)bosAccessEn.CCHANGENAME.value();
public static final short CHANGEOWNER =
(short)bosAccessEn.CCHANGEOWNER.value();
public static final short CHANGEPOLICY =
(short)bosAccessEn.CCHANGEPOLICY.value();
public static final short CHANGETYPE =
(short)bosAccessEn.CCHANGETYPE.value();
public static final short CHECKIN =
(short)bosAccessEn.CCHECKIN.value();
public static final short CHECKOUT =
(short)bosAccessEn.CCHECKOUT.value();
public static final short CREATE =
(short)bosAccessEn.CCREATE.value();
public static final short DELETE =
(short)bosAccessEn.CDELETE.value();
public static final short DEMOTE =
(short)bosAccessEn.CDEMOTE.value();
public static final short DISABLE =
(short)bosAccessEn.CDISABLE.value();
public static final short ENABLE =
(short)bosAccessEn.CENABLE.value();
public static final short EXECUTE =
(short)bosAccessEn.CEXECUTE.value();
public static final short FREEZE =
(short)bosAccessEn.CFREEZE.value();
public static final short FROMCONNECT =
(short)bosAccessEn.CFROMCONNECT.value();
public static final short FROMDISCONNECT =
(short)bosAccessEn.CFROMDISCONNECT.value();
public static final short GRANT =
(short)bosAccessEn.CGRANT.value();
public static final short LOCK =
(short)bosAccessEn.CLOCK.value();
public static final short MODIFY =
(short)bosAccessEn.CMODIFY.value();
public static final short MODIFYFORM =
(short)bosAccessEn.CMODIFYfORM.value();
public static final short NONE =
(short)bosAccessEn.CNONE.value();
public static final short OVERRIDE =
(short)bosAccessEn.COVERRIDE.value();
public static final short PROMOTE =
(short)bosAccessEn.CPROMOTE.value();
public static final short READACCESS =
(short)bosAccessEn.CREADACCESS.value();
```

```
public static final short REVISE =
(short)bosAccessEn.CREVISE.value();
public static final short SCHEDULE =
(short)bosAccessEn.CSCHEDULE.value();
public static final short THAW =
(short)bosAccessEn.CTHAW.value();
public static final short TOCONNECT =
(short)bosAccessEn.CTOCONNECT.value();
public static final short TODISCONNECT =
(short)bosAccessEn.CTODISCONNECT.value();
public static final short UNLOCK =
(short)bosAccessEn.CUNLOCK.value();
public static final short UNUSED =
(short)bosAccessEn.CUNUSED.value();

public short getAccess(String str)
{
    if (str == "ALLACCESS")
    return ALLACCESS;
    else if (str == "CHANGELATTICE")
    return CHANGELATTICE;
    else if (str == "CHANGENAME")
    return CHANGENAME;
    else if (str == "CHANGEOWNER")
    return CHANGEOWNER;
    else if (str == "CHANGEPOLICY")
    return CHANGEPOLICY;
    else if (str == "CHANGETYPE")
    return CHANGETYPE;
    else if (str == "CHECKIN")
    return CHECKIN;
    else if (str == "CHECKOUT")
    return CHECKOUT;
    else if (str == "CREATE")
    return CREATE;
    else if (str == "DELETE")
    return DELETE;
    else if (str == "DEMOTE")
    return DEMOTE;
    else if (str == "DISABLE")
    return DISABLE;
    else if (str == "ENABLE")
    return ENABLE;
    else if (str == "CHECKIN")
    return CHECKIN;
    else if (str == "FREEZE")
    return FREEZE;
    else if (str == "FROMCONNECT")
    return FROMCONNECT;
    else if (str == "FROMDISCONNECT")
    return FROMDISCONNECT;
    else if (str == "GRANT")
    return GRANT;
    else if (str == "LOCK")
    return LOCK;
    else if (str == "MODIFY")
    return MODIFY;
    else if (str == "MODIFYFORM")
    return MODIFYFORM;
    else if (str == "NONE")
    return NONE;
    else if (str == "OVERRIDE")
    return OVERRIDE;
    else if (str == "PROMOTE")
    return PROMOTE;
    else if (str == "READACCESS")
    return READACCESS;
    else if (str == "REVISE")
    return REVISE;
    else if (str == "THAW")
    return THAW;
    else if (str == "TOCONNECT")
    return TOCONNECT;
    else if (str == "TODISCONNECT")
    return TODISCONNECT;
    else if (str == "UNLOCK")
    return UNLOCK;
    else if (str == "UNUSED")
    return UNUSED;
    else return 0;

}

   public TestCheckAccess ()
   {
        // Variables.
        Context _context = null;

     try
     {
// Create the context object and connect to the Collaboration Server.
      // Please set your host and server names accordingly.
           _context = new Context(":bos", "rheia");

// Set the userid.
           _context.setUser("MX-PERSON-1");

// Set the user password.
           _context.setPassword("");

         //   _context.setLattice("");
```

```
                      _context.connect();
```

// Instantiate the Business object.

```
                       BusinessObject _object = new
            BusinessObject("MX-TYPE-1", "TEST5", "1", "MX-VAULT-2");

                      _object.open(_context);

            System.out.println(_object.getTypeName()+"::"+_object.getRevisi
            on()+"::"+_object.getVault());

                       StringList strLst = new StringList();
                       strLst.addElement("ALLACCESS");
                       strLst.addElement("CHANGELATTICE");
                       strLst.addElement("CHANGENAME");
                       strLst.addElement("CHANGEOWNER");
                       strLst.addElement("CHANGEPOLICY");
                       strLst.addElement("CHANGETYPE");
                       strLst.addElement("CHECKIN");
                       strLst.addElement("CHECKOUT");
                       strLst.addElement("CREATE");
                       strLst.addElement("DELETE");
                       strLst.addElement("DEMOTE");
                       strLst.addElement("DISABLE");
                       strLst.addElement("ENABLE");
                       strLst.addElement("EXECUTE");
                       strLst.addElement("FREEZE");
                       strLst.addElement("FROMCONNECT");
                       strLst.addElement("FROMDISCONNECT");
                       strLst.addElement("GRANT");
                       strLst.addElement("LOCK");
                       strLst.addElement("MODIFY");
                       strLst.addElement("MODIFYFORM");
                       strLst.addElement("NONE");
                       strLst.addElement("OVERRIDE");
                       strLst.addElement("PROMOTE");
                       strLst.addElement("READACCESS");
                       strLst.addElement("REVISE");
                       strLst.addElement("SCHEDULE");
                       strLst.addElement("THAW");
                       strLst.addElement("TOCONNECT");
                       strLst.addElement("TODISCONNECT");
                       strLst.addElement("UNLOCK");
                       strLst.addElement("UNUSED");

                      StringItr strItr = new StringItr(strLst);
```

// This will print the various accesses for the business object

```
                      while (strItr.next())
                      {
```

```
           boolean _access =
_object.checkAccess(_context,getAccess(strItr.obj()));

            if (_access)
            System.out.println("User "+_context.getUser()+" has
access to "+strItr.obj());
            else
              System.out.println("User "+_context.getUser()+"
does not have access to "+strItr.obj());

           }

            _object.close(_context);
      }
      catch (MatrixException e)
      {
        e.printStackTrace();
      }
   }


   public static void main( String args[])
   {
     new TestCheckAccess();
   }

     // End of class declaration.
```

# C

# Tcl Scripting

## MQL commands not used for JPOs

### Downloadable Clause

*Not used for Java Program Objects.*

If the program includes code for operations that are not supported on the Web product (for example, Tk dialogs or reads/writes to a local file) you can include the `downloadable` clause. If this is included, this program is downloaded to the Web client for execution (as opposed to running on the Collaboration Server). For programs not run on the Web product, this flag has no meaning.

```
add program NAME downloadable;
```

If the downloadable clause is not used, `notdownloadable` is assumed.

Due to the restriction that downloaded programs must execute in a deferred mode, there are several cases that need to be addressed by system logic. If just the `downloadable` clause is given, then `deferred` is assumed (see *Execute Clause* of add program statement in *MQL Guide*). If the `downloadable` clause is given, and the `execute` clause is `immediate`, an error will be generated. Likewise, if on program modification command a mismatch occurs, an error will be generated that reads:

```
    A program that is downloaded cannot execute immediately.
```

Note that the `usesexternalinterface` clause continues to be supported for historical reasons. Scripts and programs that use this clause result in Program objects which have their `execute` flag set to `deferred` and their `downloadable` flag set.

## Piped Clause

*Not used for Java Program Objects.*

You can specify that external program objects use the "piped" service. Piped programs may use a built-in MQL command line service to handle standard input and output. When piped is specified, External is assumed. The piped service is not available to MQL or Java program objects. Execution may be immediate or deferred. Piped programs cannot be downloadable. Refer to *Writing Piped Program Code* for more information.

## Pooled Clause

*Not used for Java Program Objects.*

Each time an MQL program object runs Tcl code and then exits out of Tcl mode, a Tcl interpreter is initialized, allocated, and then closed. During an ENOVIA Live Collaboration session, you may execute several programs, and one program may call other programs, all of which require a Tcl interpreter and therefore the overhead of its use. In an effort to optimize multiple Tcl program execution, Tcl program objects may be specified as "pooled." When such a program is first executed in a session, a pool of Tcl interpreters is initialized, one of which is allocated for the executing code. When the code is completed, the interpreter is freed up. Subsequent Tcl code that is executed in a pooled program during the session will use an interpreter from the already initialized pool.

When programs are created, the default is that they are not pooled. To define or modify an MQL type program to use the pool of interpreters, use the following syntax:

mql< > add|modify program PROG_NAME [!]pooled;

The "!" may be used to turn off the `pooled` setting of a program.

The number of interpreters available in a session is controlled by the `MX_PROGRAM_POOL_SIZE` setting in the initialization file (matrix.ini or ematrix.ini). `MX_PROGRAM_POOL_SIZE` sets the initial size of the Tcl interpreter pool. This setting is also used to extend the pool size when all the interpreters in the pool are allocated and another is requested. The default is 10.

### Usage

Enabling the Tcl interpreter benefits MQL/Tcl programs that are nested or run in a loop, such as the trigger manager. Also, while wizards do not support the pooled setting, the Tcl programs that make up a wizard (load, validate, etc.) should make use of an interpreter pool to optimize performance.

Unexpected results may occur if the pooled setting is turned on in a program without first reviewing and validating its code. Good programming techniques must be adhered to ensure proper results. When using an interpreter pool, Tcl variables are not cleared before freeing up an interpreter. This means that programs must explicitly set variables before using them, in case a previously executed program made use of the same variable name.

External, downloadable and Java programs do not use the Tcl interpreter pool, regardless of the setting in the program definition. In addition, MQL/Tcl programs that use the TK toolkit will not benefit from using the pooled setting, since user interaction is required. In fact, unexpected results, such as leaving a TK dialog displayed, may occur due to the use of variables as described above.

*Before modifying existing programs to use the pooled setting, the code should be reviewed and validated. Only programs that include Tcl code but no TK code should use the pooled setting.*

# Handling Variables

## Using the Runtime Program Environment

The *Runtime Program Environment* (RPE) is a dynamic heap in memory used to hold name/value pairs of program environment variables. The RPE makes it possible to pass parameters between internal programs and scripts. External programs cannot access the RPE.

The RPE is initialized when the ENOVIA Live Collaboration application starts and terminates when the ENOVIA Live Collaboration application ends. Variables placed in RPE memory are generally accessible only for the duration of the outermost program. Variables can be made available for the duration of the application if the `global` keyword is used. A variable can be removed from the RPE at any time.

1. Pooled does not apply to JPOs.

2. RPE replaced macros to allow passing arguments into and out of Tcl programs (there was no other way). With the introduction of Java programs, we are now recommending passing arguments in directly through the calling sequence, and returning information in the form of a return argument. In other words, the Java language does everything needed. No more need for the RPE.

### RPE and MQL

The RPE can hold, in addition to input arguments, any arbitrary variables that a user chooses to define. The MQL "environment" set of commands is used for defining and managing RPE variables. These commands are accessible from the Tcl portion of MQL, and allow a program to write, read, delete, and list program environment variables.

Note that even though the program environment is accessible in Tcl mode (since this is where variable manipulation normally takes place), the program environment is managed by ENOVIA Live Collaboration. Note also that program environment variables are lost when the session ends. The commands are:

**Runtime Program Environment Commands**

| Mql Command | Arguments | Comment |
|---|---|---|
| `set env` | [global] VARNAME VALUE | Write variable to the RPE. |
| `get env` | [global] VARNAME | Read variable from the RPE. |
| `unset env` | [global] VARNAME | Delete variable from the RPE. |
| `list env` | [global] | List variables in the RPE. |
| `clear env` | | Delete all variables from the RPE. |
| `exists env` | [global] VARNAME | Checks if an RPE variable exists. If the variable exists, "1" is returned, otherwise "0" is returned. |
| `listnames env` | [global] | Returns just the names (not values) of all global RPE variables. |

When macros are filled during the launch of a Program object, the system places the existing set of macros into the RPE, so that they can be used by running programs.

Notes:

- The `set env` command will overwrite NAME with a new value if it already exists.
- The `get env` command returns VALUE (i.e., prints to stdout).
- The `list env` command prints all of the NAME=VALUE pairs.
- All RPE command outputs are sensitive to the `quote on|off` command.
- Only a System Administrator can issue the `clear env` command.
- The keyword `global` (explained later) is optional.
- VARNAME and VALUE are case sensitive, and need to be quoted if they contain spaces.

RPE and Tcl are closely integrated: you can set Tcl variables with RPE variables (using `set var [mql get env NAME]`) and set RPE variables with Tcl variables (using `mql set env NAME $var`). Of course, the VARNAME field itself can also be a Tcl variable.

In order to reduce the potential build-up of variables over time (if they are not explicitly removed from the RPE), the RPE has been divided into two different name spaces:

- **local** — cleared automatically by the system when the system reaches the idle state.
- **global** — never cleared by the system.

The **local** name space will keep variables around temporarily, but long enough for all nested programs/scripts to complete.

The **global** name space will keep variables around until specifically removed by the `unset` command. Of course, both name spaces go away when the application terminates.

When using MQL, it makes no sense to set a local RPE variable at the MQL command prompt (what is sometimes called the "idle state"). This is due to the fact that the local name space is cleared automatically by the system when the next MQL command prompt appears. Therefore, the system will automatically place RPE variables set at the MQL command prompt into the global name space (even if the `global` keyword is not given).

For example:

```
MQL<6> set env SITE Chicago;

MQL<7> set env global COURSE "Intro to MQL";

MQL<8> list env;
```

will result in both SITE and COURSE being in the list because both were placed in the global name space, which is not automatically cleared by the system. Note that it is good style (but certainly not required) to use all uppercase letters for global RPE variable names. Note also that the last command results in both name spaces being listed, but since the local name space is always empty at the command prompt, this command could have been `list env global` and resulted in the exact same list.

To summarize:

- When running MQL, the system is at the idle state when the MQL prompt appears. When running the GUI applications, the system is at the idle state when not specifically processing a request. (The message bars say "Ready.")

- By default, the RPE commands operate on the local name space. The one exception is that any variables set at the idle state automatically go into the global name space, even if the `global` keyword is not supplied.
- When the `global` keyword is used, only the global name space is searched/updated.
- When fetching variables from the RPE, the local name space is searched first and then the global name space. This is how most programming languages behave. If a local and global variable share the same name, then one must use the `global` keyword to fetch the global variable.
- All local variables placed into the RPE (no matter the depth of nesting of the programs/scripts) stay in the RPE until explicitly removed or the idle state is reached.

## Program Environment Variables

In using the RPE to pass data between program objects, careful attention must be paid to the naming of program environment variables and to the cleaning up of these variables. All data is passed in the form of strings. Here are some guidelines:

- The program name is automatically associated with a variable named “0”.

*When calling a JPO method that returns a String, a RPE variable named after the JPO name is created and set to the value of the returned string. This is not the case when calling a JPO method that returns an integer. No such RPE variable is created.*

- Program input parameters are named “1”, “2”, etc.
- The calling program can manually place the expected local variables into the RPE (using the naming scheme above) before executing the called program, or simply add them to the end of the execute command; the system will automatically place them in the RPE with the proper names.
- Programs should return their data in a variable with the name identified with the 1st input parameter.
- Lists should be returned using a Tcl form (space separated, with curly braces used to surround items that contain spaces or special characters).
- Since the RPE is shared memory, any variable can be overwritten by a variable of the same name. It is good practice to always capture program variables immediately into custom local variables to be assured that the values will be available where necessary within the program. RPE variables can be saved as local variables (in Tcl: `set localUser [mql get env USER]`) or saved back into the RPE under a different name that won’t be overwritten (for example: `programName.USER` where programName is the name of the program).

  Any ENOVIA Live Collaboration program or trigger that runs at any time before a program completes has the potential to alter RPE variables. For example, a user starts a wizard, then checks the attributes of another object in the database. Depending on how the object is configured, a program or trigger may fire automatically without the user’s knowledge that could overwrite existing RPE variables.

  A properly written program should *always* fetch out of the RPE all variables needed before proceeding.

For examples of using the Runtime Program Environment, see the Wizard Command in the programming reference section of the *MQL Guide*.

## Macro Processing and Command Syntax

*Macro processing* involves the replacement of variable and macro names with their current values to create the precise code that is executed from a Program object or with the `shell` command. Macro values are not evaluated unless and until they are actually used. For example, a modify attribute trigger will not cause an attribute range program to be executed unless the macro ${ATTRCHOICES} is explicitly used by the program.

Macros can affect the surrounding text, if the required characters have additional uses in programming, such as “$” or “\”. Because of this, in order to be able to include a literal "${" in your text, you must turn off the macro processing substitution on a case-by-case basis by using the escape character “\”.

For example, if `USER=guest` then:

```
output 'The value of \${USER} is ${USER}'
```

results in the output:

```
The value of ${USER} is guest
```

But one may also want the “\$” in the text to be macro processed. This can be accomplished by using a pair of backslashes. For example, if `MATRIXHOME=tools\matrix-9.0\` then:

```
output 'The file can be found in \\${MATRIXHOME}'
```

results in the output:

```
The file can be found in \tools\matrix-9.0\
```

So, to restate, macro processing of a backslash (\):

- followed by a “$”, results in a literal $
- followed by another backslash, results in a single backslash
- in all other cases is treated literally

The consequence of this means that any backslashes that need to appear in pairs (like the beginning of a UNC path) will require escaping twice by using four backslashes. For example, one might use:

```
shell '\\\\ourserver\Matrix9\bin\winnt\mql.exe -v';
```

Note that macro processing is done in a single pass. Therefore, the values substituted in for macro names are not subject to macro processing. For example, if: `MATRIXHOME=\\matrix9`, then one could use:

```
shell '${MATRIXHOME}\bin\winnt\mql.exe -v';
```

Also note that Tcl also recognizes the backslash (among other characters) as an escape character. This means any Tcl code in a Program may require additional escaping. Refer to Tcl documentation for information on escaping in Tcl.

### MQL and Tcl

Because of macro processing (as discussed above), parsing of Tcl statements containing backslashes within an MQL program object is different than when those same statements are run from an MQL command prompt.

For example, create a text file with the command lines:

```
    tcl;
    set a "one\\two\\three"
    puts "a contains $a"
    puts "b will use slashes instead of backslashes"
    regsub -all {\\} $a {/} b
    puts "now b contains $b"
```

Run this file from MQL (or alternatively type each line in MQL command window). The final line of output will be:

```
now b contains one/two/three
```

This is consistent with how the same lines of code behave in native Tcl.

However if you use this same code as an MQL program object, to get the same output you need to use triple rather than double backslashes so your code becomes:

```
    tcl;
    set a "one\\\two\\\three"
    puts "a contains $a"
    puts "b will use slashes instead of backslashes"
    regsub -all {\\\} $a {/} b
    puts "now b contains $b"
```

# Program Strategies for Tcl

## Nesting Programs

Programs can be nested—one routine can call another. In this way, programs can be fairly generic. For example, a larger program may need to have an application opened, so it could call a program originally created as a format command.

You must follow these rules when nesting programs:

- External programs can call either MQL or other external programs.
- MQL programs can call other MQL programs only. (With the inclusion of Tcl in MQL, most functions could be written in MQL.)
- Nesting update transactions within a read transaction should be avoided.

You cannot pass ENOVIA Live Collaboration-defined macros from program to program. These values from the database can be passed only to the first program called. If you need to pass values between programs, use the RPE, as explained in the section *Using the Runtime Program Environment*.

## Notifying Users (Notice, Warning and Error Commands)

MQL contains three commands which allow the programmer to provide graphical messages to the end user. The commands are:

```
notice MESSAGE;
warning MESSAGE;
error MESSAGE;
```

where `MESSAGE` is the text that is displayed to the end user.

These commands pop up a dialog box containing a message and an icon appropriate to the severity level. The advantage of using these commands is that it does not require that you write a Tk interface to report these messages. Some examples:

```
mql notice "Missing value for attribute $szAttr"
mql warning "Missing value for attribute $szAttr"
mql error "Missing value for attribute $szAttr"
```

The only difference in the above commands is the severity level. None of them will halt a transaction, but simply provide feedback to the user.

*Notice, Warning, and Error commands do NOT halt a transaction.*

*Use of the percent character (%) in Notice commands may give unpredictable results. If you must use the % character, use two percent signs (%%) so that subsequent characters will be treated as a literal string.*

## Writing Piped Program Code

Piped programs are run as a separate process, but the launching thread in the core acts as an "mql command" service. Assuming the piped program uses this service as the means for performing its ENOVIA Live Collaboration-related work (the externally-run program will use the pipes (stdin/stdout) provided to communicate to ENOVIA Live Collaboration), the piped program is essentially running inside the transaction of the launching thread. Not only does this mean that the piped program can rollback the transaction, it is already authenticated and acting in the context of the launching thread.

You can use the Web client (or Studio Customization Toolkit) to launch a piped program, which will be run on the server. In this sense, piped programs could be said to work "from the Web." You can use piped programs to implement all your business object methods, perhaps resulting in a "thinner" client.

Piped-programs cannot be downloadable. Therefore, a piped program would be the wrong choice to handle cases where client-side activities need to be performed (for example, a user interface). There are currently no plans to support downloaded piped programs in this general sense.

When external program objects are created, the code section indicates the path to the external program file that should be executed. The external file is either a compiled program or a script file. If a script file is used, it needs to be launched through its interpreter in the defining program object. For example the following are valid code values for piped program objects:

```
ENOVIA_INSTALL\bin\winnt\PIPEDPROGRAM.exe

java -cp \Matrix\lib\java PIPEDPROGRAM

wish80 PIPEDPROG.tcl
```

In the source code of these programs MQL commands can be passed through stdout and the results of these commands retrieved through stdin. When piped program objects are launched, a service thread is started which supports a simple MQL command line interface. The service is available through a full duplex “pipe” mechanism. Programs may issue MQL commands by simply writing the command string to their standard output device, prefixed by an “escape” (`\033`) character. The result of the command may be fetched by the program by reading the standard input device.

Standard input should be read either a character at a time or a line at a time. Reading the full buffer will seem to hang the system since no end of file mark is generated. The last line of the MQL output stream *for each command* is marked by a new line consisting of a single escape (\033) character, followed by another new line. This means that even if a command has no output, a line containing only the escape character is added to stdin, and will only be cleared by fetching the results.

If it is necessary to terminate an external program, it will not affect ENOVIA Live Collaboration or whatever process executed the external program.

The sections below show what changes to existing code should be made to take advantage of the pipe service.

## Tcl/Tk

When changing an external Tcl/Tk program object to piped, the “mql” is no longer required before MQL commands. You could create a Tcl procedure to write output to `stdout` and read from `stdin` such as:

```
    proc mqlCommand {command}{
    puts ”\033 $command”
    sets result{}
    gets stdin temp
    while {[string index $temp 0] != “\033”} {
    lappend result $temp
    gets stdin temp
    }
    return $result
    }
```

Then, in existing code, whenever an MQL command is issued, replace the preceding “mql” in all MQL lines with mqlCommand. For example:

```
    mqlCommand “print context”
```

## C++

MQL commands should be printed to `stdout`, and read from `stdin`. Sample code:

```
/*
** Include system header files.
*/
#include <stdio.h>
#include <stdlib.h>

/*
** Define global constants.
*/
#define MAX_BUFFER 8192

#define C_MQL_ESCAPE  '\033'
#define C_NEWLINE     '\n'

/*
** Define functions.
*/
char* mql( char* sCommand, char* sOutput );

int main( int argc, char* argv[], char* envp[] ) {

   char sOutput[MAX_BUFFER];

/*
** Example that returns output of multiple commands.
*/
   mql( "print context;list vault;list user;", sOutput );
   printf( "Output 1:\n%s\n", sOutput );

/*
** Example that returns output of a single command.
*/
```

```
   mql( "print person creator select name business system dump
|;", sOutput );
   printf( "Output 2:\n%s\n", sOutput );

/*
** Example that returns an error.
*/
   mql( "lsit vault;", sOutput );
   printf( "Output 3:\n%s\n", sOutput );

/*
** Return to the calling program.
*/
   return(0);

}

char* mql( char* sCommand, char* sOutput ) {

/*
** Define local variables.
*/
   char* sTmp = sOutput;
   char  c = (char) NULL;

/*
** Execute the MQL command.
*/
   fprintf( stdout, "%c%s%c", C_MQL_ESCAPE, sCommand, C_NEWLINE
);
   fflush( stdout );

/*
** Get the result.
*/
   while ( (c=getc(stdin)) != C_MQL_ESCAPE )
   {
   *(sTmp++) = c;
   }
   *(sTmp++) = (char) NULL;
   fflush( stdin );

/*
** Return the output.
*/
   return( sOutput );
}
```

## Avoid Initializing multiple Tcl Interpreters

Due to the time to initialize the Tcl session, a performance penalty is incurred every time an MQL/Tcl program is executed. For information on reducing this penalty using the Tcl Interpreter Pool feature, see description of the pooled clause of the add program statement in the *MQL Guide*. Also, programmers should be aware of the following:

- Use wizard frame prologue and epilogue programs instead of widget load and validate programs if the frame has many widgets.

- Avoid using the `exec program` or `execute businessobject` commands (or their Studio Customization Toolkit equivalents) to execute reusable code. A better approach is to define the reusable code as MQL/Tcl library program objects.
- Only use program expressions in an ENOVIA Live Collaboration table if absolutely necessary. Consider making minor schema changes if it eliminates the need to execute an MQL/Tcl program to populate the cell for each business object row.

## Storing “Static” Data

Program objects are often used to store information that is seldom changed and is required when executing custom code. In these cases, the program object does not have code in it. It is just being used as a text buffer. Some implementations have used program objects to store the results of queries, which take a long time to execute, but have results that change infrequently. The program object's contents are updated as a triggered event whenever any of the “queried” data is created, modified, or deleted. This approach allows the programs that need the query results to access the program object's contents instead of performing a time-consuming query. Basically, this approach penalizes the user who makes the infrequent change, but rewards the users who make the frequent queries.

The RPE is also an excellent technique to store this kind of information. With the RPE, the initial MQL commands to retrieve the data must still be executed. However, the results can then be stored in the RPE to avoid re-executing the same commands later in the session. It is quite common to store this information in the RPE during the “ValidatePasswordAction” program since end users are typically more tolerant of slower performance when logging onto the system.

The techniques described in this section are very valuable for improving the performance of a Wizard's widget load programs if they currently execute a query.

# Programming for the Web Navigator

When writing programs that may be used with the Web version of Matrix Navigator, there are considerations that need to be addressed. This section provides some tips to keep in mind if programs will be run on the Web.

## Downloadable Programs

In order for a program to be executed on the Web client, it must be marked as Downloadable (see *Downloadable Clause*). Downloadable programs can be run only in a deferred mode. Therefore, selecting the **Downloadable** option automatically sets the Execute option to Deferred.

If Downloadable is specified and the Execute option is Immediate, an error is generated. Likewise, if on program modification, a command mismatch occurs, an error is generated that reads:

```
    A program that is downloaded cannot execute immediately.
```

Programs not marked downloadable that are executed via a Web session (a trigger, for example) are executed on the Collaboration Server machine.

## Downloadable Program Threads Share Context

Beginning with the version 9 series of the ENOVIA Live Collaboration products, the handling of downloaded programs has been modified to fit within a "Web-centric" model. For technical reasons, all threads running on behalf of the Web browser share a single context. In versions prior to 9, each thread had its own independent context. This change has these two important implications, which are described in detail in the following sections:

- Matrix Web Navigator runs in its own thread and each downloaded program runs in its own thread. To avoid having the common context affected by concurrent threads, the Matrix Web Navigator thread is suspended until the downloaded program thread has completed. Therefore, all windows and dialog boxes associated with the Matrix Web Navigator are disabled until the other threads are completed.
- A single Runtime Program Environment (RPE) is associated with the single context that is now shared among all threads. In the rare cases when multiple downloaded programs run concurrently, this shared RPE may cause problems.

### Disabled Matrix Web Navigator Windows

While one (or more) downloaded programs are executing on the client, the Matrix Web Navigator dialog boxes and windows are disabled. Any dialogs displayed by the downloaded program(s) are fully operational. When all downloaded programs complete, the Matrix Web Navigator dialog boxes are once again enabled.

A downloaded program is deemed completed when it either exits or fails. The wish80 Tcl interpreter does not treat the Tcl `return` command the same as the Tcl `exit` command. According to the Tcl documentation, a `return` command should only be used within a Tcl procedure or inside a file that is passed to the Tcl `source` command. The only proper way to exit from the main code of a Tcl program is to use the Tcl `exit` command. This is extremely important now since a downloaded program that uses the `return` command where an `exit` command should have been used results in the Matrix Web Navigator never gaining control back. The only workaround is to kill the orphaned wish80 executable, which signals that the downloaded program has finished; control then returns to Matrix Web Navigator.

The exit, return, and source Tcl commands are described in detail below.

| Tcl Command | Synopsis | Description |
|---|---|---|
| `exit` | **exit** ?*returnCode*? | Terminate the process, returning returnCode to the system as the exit status. If returnCode isn't specified, then it defaults to 0. |
| `return` | **return** ?**-code** *code*?<br>?**-errorinfo** *info*?<br>?**-errorcode** *code*?<br>?*string*? | Return immediately from the current procedure (or top-level command or `source` command), with string as the return value. If string is not specified, then an empty string is returned.<br>In the usual case where the **-code** option isn't specified, the procedure returns normally (its completion code will be TCL_OK). However, the **-code** option may be used to generate *Exceptional Returns* from the procedure. |
| `source` | **source** *fileName*<br>**source -rsrc** *resourceName* ?*fileName*?<br>**source -rsrcid** *resourceId* ?*fileName*? | This command takes the contents of the specified file or resource and passes it to the Tcl interpreter as a text script. The return value from source is the return value of the last command executed in the script. If an error occurs in evaluating the contents of the script, then the `source` command returns that error. If a `return` command is invoked from within the script, then the remainder of the file is skipped and the `source` command returns normally with the result from the `return` command.<br>The **-rsrc** and **-rsrcid** forms of this command are only available on Macintosh computers. These versions of the command let you source a script from a TEXT resource. You may specify what TEXT resource to source by either name or id. By default, Tcl searches all open resource files, which include the current application and any loaded C extensions. Alternatively, you can specify the fileName where the TEXT resource can be found. |

## Exceptional Returns

The `-code` option is rarely used. It is provided so procedures that implement new control structures can reflect exceptional conditions back to their callers. *Code* may have any of the following values:

- **ok**

  Normal return: same as if the option is omitted.

- **error**

  Error return: same as if the `error` command were used to terminate the procedure, except for handling of `errorInfo` and `errorCode` variables. These two variables may be used to provide additional information during error returns. They are ignored unless *code* is **error**.

  The `-errorinfo` option specifies an initial stack trace for the `errorInfo` variable. If `errorInfo` is not specified, then the stack trace left in `errorInfo` includes the call to the procedure and higher levels on the stack but it does not include any information about the context of the error within the procedure. Typically the *info* value is supplied from the value left in `errorInfo` after a `catch` command trapped an error within the procedure.

If the `-errorcode` option is specified, then *code* provides a value for the `errorCode` variable. If the option is not specified, then `errorCode` defaults to `NONE`.

- **return**

  The current procedure returns with a completion code of `TCL_RETURN`, so that the procedure that invoked it returns also.

- **break**

  The current procedure returns with a completion code of `TCL_BREAK`, which terminates the innermost nested loop in the code that invoked the current procedure.

- **continue**

  The current procedure return with a completion code of `TCL_CONTINUE`, which terminates the current iteration of the innermost nested loop in the code that invoked the current procedure.

- *value*

  *Value* must be an integer; it is returned as the completion code for the current procedure.

## Shared RPE

Having a single shared context implies a single shared RPE. Therefore all threads read and write to the same RPE. Since Matrix Web Navigator is disabled during the running of a downloaded program, this is not a problem when a single downloaded program is executed. Similarly, a downloaded program that causes another downloaded program to execute (referred to as *chaining*) is not a problem. However, if multiple downloaded programs are executing concurrently then there are issues to be aware of.

First, since all programs are sharing the same RPE, when multiple downloaded programs are running concurrently there is the potential for one program to overwrite an RPE entry that was defined by another program. This behavior can actually be leveraged for positive gain when performed in a controlled fashion. When this is not desired behavior, it is a good idea to follow a convention for naming RPE entries that minimizes the chance of name clash. For example, since program names must be unique, each program could name its RPE entries by pre-pending the program name to the front of the variable name.

Second, since programs now share the same RPE, there are problems when one program, a parent program, launches multiple downloaded child programs. Unlike "chaining", where one program runs after the previous program is downloaded, this "multiple downloaded children" case results in all programs being downloaded at the same time (even if there are lots of lines of code in between the exec statements that launch each child program). This is due to the fact that downloaded programs are deferred and therefore queued up for execution when the system commits the transaction surrounding the parent program.

Along with each program on the queue is a snapshot of the RPE when the program was to be executed (that is, when the `exec` statement was encountered). This saved RPE snapshot is restored when the program is removed from the queue and executed. Normally (for deferred programs that are not downloaded) this results in the correct behavior, including the fact that any arguments passed to the program are correctly found in the RPE entries "0", "1", etc. However, since a number of queued, downloaded programs will be removed from the queue before they actually run (due to the latency of getting down to the client) the result is the snapshot associated with last program on the queue will overwrite the RPE. For example, the arguments passed to the last program will be found in the RPE entries "0", "1", etc. and therefore all programs will pick up these values (including the ones queued to execute prior to the last program on queue).

## Tcl Programs

If Web client users need to use tools, wizards or methods written with Tcl, they must install Tcl/Tk version 8 and a required Matrix .dll file on the client machine. For details, see *Setting up Matrix for Web Client Access* in the *Installing Matrix Applet* chapter of the *ENOVIA Live Collaboration Installation Guide*.

## Program Output

When a Tcl program is run, where does its output go? Since ENOVIA Live Collaboration can run programs on both the desktop client and the Web, the answer to this question may change depending on where the program is run. For example, if a program is run on the desktop client and its output goes the MQL window, the same program run on the Collaboration Server will send its output to the Collaboration Server window.

In a program run by ENOVIA Live Collaboration, there are three kinds of output from the program:

- Tcl command echo: every Tcl command echoes its output:

  ```
  % set var ^my value^
  my value
  %
  ```

- Tcl stdout/stderr: output generated by puts, Tcl error, return, exit
- MQL stdout/stderr: the output generated by ENOVIA Live Collaboration on execution of an MQL command, such as mql output, mql notice, wizards, etc. should always go exactly where the program specifies: MQL output goes to the MQL window; notice and wizards pop up appropriate dialogs.

The following explains how the types of output listed above are treated in different circumstances. For each of the following, if the output goes to the MQL window, a program run on the Collaboration Server will go to the Collaboration Server window. Otherwise, the output is the same whether the program is run on the desktop client or on the Collaboration Server.

1. Tcl commands that are not run inside of some level of {..} will echo to the MQL window. Enclosing the program in `eval { ... }` suppresses this.
2. Tcl stdout/stderr always goes to the MQL window.
3. Naked MQL command (no ‘set var’, no ‘catch’):

   ```
   % mql command
   ```

   Both stdout and stderr go to the MQL window.
4. When a Tcl variable is used to receive stdout:

   ```
   % set tclVar [mql command]
   ```

   Stdout goes to tclVar, stderr goes to the MQL window.
5. When catch is used to keep error from popping up MQL window:

   ```
   % set z [catch {mql command}]
   ```

   Error CODE (0 or 1) goes to variable z.

   Stdout and stderr disappear. Catch prevents them from going to the MQL window, but there is no variable to receive stdout/stdin.
6. Use catch and direct MQL stdout/err to Tcl variable:

   ```
   % set z [catch {mql command} tclVar]
   ```

   Error CODE (0 or 1) goes to variable z.

   Stdout/stderr goes to outStr.
7. Results can be different when the MQL command in 4,5,6 itself calls a program. In the examples below, this is shown as `exec prog`, but it could also be a method, or any command that causes a trigger to fire.

The Tcl output from such a second level is handled the same way: Tcl stdout/stderr goes to the MQL window. The following table shows where MQL stdout, stderr goes:

| **Command** | **MQL stdout and stderr:** |
|---|---|
| `mql exec prog SubProg` | go to MQL window |
| `set tclVar [mql exec prog SubProg]` | stdout goes to tclVar,<br>stderr to MQL window |
| `set z [catch {mql exec prog SubProg}]` | disappear |
| `set z [catch {mql exec prog SubProg} tclVar]` | go to tclVar |

# Index