

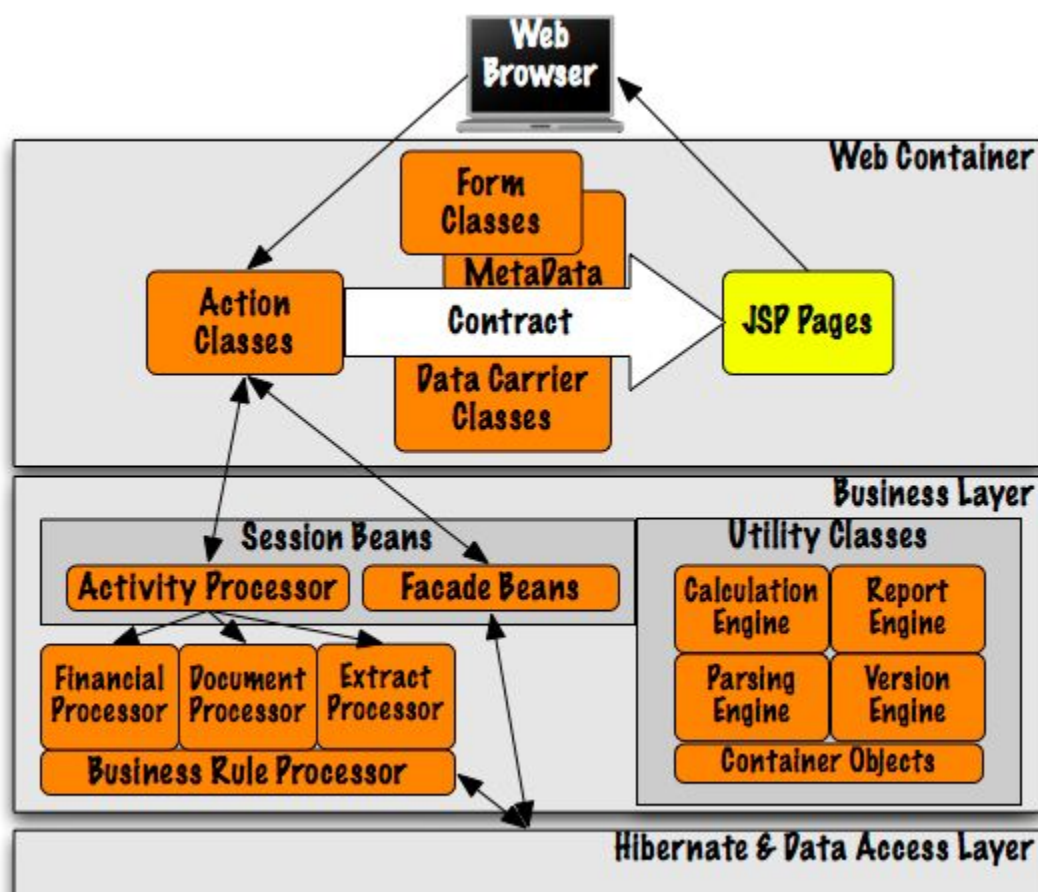
Section 3: Programming Model

Oracle's Programming Model.....	4
JAVA SERVER PAGES : JSPs.....	4
STRUTS	8
ENTERPRISE JAVA BEANS : EJB.....	10
BUSINESS LOGIC LAYER : BLL.....	10
DATA ACCESS LAYER : DAL	10
DATA ACCESS OBJECT : DAO	11
DATA CARRIER LAYER: DCL	11
JavaScript	11
FILE NAMES, FUNCTIONS, PARAMETERS.....	12
OPERATORS	14
CONDITIONAL STATEMENTS	17
LOOPING STATEMENTS	19
GLOBAL FUNCTIONS	21
COMMON METHODS	22
RESERVED WORDS	22
COMMON OBJECTS	23
Style Sheets	25
CASCADING STYLE SHEETS	25
OIPA J2EE folder structure.....	26
OIPA J2EE Naming Conventions	27
Class Names.....	27
Variables.....	27
Functions.....	27
Constants/Enums.....	28
Block Structures/Spacing	28
Declarations.....	28
try/catch/finally.....	28
Descriptions for Blocks.....	29
Line breaks/Concatenations.....	29
XML Format.....	29
Glossary.....	30
Debugging an Activity.....	30
SQL Server 2000 (if applicable).....	44
QUERY ANALYZER	44
Starting Query Analyzer.....	44
Ad-hoc querying (basics and important settings).....	45
Viewing the table structures.....	46
Altering and creating stored procedures and functions	47
Some basic OIPA queries.....	48
ENTERPRISE MANAGER	49
Introduction.....	49
Starting Query Analyzer.....	51
Altering and creating tables and indexes	52
Backup and restoring	56
PROFILER	63

<i>As a debugging tool</i>	63
<i>Index tuning wizard</i>	66
Source Code and Business Rule Source	72
CREATING A VSS DATABASE	72
CHANGING VSS DATABASE AND LOGIN.....	73
LOCAL COPY OF THE SOURCE CODE.....	76
CREATING PROJECTS IN VSS	76
SET WORKING FOLDER.....	78
GUID (Globally Unique Identifier).....	79
GUIDS IN SQL SERVER.....	79
XML: eXtended Markup Language	80
HISTORY OF XML.....	80
BUILDING BLOCKS OF XML.....	80
<i>Tag</i>	81
<i>Attributes</i>	81
XPATH QUERY	82
OIPA AND XML.....	82
XML STANDARDS	82
XML PAYLOAD AND SCALABILITY	83
<i>Mapping Information Data</i>	83
<i>Key and Business Critical Data</i>	83
<i>Business Detail Data</i>	83
<i>An Example</i>	83
Developing a Sample J2EE Presentation Component and Business Logic Component.....	84
PRESENTATION COMPONENT	84
<i>Java Server Page</i>	84
<i>Struts Action</i>	86
SAMPLE BUSINESS LOGIC COMPONENT	88
<i>Session EJB</i>	88
<i>Data Access Layer</i>	91
<i>Data Carrier and Hibernate Mapping</i>	92
TESTING.....	92

Oracle's Programming Model

The Oracle Insurance Policy Administration (OIPA) J2EE system is a scalable nTier web application where the web server only pushes HTML, JavaScript, and XML to the client's browser. Using this programming model, the OIPA architecture provides a model for developers to create a flexible and reusable application. By breaking the system into tiers, Oracle developers only modify or add a specific layer, rather than rewrite the entire application, if they decide to change technologies or scale up.



Java Server Pages : JSPs

Java Server Pages (JSP) separate the dynamic part of the pages from the static HTML. You simply write the regular HTML in the normal manner, using whatever Web-page-building tools you normally use. You then enclose the code for the dynamic parts in special tags, most of which start with "<%" and end with "%>".

JSPs are a server-side technology developed by Sun as an extension to the Java servlet technology. JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements -- the actual design and display of the page -- to help make the HTML more functional (i.e. dynamic database queries).

A JSP is translated into Java servlet before being run. It processes HTTP requests and generates responses like any servlet; however, JSP technology provides a more convenient way to code a servlet. Translation occurs the first time the application is run. A JSP translator is triggered by the .jsp file name extension in a URL. JSPs are fully interoperable with servlets. You can include output from a servlet or forward the output to a servlet, and a servlet can include output from a JSP or forward output to a JSP.

Aside from the regular HTML, there are three main types of JSP constructs that you embed in a page: *scripting elements*, *directives*, and *actions*.

Scripting elements let you specify Java code that will become part of the result servlet. Directives let you control the overall structure of the servlet. Actions let you specify existing components that should be used, and otherwise control the behavior of the JSP engine. To simplify the scripting elements, you have access to a number of predefined variables.

JSP Element	Syntax	Interpretation	Notes
JSP Expression	<code><%= expression %></code>	Expression is evaluated and placed in output.	XML equivalent is <code><jsp:expression>expression</jsp:expression></code> . Predefined variables are request, response, out, session, application, config, and pageContext (available in scriptlets also).
JSP Scriptlet	<code><% code %></code>	Code is inserted in service method.	XML equivalent is <code><jsp:scriptlet>code</jsp:scriptlet></code> .
JSP Declaration	<code><%! code %></code>	Code is inserted in body of servlet class, outside of service method.	XML equivalent is <code><jsp:declaration>code</jsp:declaration></code> .
JSP page Directive	<code><%@ page att="val"%></code>	Directions to the servlet engine about general setup.	XML equivalent is <code><jsp:directive.page att="val"></code> . Legal attributes, with default values in bold, are: <ul style="list-style-type: none"> • <code>import="package.class"</code> • <code>contentType="MIME-Type"</code> • <code>isThreadSafe="true false"</code> • <code>session="true false"</code> • <code>buffer="sizekb none"</code> • <code>autoflush="true false"</code> • <code>extends="package.class"</code> • <code>info="message"</code>

			<ul style="list-style-type: none"> • <code>errorPage="url"</code> • <code>isErrorPage="true false"</code> • <code>language="java"</code>
JSP include Directive	<code><%@ include file="url" %></code>	A file on the local system to be included when the JSP page is translated into a servlet.	XML equivalent is <code><jsp:directive.include file="url"></code> . The URL must be a relative one. Use the <code>jsp:include</code> action to include a file at request time instead of translation time.
JSP Comment	<code><%-- comment --%></code>	Comment; ignored when JSP page is translated into servlet.	If you want a comment in the resultant HTML, use regular HTML comment syntax of <code><-- comment --></code> .
The <code>jsp:include</code> Action	<code><jsp:include page="relative URL" flush="true"/></code>	Includes a file at the time the page is requested.	If you want to include the file at the time the page is translated, use the <code>page</code> directive with the <code>include</code> attribute instead. Warning: on some servers, the included file must be an HTML file or JSP file, as determined by the server (usually based on the file extension).
The <code>jsp:useBean</code> Action	<code><jsp:useBean att=val*/></code> or <code><jsp:useBean att=val*></code> ... <code></jsp:useBean></code>	Find or build a Java Bean.	Possible attributes are: <ul style="list-style-type: none"> • <code>id="name"</code> • <code>scope="page request session application"</code> • <code>class="package.class"</code> • <code>type="package.class"</code> • <code>beanName="package.class"</code>
The <code>jsp:setProperty</code> Action	<code><jsp:setProperty att=val*/></code>	Set bean properties, either explicitly or by designating that value comes from a request parameter.	Legal attributes are <ul style="list-style-type: none"> • <code>name="beanName"</code> • <code>property="propertyName *"</code> • <code>param="parameterName"</code> • <code>value="val"</code>
The <code>jsp:getProperty</code> Action	<code><jsp:getProperty name="propertyName" value="val"/></code>	Retrieve and output bean properties.	
The <code>jsp:forward</code> Action	<code><jsp:forward page="relative URL"/></code>	Forwards request to another page.	
The <code>jsp:plugin</code> Action	<code><jsp:plugin attribute="value"*></code> ... <code></jsp:plugin></code>	Generates OBJECT or EMBED tags, as appropriate to the browser type, asking that an applet be run using the Java Plugin.	

Other Available Parameters:

fieldName (required)

The name of the field as it is in the metadata (aka Screen Rule XML).

formProperty (required)

The name of the property in the form object for this screen.

onChange (Optional)

The include will automatically generate an onChange to set the form.changed property to "yes" and any other change event code from the form metadata. This property lets you add custom functions to be called in the event of an onChange.

fieldValue (Optional)

Sets the default value for a select/pick list.

disabled (Optional)

Allows you to override the disabled status of a field.

dynamic (Optional)

Must be set to yes for those fields that are dynamically created based on Screen Rule XML.

fieldId (Optional)

Sets the styleId of this field. Will default to the value in formProperty if not set.

styleWidth (Optional)

Sets the width in pixels of the field for display. Does not affect the max-length setting of fields. This value will be adjusted down for you if this is a field that has a quick entry button next to it like a calendar or calculator.

useDefaultOnChange (Optional)

Defaults to "Yes." Allows the user to turn off default onChange processing for a field.

searchField (Optional)

Must be set to yes for those fields that are "fixedFields" in the search screen XML data.

Struts

Oracle uses struts in our design to enforce more disciplined coding; which makes code easier to read and maintain. The use of struts creates a clear separation between presentation (JSP) and business logic. This allows us to create an extensible development environment based on coding standards and proven design patterns. Struts provides a unified framework for deploying servlet and JSP applications that use the MVC (Model View Controller) architecture.

Rather than hard-coding information into Java programs, many Struts values are represented in XML or property files. This loose coupling means that many changes can be made without modifying or recompiling Java code, and that wholesale changes can be made by editing a single file.¹

- Form beans

- In JSP, you can use `property="*" with jsp:setProperty to automatically populate a JavaBean component based on incoming request parameters. Apache Struts extends this capability to Java code and adds in several useful utilities, all of which serve to greatly simplify the processing of request parameters.`

- Bean tags

- Struts provides a set of custom JSP tags (`bean:write`, in particular) that let you easily output the properties of JavaBeans components. Basically, these are concise and powerful variations of the standard `jsp:useBean` and `jsp:getProperty` tags.

Struts tag properties

Simple (name) - The specified name identifies an individual property of a particular JavaBean. The name of the actual getter or setter method to be used is determined using standard JavaBeans introspection, so that (unless overridden by a `BeanInfo` class, a property named "xyz" will have a getter method named `getXYZ()` or (for boolean properties only) `isXYZ()`, and a setter method named `setXYZ()`.

Nested (name1.name2.name3) - The first name element is used to select a property getter, as for simple references above. The object returned for this property is then consulted, using the same approach, for a property named name2, and so on. The property value that is ultimately retrieved or modified is the one identified by the last name element.

¹ *Jakarta Struts: An MVC Framework Overview, Installation, and Setup. Struts 1.2 Version. Author- Marty Hall*

Indexed (nameindex) - The underlying property value is assumed to be an array, or this JavaBean is assumed to have indexed property getter and setter methods. The appropriate (zero-relative) entry in the array is selected. List objects are now also supported for read/write. You simply need to define a getter that returns the List

Mapped (name(key)) - The JavaBean is assumed to have a property getter and setter method with an additional attribute of type java.lang.String.

Combined (name1.name2index.name3(key)) - Combining mapped, nested, and indexed references are also supported.

Defining a Bean

If you need to do an either or type for defining a variable, you can use nested content within the bean:define. This gives a developer the ability to define a default value.. Here is a code snippet from the Client Screen:

```
<bean:define id="displayIndividual" type="java.lang.String">
  <logic:present name="ClientForm" property="metaDataDcl.displayIndividualFields">
    <bean:write name="ClientForm" property="metaDataDcl.displayIndividualFields"/>
  </logic:present>
  <logic:notPresent name="ClientForm" property="metaDataDcl.displayIndividualFields">
    No
  </logic:notPresent>
</bean:define>
```

In the above example, if the ClientForm.getMetaDataDcl().getDisplayIndividualFields() is present, the bean displayIndividual will be assigned its value. Otherwise displayIndividual will be assigned a value of "No".

Enterprise JavaBeans : EJB

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, remote Clients can access the inventory services provided by the application.

Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container--not the bean developer--is responsible for system-level services such as transaction management and security authorization.

Second, because the beans--and not the Clients--contain the application's business logic, the client developer can focus on the presentation of the Client. The Client developer does not have to code the routines that implement business rules or access databases. As a result, the Clients are thinner, a benefit that is particularly important for Clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant J2EE server.²

EJBs in addition, allow the web tier to access objects on the Business tier through an interface called façade.

Business Logic Layer : BLL

BLL implements specific business logic possessing and rules. Within the Oracle Insurance Policy Administration (OIPA) system this layer processes activities, cycle and more.

Data Access Layer : DAL

This layer accesses the database to retrieve and store data from the code.

² *The J2EE Tutorial - Sun Microsystems*

Data Access Object : DAO

The Data Access Object is used for accessing the database and retrieving/storing data through a direct database connection. The object is used primarily for performing direct SQL calls and to invoke stored procedures.

The DAO implements the access mechanism required to work with the data source. The business component that relies on the DAO uses the simpler interface exposed by the DAO for its Clients. The DAO completely hides the data source implementation details from its Clients. Because the interface exposed by the DAO to Clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its Clients or business components. Essentially, the DAO acts as an adapter between the component and the data source.³

Data Carrier Layer: DCL

This layer holds data that has been retrieved from the database or is to be stored in the database. For example, a recordset that is returned from a select statement within the system has a row of data that would be a DCL. A DCL is an object representation of one row in the recordset.

JavaScript

JavaScript is used to write server side processing on Java Server Pages.

- Variables can be declared only by using var:
var vCustomer.
- Multiple variables can be declared on one line:
var vCustomer, vAddress, vState

JavaScript is used on every page in the system. It performs validations, changes values on the screen, and submits data to an existing or new page at the click of a button. It is used for real time access to a page and its data. Calls to JavaScript functions can be written into the page at build time and later

³ Core J2EE Pattern Catalog Training Manual- Sun Microsystems

called at the firing of some event. For example, when a date field is filled, as soon as the cursor leaves the field, JavaScript is used to validate and reformat the value entered by the user.

One of the primary functions of JavaScript is to read data that has been written (not visibly) on the page at build time. With JavaScript, a user can list records in XML and later, at the click of a button, read the XML and populate fields already existing on the page. This minimizes traffic between the page and the server.

Much of the JavaScript used on the pages is actually brought in through include files.

```
<script LANGUAGE="JavaScript" SRC="../Lib/Java/AsCommon.js"></script>
<script LANGUAGE="JavaScript" SRC="../Lib/Java/AsValidation.js"></script>
<script LANGUAGE="JavaScript" SRC="../Lib/Java/AsDate.js"></script>
```

By pasting these lines at the top of the page, all their inherent functionality can be accessed.

The most common functions contained on the page are functions for closing it and for saving data. Apart from these, most additional functions are customized to the page's needs.

For reference, below is a table listing the common JavaScript library files and the functions contained within each of these files.

File Names, Functions, Parameters

File Name	Functions	Parameters
AsCommon.js	FindFieldName	
AsCommon.js	IsDefined	Object
AsCommon.js	MakeArray	Index
AsCommon.js	MouseOver	Button, Index
AsCommon.js	MouseOut	Button, Index
AsCommon.js	MouseDown	Button, Index
AsCommon.js	MouseClicked	Button, Index
AsCommon.js	SelectItem	ComboBox, Value
AsCommon.js	SelectItemByText	ComboBox, Text
AsCommon.js	TrimString	String
AsCommon.js	TaxIDOf	TypeCode, TaxID
AsCommon.js	TextOf	ComboBox, Value
AsCommon.js	ToProper	TextField
AsCommon.js	FindQuote	String
AsCommon.js	RemoveQuote	String
AsCommon.js	ReplaceQuote	String, Position
AsCommon.js	HighlightRow	TableName,RowIndex, StartCell, EndCell
AsCommon.js	DelightRow	TableName,RowIndex, StartCell, EndCell
AsCommon.js	HighlightRows	TableName,RowIndex, iEndIndex, StartCell, EndCell
AsCommon.js	DelightRows	TableName,RowIndex, iEndIndex, StartCell,

		EndCell
AsCommon.js	ShowAllHelp	FormName, HelpFolder
AsCommon.js	BuildAndDisplayHelp	FormName, HelpFolder, XMLData
AsCommon.js	HelpTextFixed	DialogName, ObjectName, X-CoOrd, Y-CoOrd
AsCommon.js	ShowHelpTextFixed	Title, Description, X-CoOrd, y-CoOrd
AsCommon.js	HideHelpTip	None
AsCommon.js	DisplaySecurity	FormName, PageName, SecurityFolder
AsCommon.js	BuildAndDisplaySecurity	FormName, PageName, SecurityFolder, XMLData
AsCommon.js	GetButtonPermission	PageName, ButtonName
AsCommon.js	LoginPageOf	None
AsCommon.js	WriteProcessingDiv	None
AsCommon.js	SetProcessingLocation	None
AsCommon.js	DisableButtons	FormName
AsCommon.js	SubmitPage	FormName, Operation, Action, Target, DisableButtons, Status
AsCommon.js	ShowPageMessage	PageMessage, LoginClientNumber
AsCommon.js	ShowOrHideDialog	DialogName

File Name	Functions	Parameters
AsCookie.js	GetCookie	CookieName
AsCookie.js	SetCookie	CookieName, CookieValue, CookieExpires, CookiePath, CookieDomain, CookieSecure
AsCookie.js	DeleteCookie	CookieName

File Name	Functions	Parameters
AsDate.js	IsLeapYear	Year
AsDate.js	IsValidDate	Date
AsDate.js	DateAdd	Interval, Number, OldDate
AsDate.js	GetSemiAnnual	OldDate
AsDate.js	GetBiMonthly	OldDate
AsDate.js	LocalTimeOf	GMT

File Name	Functions	Parameters
AsFinancial.js	DescriptionOf	CodeName, StatusCode
AsFinancial.js	GetFieldValue	FieldName
AsFinancial.js	IsGUID	String

File Name	Functions	Parameters
AsPageEnd.js	ActivateShadow	None
AsPageEnd.js	HandleRightClick	None

File Name	Functions	Parameters
AsUpload.js	ShowUploadDialog	LoginClientNumber, CompanyCosmetics, Operation, Title, XMLData, TypeCode
AsUpload.js	ShowDownloadDialog	LoginClientNumber, CompanyCosmetics, Operation, Title, XMLData, TypeCode

File Name	Functions	Parameters
AsValidation.js	CheckDate	txtDate
AsValidation.js	FormatDate	dtDate
AsValidation.js	CheckNumber	txtNumber, bPositiveOnly, bIntegerOnly, bMoneyOnly, disNumber
AsValidation.js	CheckPercent	txtNumber
AsValidation.js	ValidateXML	sXML
AsValidation.js	CheckNumberList	txtNumber, bPositiveOnly, bIntegerOnly, bMoneyOnly
AsValidation.js	FormatToMoney	sFormName, sField
AsValidation.js	IsANumber	sNumber
AsValidation.js	CheckMask	txtField, sMask
AsValidation.js	FillByMask	txtField, sMask, sMaskType, sReplace
AsValidation.js	FormatNumber	sNumber, iDecimalPlaces

A list of some of the operators/commands/conditional statements/etc. available in JavaScript is shown below.

Operators

Arithmetic Operators

Operator	Description	Examples
+ (add)	Adds numeric operands or concatenates two or more strings.	1) x+y 2) "Me"+" You" Result: "Me You"
- (subtract)	Subtracts numeric operands.	x-7
* (multiply)	Multiplies numeric operands.	5*2
/ (divide)	Divides numeric operands.	y/x
% (modulo)	Returns the remainder when the first operand is divided by the second operand.	7%2 Result: 1

++ (increment)	<p>Increments the operand by 1. Note the following two possible behaviors:</p> <p>1) If operator is used before operand (ie: ++x), it increments the operand and evaluates to the incremented value.</p> <p>2) If operator is used following operand (ie: x++), it increments the operand but evaluates to the unincremented value.</p>	<p>Using x=2 for each example below:</p> <p>1) x++ Result: x=3</p> <p>2) y= ++x Result: 3</p> <p>3) y= x++ Result: y=2</p>
-- decrement	<p>Decrements the operand by 1. Note the following two possible behaviors:</p> <p>1) If operator is used before operand (ie: --x), it increments the operand and evaluates to the incremented value.</p> <p>2) If operator is used following operand (ie: x--), it increments the operand but evaluates to the unincremented value.</p>	x--

Note: With all of the Operators above (except Addition (+)), if an operand used is non numeric, operator will attempt to convert it to a number first.

Comparison Operators

Operator	Description	Examples
==	Tests for equality in value between two operands.	3==8 Result: returns false
===	Tests for equality between two operands both in terms of value and type. Supported in JavaScript 1.3+	Using x=3 and y="3": 1) x==y Result: returns true 2) x===y Result: returns false
<	Less than.	x<y
<=	Less than or equal to.	5<=x
>	Greater than.	y>4
>=	Greater than or equal.	x>=y

Assignment Operators

Operator	Example	Equivalent to:
=	y=x+3	n/a
+=	x+=3	x=x+3
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Logical Operators

Operator	Description	Examples
&& (AND)	Logical AND.	(x<3 && y<=5)
(OR)	Logical OR	(x<3 y<=5)
! (NOT)	Logical NOT.	!(x>3)

Bitwise Operators

Operator	Description
&	Bitwise AND.
	Bitwise OR
^	Bitwise XOR.
<<	Shift left.
>>	Shift right.
>>>	Shift right, zero fill.

Other Operators

Operator	Description														
?:	<p>"?:", or the Conditional Operator, is a shorthand method for constructing an evaluation, and executing two different assignments depending on the result. The syntax is:</p> <pre>{condition}? iftrue : iffalse</pre> <p>For example:</p> <pre>y=-1 x=(y<0)? 5: 10</pre> <p>x will contain "5" in the above case.</p>														
typeof	<p>The "typeof" operator allows you to probe the data type of its operand, such as whether a variable is string, numeric, or even undefined. Here are some of the possible return values:</p> <table> <tr> <th>Evaluates to</th><th>Indicates</th></tr> <tr> <td>"number"</td><td>Operand is a number</td></tr> <tr> <td>"string"</td><td>Operand is a string</td></tr> <tr> <td>"boolean"</td><td>Operand is a Boolean</td></tr> <tr> <td>"object"</td><td>Operand is an object</td></tr> <tr> <td>null</td><td>Operand is null.</td></tr> <tr> <td>"undefined"</td><td>Operand is not defined.</td></tr> </table>	Evaluates to	Indicates	"number"	Operand is a number	"string"	Operand is a string	"boolean"	Operand is a Boolean	"object"	Operand is an object	null	Operand is null.	"undefined"	Operand is not defined.
Evaluates to	Indicates														
"number"	Operand is a number														
"string"	Operand is a string														
"boolean"	Operand is a Boolean														
"object"	Operand is an object														
null	Operand is null.														
"undefined"	Operand is not defined.														

instanceof	Returns a Boolean value indicating whether the left hand operand is of the type specified in the right hand operand. For example:
	<pre>var today=new Date() alert(today instanceof Number) //alerts false, since today is not a Number.</pre>
delete	Deletes a custom object property, method, or array element. For example:
	<pre>var mycar=new Object() mycar.color="red" delete mycar.color //deletes the property "color"</pre>

Conditional Statements

Statement	Description
if	<p>Syntax:</p> <pre>if (expression) statement</pre> <p>OR:</p> <pre>if (expression){ statement1 statement2 }</pre>
if/ else	<p>Syntax:</p> <pre>if (expression) statement1 else statement2</pre> <p>OR:</p> <pre>if (expression) statement1 else if (expression2) statement2 else statement3</pre>
switch case break default:	<p>The "switch" statement lets you easily execute a different statement depending on the value of an expression (label).</p> <p>Syntax:</p> <pre>switch (expression){ case label1: statement1 break case label2: statement2 break default: statement3;</pre>

	<pre>} "break" is an optional statement that tells "switch" to exit should the proceeding statement be executed. "default" allows you to specify the statement to execute if non of the statements above are executed. An example speaks a thousand words: Example: switch (favoritemovie){ case "Annuity": alert("You chose Annuity ") break case "Life Insurance": alert("You chose Life Insurance ") break case "Health Insurance": alert("You chose Health Insurance ") break default : alert("You forgot to choose a product!") }</pre>
?:	<p>"?:", or the Conditional Operator, is a shorthand method for constructing an evaluation, and executing two different assignments depending on the result. The syntax is:</p> <pre>{condition}? iftrue : iffalse</pre> <p>Example:</p> <pre>y=-1 x=(y<0)? 5: 10</pre> <p>x will contain "5" in the above case.</p>
return	<p>Used inside functions, "return" lets you exit a function and optionally return a value.</p> <p>Example 1:</p> <pre>function test(a, b) var c=a*b return c }</pre> <p>Example 2:</p> <pre>function whatever(a){ if (a<3) return }</pre>

Looping Statements

Statements	Description
for	<p>Probably the common looping statement, "for" executes the enclosed statements any desired number of times.</p> <p>Syntax:</p> <pre>for (initialValue; test; increment) statement</pre> <p>Example:</p> <pre>for (x=0; x<3; x++) document.write("This text is repeated three times
")</pre>
while	<p>A while loop continuously executes the enclosed statements as long as the tested expression evaluates to true. If the expression is false to begin with, the entire while loop is skipped.</p> <p>Syntax:</p> <pre>while (expression) statement</pre> <p>Example:</p> <pre>var number=0 while (number<5){ document.write(number+"
") number++ }</pre>
do/while	<p>The do/while statement differs from the standard "while" in that the expression tested for is put at the end, ensuring that a do/while loop is always executed at least once.</p> <p>Syntax:</p> <pre>do statement while (expression)</pre> <p>Example:</p> <pre>var number=0 do{ document.write(number+"
") number++ } while (number<5)</pre>
for/in	<p>For/in is a special looping statement that lets you display the property names (or properties' value) of an object.</p>

	<p>Syntax:</p> <pre>for (variable in object) statement</pre> <p>"variable" is any arbitrary variable that will hold the current property of the object in question as it is being looped. Here are two examples.</p> <p>Example 1: This example writes out all the properties of the "navigator" object:</p> <pre>for (myprop in navigator) document.write(myprop+"
")</pre> <p>Example 2: This example writes out the properties, plus their corresponding value, of the "navigator" object:</p> <pre>for (myprop in navigator) document.write(myprop+": "+navigator[myprop]+"
")</pre>
break	<p>The break statement causes an exit of the innermost loop the statement is contained in. It's useful to prematurely end a loop:</p> <p>Example:</p> <pre>for (i=0; i<6; i++){ if (puppies[i].name=="George") alert("I found George!") break }</pre>
continue	<p>The continue statement causes JavaScript to skip the rest of the statements that may follow in a loop, and continue with the next iteration. The loop is not exited, unlike in a break statement.</p> <p>Example:</p> <pre>for (i=1; i<6; i++){ if (i==4) continue document.write(i+"
") } //1, 2, 3, 5 is written out.</pre>

Global functions

Note: "[]" surrounding a parameter below means the parameter is optional.

Operator	Description
parseInt(x, [radix])	<p>Parses any string "x" and returns the first valid number (integer) it encounters. If the first character in the string is not a number, white spaces, or a leading minus sign, parseInt() returns NaN instead. You can test for NaN using the isNaN() function below.</p> <p>parseInt() supports an optional 2nd "radix" parameter to specify the base of the number to be parsed (valid range is 2-36). Entering "10" would parse the number in the familiar decimal system, while "16" would be hexadecimal. Without this parameter present, parseInt() assumes any number that begins with "0x" to be radix 16, "0" to be radix 8, and any other number to be radix 10.</p> <p>Examples:</p> <pre>parseInt("3 chances") //returns 3 parseInt(" 5 alive") //returns 5 parseInt("I have 3 computers") //returns NaN parseInt("17", 8) //returns 15</pre>
parseFloat(x)	<p>Parses any string "x" and returns the first valid floating point number it encounters. Use this function to extract numbers with decimals, for example. If the first character in the string is not a number, white spaces, or a leading minus sign, parseFloat() returns NaN instead. You can test for NaN using the isNaN() function below.</p> <p>Example:</p> <pre>parseFloat("-3.98 points") //returns -3.98</pre>
isNaN(x)	<p>isNaN() checks its parameter "x" to see if it's the value "NaN" (not a number)- an illegal number. The function returns true if "x" is NaN, and false if not. A common example of NaN is 0 divided by 0. This function is required to detect NaN, as using equality operators (== or ===) won't do.</p>
isFinite(x)	<p>Determines whether a number is finite. Returns false if x is +infinity, -infinity, or NaN. JavaScript 1.3 statement.</p>
eval(s)	<p>Evaluates the string ("s") and returns the results of the evaluation (if available). Eval() allows you to dynamically construct a JavaScript statement or expression.</p> <p>Examples:</p> <pre>eval("8+3+1") //returns 12 eval("alert(Math.random())") //alerts Math.random()</pre>
escape(s)	<p>Accepts a string "s" and returns its encoded version. Original string isn't modified. All characters within the string that are non-alphanumeric nor one of the characters @ * - + . / are encoded. The escape() function is often used to encode cookie values before they're stored into a cookie (such as a URL).</p> <p>Note that escape() has been deprecated in ECMAScript 3.0 in favor of encodeURIComponent(). The later is supported in JavaScript 1.5 and up, while the former indefinitely (though formally deprecated).</p> <p>Example:</p>

	escape("JavaScript Kit") //returns "JavaScript%20Kit"
unescape(s)	<p>Unescapes an encoded string "s" previously encoded by escape().</p> <p>Note that escape() has been deprecated in ECMAScript 3.0 in favor of decodeURI(). The later is supported in JavaScript 1.5 and up, while the former indefinitely (though formally deprecated).</p>

Common Methods

methods	Description
alert(msg)	Displays an Alert dialog box with the desired message and OK button.
confirm(msg)	Displays a Confirm dialog box with the specified message and OK and Cancel buttons.
prompt(msg, [input])	Displays a Prompt dialog box with a message. Optional "input" argument allows you to specify the default input (response) that gets entered into the dialog box. Set "input" to "" to create a blank input field.

Reserved Words

abstract boolean break byte case catch char class const continue debugger default delete do double	else enum export extends false final finally float for function goto if implements import in	instanceof int interface long native new null package private protected public return short static super	switch synchronized this throw throws transient true try typeof var void volatile while with
--	--	--	---

Common Objects

Object	Properties
Anchor	Anchors are defined in HTML with the tag , and represented in JavaScript via the Anchor object
Applet	The Applet object inherits all public properties of the Java applet in question.
Area	Each link on your page is represented by a corresponding link object in JavaScript, with events and properties as described below. You may also loop through every link on your page via the links collection
Array	Alternative way of defining variables that allow multiple variables to be quickly declared,
Boolean	The Boolean object is an object wrapper for a boolean (true or false) value
Date	Ways to instantiate the date object: new Date() new Date(milliseconds) new Date(dateString) new Date(year, month, day, hours, minutes, seconds, milliseconds) //most parameters here are optional. Not specifying causes 0 to be passed in.
Document	Document is the parent object of numerous other objects, such as "images", "forms" etc.
Event	The Event object (IE5+/NS6+) keeps tracks of various events that occur on the page, such as the user moving the mouse or clicking on the link, and allows you (the webmaster) to react to them.
Form	Forms on the page are represented in JavaScript via the form object. <ul style="list-style-type: none"> • Button • Checkbox • elements • FileUpload • Hidden • Option • Password • Radio • Select • Submit • Text • Textarea
Reset	The reset button (<input type="reset">) within your form can be accessed and manipulated using JavaScript, via the corresponding Reset object.
Frame	The Window object is the top level object in JavaScript, and contains in itself several other objects, such as "document", "history" etc..
Function	<p>Function Statement:</p> <pre>function functionname(arguments){ //standard function statements... }</pre> <p>function (arguments){ //unnamed function. JavaScript 1.2. statements... }</p> <p>Constructor:</p> <pre>new Function(arguments, body)</pre>

History	The History object contains an array of previously visited URLs by the visitor. To simulate the browser's back button.
Image	Images on the page are represented by the Image object in JavaScript.
Link	Each link on your page is represented by a corresponding link object in JavaScript, with events and properties as described below. You may also loop through every link on your page via the links collection.
Location	Location contains information about the current URL of the browser. The most common usage of Location is simply to use it to automatically navigate the user to another page.
Math	The Math object allows you to perform common math related tasks.
Navigator	The Navigator object of JavaScript returns useful information about the visitor's browser and system.
Number	The Number object is an object wrapper for primitive numeric values. It contains some useful methods, introduced in JavaScript 1.5.
Object	You can define your own objects in JavaScript, with custom properties and methods.
RegExp	<p>Regular expressions are a powerful tool for performing pattern matches in Strings in JavaScript. You can perform complex tasks that once required lengthy procedures with just a few lines of code using regular expressions.</p> <p>Regular expressions are implemented in JavaScript in two ways:</p> <ol style="list-style-type: none"> 1) Using literal syntax. 2) When you need to dynamically construct the regular expression, via the RegExp() constructor.
Screen	Use the Screen object of JavaScript to derive basic information about the user's screen, such as dimensions.
String	The String object of JavaScript allows you to perform manipulations on a stored piece of text, such as extracting a substring, searching for the occurrence of a certain character within it etc.
Style	The Style object of the DOM allows you to dynamically change the values of your CSS properties, whether defined inline or via an external style sheet. The changes are instantly reflected on the page.
Window	The Window object is the top level object in JavaScript, and contains in itself several other objects, such as "document", "history" etc.

Style Sheets

Cascading Style Sheets

Cascading style sheets are used in all pages of the system. They act much like a library file, which stores descriptions of common elements among screens objects such as text boxes, combo boxes, and tables.

```
<link REL="stylesheet" HREF="../Lib/Style/AsCommon.css">

<td WIDTH="210">
    <input TYPE="TEXT" CLASS="styleFieldInColumn">
</td>
```

This is a simple example of the use of cascading style sheets (CSS). In this case, the referenced class is StyleFieldInColumn, which is defined in AsCommon.css. Below is the actual definition of this style sheet:

```
.styleFieldInColumn {font-family:MS Sans Serif; font-size:8pt;}
```

This merely defines font and font size. Many more properties can also be defined through style sheets, including:

```
background-color
border-width
border-color
border-style
filter
left
top
position
cursor
text-align
```

Alternately, the following code could be used:

```
<td WIDTH="210">
    <input TYPE="TEXT" STYLE=" font-family:MS Sans Serif; font-size:8pt;">
</td>
```

However, this code is much more cumbersome, especially when defining the values of several more style properties.

Using style sheets saves considerable time in page design and implementation and also helps maintain the standard look and feel. It also facilitates easier global changes to the site. All style sheets can be seen in the ../lib/style directory.

OIPA J2EE folder structure

OIPA uses the Eclipse Platform as its version control software. Outlined below is the folder structure.

```
[development].....Root folder
  [adminserver.j2ee.ear] ..... 'Complete System compile and build'
files
    application.xml
    build.xml
  [adminserver.j2ee.ejb] ..... AdminServer EJB Root folder
    [build] ..... EJB build files
      build.xml
      *.xdt
    [source] ..... Entity Java Beans source folder
      [com] ..... 'com' namespace
        [adminserver] ..... 'adminserver' namespace
          [dao] ..... Data Access Objects
            *.java
          [dal] ..... Data Access Layer source folder
            *.java
          [util] ..... Utility objects source folder
            *.java
          [bll] ..... Business Logic Layer source folder
            *.java
  [adminserver.j2ee.libs] ..... AdminServer support libraries folder
    *.jar
  [adminserver.j2ee.web] ..... AdminServer WebApp Root folder
    [build] ..... 'Web compile and build' files
      build.xml
      [merge] ..... Deployment descriptor files
        *.xml
    [jsp] ..... JSP files folder
      index.jsp
      [s3] ..... Web sub folder
      [activity]
      .
      .
      .
    [lib]
      [javascript] ..... JS files folder
        *.js
      [includes] ..... Common JSP includes folder
        *.jsp
      [style] ..... Style Sheets folder
        *.css
    [source] ..... Source Java files for Web pages
      [com] ..... 'com' namespace
        [adminserver] ..... 'adminserver' namespace
          [ui] ..... User Interface Processing source folder
            *.java
          [dcl] ..... Data Carrier Layer source folder
            *.java
          [util] ..... Utility objects source folder
            *.java
```

OIPA J2EE Naming Conventions

Class Names

- Follow the pascal naming convention.

Example:

```
public class ScreenUnmarshaller extends BaseUnmarshaller {
```

Variables

- All variables follow the camel naming convention.
- All variables instancing OIPA classes should be suffixed by the following three letter abbreviation.

Data Carrier Layer	Dcl
Business Logic Layer	Bll
Data Access Objects	Dao
Utility Objects	Utl
User Interface processing Objects	Uip

- Variables representing XML tags and attributes should be named the same as the tag or attribute.
- Variables representing database columns should be named the same.
- Function variables are declared when needed and not at the beginning of a function call.
- Member variables are declared at the beginning of a class.

Example:

```
private String calcType = null;
private Expression expressionUtl = null;
private DisplayField fieldDcl = null;
```

Functions

- All functions follow the camel naming convention
- First word of a function/method should be a verb. Exceptions are listed below.
- Boolean functions can begin with any of the following: is, are, etc.
- Functions which are constructors, destructors, iterators, and inherited from system classes.

Example:

```
private String translateCode( String code ) {
    protected FormMetaData getMetaDataUtl() {
```

Constants/Enums

- Format - Class.UPPERCASE_PascalCasing

Block Structures/Spacing

- { starts on the same line
- } ends on a separate line

Example:

```
private String translateCode( String code ) {  
    }  
    if( token.equals( metaDataUtl.getFormOldName() ) ) {  
    }  
    else {  
    }  
}
```

Declarations

- No specific lines for declaring a variable.
- Declaration of variables become part of the code.
- Variables are declared when needed and not at the beginning of a function call.
- Member variables are declared at the beginning of a class.

Example:

```
while( tokenizer.hasMoreTokens() ) {  
    String token = tokenizer.nextToken();  
}
```

try/catch/finally

- try/catch/finally needed for all public functions.
- try/catch/finally optional for private functions.
- Group lines under a try/catch as needed.
- OIPA specific Exception class used for throwing exceptions.
- OIPA specific Exception class cannot be used for auto generated exception throws.

Example:

```
if( "Item".equals( localName ) ) {  
    try {  
    }  
    catch( NumberFormatException numberFormatException ) {  
    }  
}
```

Descriptions for Blocks

// Copyright... on the first line of every file

/**

* *Description*

* [*@deprecated*]

* [*@see* *list of files*]

*/

above every class declaration, where *@deprecated* is used when the class exists only for backward compatibility and *@see* is used when this class has some relevance in another file

/**

* *Description*

* *@return* *Description*

* *@param1* *Variable* *Description*

* *@exception* *Exception* *Description*

* [*@deprecated*]

* [*@see* *list of files/functions*]

above every function declaration, where *@deprecated* is used when the function exists only for backward compatibility and *@see* is used when this function has some relevance in another file

Line breaks/Concatenations

- No line breaks.
- SQL queries.
 sqlQuery = "SELECT ..."
 + "..."

XML Format

- Tags are pascal casing.
- Attributes are uppercase.

Example:

<Assignment TYPE="Apply">

Glossary

Pascal Casing

Word start with an upper case letter.

Camel Casing

Word starts with a lower case letter.

Subclassing

Is the same as inheritance.

Debugging an Activity

In this example, a PostActivationPremium transaction that causes an error runs, and the steps needed to debug the error, correct it, and run the transaction successfully are followed.

Policy	Client	Inquiry	Tables	Rules	Administration	System
Policy ⌨ ? ⓘ						
Quote Name: Policy Number: AIC123456789 Policy Status: Active Owner: User, Insurance		Policy Date: 07/15/2005 Company: Acme Insurance Company Plan: Acme VUL Issue State: CA				
Display: <input type="checkbox"/> Shadows <input type="checkbox"/> Reversals <input type="checkbox"/> Futures <input checked="" type="checkbox"/> Documents <input checked="" type="checkbox"/> Financial <input type="checkbox"/> Delete						<input type="button" value="Refresh"/> <input type="button" value="Close"/>
From Date: <input type="text"/>		To Date: 07/15/2005		Filters: (All Activities)		
Activity ⓘ						
Page 1 of 1		Page 1		Maximum Results: 100		
Activity: AccidentalDeathBenefitRiderAdc		<input type="button" value="Add"/>		<input type="checkbox"/> Auto-Process <input type="button" value="Process"/>		
Activity(11)	Activity Date	Status	Amount	Attachments	Action	
Anniversary	01/03/2005	Pending				
SevenPayAnniversary	01/03/2005	Pending				
AnnualStatementProjection	01/03/2005	Pending				
Monthiversary	02/03/2004	Pending				
FreeLookTransfer	02/02/2004	Pending				
PostActivationPremium	01/10/2004	Pending	\$500.00			
BillingStart	01/05/2004	Pending				
Monthiversary	01/05/2004	Active	(\$192.76)			
PremiumLimitCheck	01/05/2004	Active	(\$900.00)			
Issue	01/05/2004	Active	\$1,000,000.00			

When an error occurs, a description of the error is shown as a list of Objects, Methods, Descriptions, Files and Numbers. This is a stack trace.



exception

```
Caused By:
    Execution state: variableName=PolicyWasMEC
Caused By:
    Error occurred in source 'Main at line #112: "Checked" is not defined.
    if( MEC==Checked ) {
Caused By:
    ReferenceError: "Checked" is not defined. (Main#112)

    com.adminserver.utl.ASEExceptionUtl: Execution state: variableName=PolicyWasMEC
    at com.adminserver.utl.CalculateUtl.resolveCalculation(CalculateUtl.java:2277)
    at com.adminserver.utl.CalculateUtl.resolveCalculation(CalculateUtl.java:2199)
    at com.adminserver.bll.FinancialProcessorBll.doMath(FinancialProcessorBll.java:323)
    at com.adminserver.bll.FinancialProcessorBll.processFinancial(FinancialProcessorBll.java:80)
    at com.adminserver.bll.ActivityProcessorBllBean.processActivity(ActivityProcessorBllBean.java:324)
    at com.adminserver.bll.ActivityProcessorBllBean.initializeAndProcess(ActivityProcessorBllBean.java:935)
    at com.adminserver.bll.ActivityProcessorBllBean.processIndividual(ActivityProcessorBllBean.java:796)
    at com.adminserver.bll.ActivityProcessorBllBean.process(ActivityProcessorBllBean.java:234)
    at com.adminserver.bll.EJSRemoteStatelessActivityProcessorBll_0fc13901.process(Unknown Source)
    at com.adminserver.bll._ActivityProcessorBllRemote_Stub.process(Unknown Source)
    at com.adminserver.uip.ActivityActionUip.execute(ActivityActionUip.java:132)
    at org.apache.struts.action.RequestProcessor.processActionPerform(RequestProcessor.java:421)
    at org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:226)
```

A Java stack trace is a user-friendly snapshot of the threads and monitors in a Java Virtual Machine (JVM). A stack trace can range from fifty lines to thousands of lines of diagnostics.

Regardless of the size of the stack trace there are a few key things that you can find to help diagnose most software problems.

To diagnose the error, look at the first line within the stack trace in which a line number is present. In the above example the error occurs at line 2277 of `com.adminserver.url.CalculateUtl.resolveCalculation(Calculate.java)`.

Once you have determined the line number, you would then set up your debugging environment.

If you are running Jboss :

Windows:

1. Copy the run.bat file in jboss/bin folder and rename it *debug.bat*.
2. Right click on debug.bat and click "edit".
3. Copy the text between the "+" signs and paste, removing the existing code.

+++++

@echo off

rem -----

rem JBoss Bootstrap Script for Win32

rem -----

rem \$Id: run.bat,v 1.13.4.1 2004/12/15 16:52:20 starksm Exp \$

@if not "%ECHO%" == "" echo %ECHO%

@if "%OS%" == "Windows_NT" setlocal

set DIRNAME=.

if "%OS%" == "Windows_NT" set DIRNAME=%~dp0%

set PROGNAME=run.bat

if "%OS%" == "Windows_NT" set PROGNAME=%~nx0%

rem Read all command line arguments

REM

REM The %ARGS% env variable commented out in favor of using %* to include

REM all args in java command line. See bug #840239. [jpl]

```
REM

REM set ARGS=

REM :loop

REM if [%1] == [] goto endloop

REM     set ARGS=%ARGS% %1

REM     shift

REM     goto loop

REM :endloop


rem Find run.jar, or we can't continue


set RUNJAR=%DIRNAME%\run.jar

if exist "%RUNJAR%" goto FOUND_RUN_JAR

echo Could not locate %RUNJAR%. Please check that you are in the

echo bin directory when running this script.

goto END


:FOUND_RUN_JAR


if not "%JAVA_HOME%" == "" goto ADD_TOOLS


set JAVA=java


echo JAVA_HOME is not set. Unexpected results may occur.

echo Set JAVA_HOME to the directory of your local JDK to avoid this message.

goto SKIP_TOOLS


:ADD_TOOLS
```

```

set JAVA=%JAVA_HOME%\bin\java

if exist "%JAVA_HOME%\lib\tools.jar" goto SKIP_TOOLS

echo Could not locate %JAVA_HOME%\lib\tools.jar. Unexpected results may occur.

echo Make sure that JAVA_HOME points to a JDK and not a JRE.

:SKIP_TOOLS

rem Include the JDK javac compiler for JSP pages. The default is for a Sun JDK

rem compatible distribution to which JAVA_HOME points

set JAVAC_JAR=%JAVA_HOME%\lib\tools.jar

rem If JBOSS_CLASSPATH is empty, don't include it, as this will

rem result in including the local directory, which makes error tracking

rem harder.

if "%JBOSS_CLASSPATH%" == "" (
    set JBOSS_CLASSPATH=%JAVAC_JAR%;%RUNJAR%
) ELSE (
    set JBOSS_CLASSPATH=%JBOSS_CLASSPATH%;%JAVAC_JAR%;%RUNJAR%
)

rem Setup JBoss specific properties

set JAVA_OPTS=%JAVA_OPTS% -Dprogram.name=%PROGNAME%

set JBOSS_HOME=%DIRNAME%\..

rem Sun JVM memory allocation pool parameters. Modify as appropriate.

```

```
set JAVA_OPTS=%JAVA_OPTS% -Xms128m -Xmx512m
```

```
rem JPDA options. Uncomment and modify as appropriate to enable remote debugging.
```

```
rem SET JAVA_OPTS=-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=4000 %JAVA_OPTS%
```

```
rem Setup the java endorsed dirs
```

```
set JBOSS_ENDORSED_DIRS=%JBOSS_HOME%\lib\endorsed
```

```
echo =====
```

```
echo .
```

```
echo JBoss Bootstrap Environment
```

```
echo .
```

```
echo JBOSS_HOME: %JBOSS_HOME%
```

```
echo .
```

```
echo JAVA: %JAVA%
```

```
echo .
```

```
echo JAVA_OPTS: %JAVA_OPTS%
```

```
echo .
```

```
echo CLASSPATH: %JBOSS_CLASSPATH%
```

```
echo .
```

```
echo =====
```

```
echo .
```

```
:RESTART
```

```
"%JAVA%" %JAVA_OPTS% "-Djava.endorsed.dirs=%JBOSS_ENDORSED_DIRS%" -classpath "%JBOSS_CLASSPATH%"  
org.jboss.Main %"
```

```
IF ERRORLEVEL 10 GOTO RESTART
```

:END

if "%NOPAUSE%" == "" pause

:END_NO_PAUSE

+++++

1. Save the file. You will now be able to run this when debug mode is desired.

*nix/OS X:

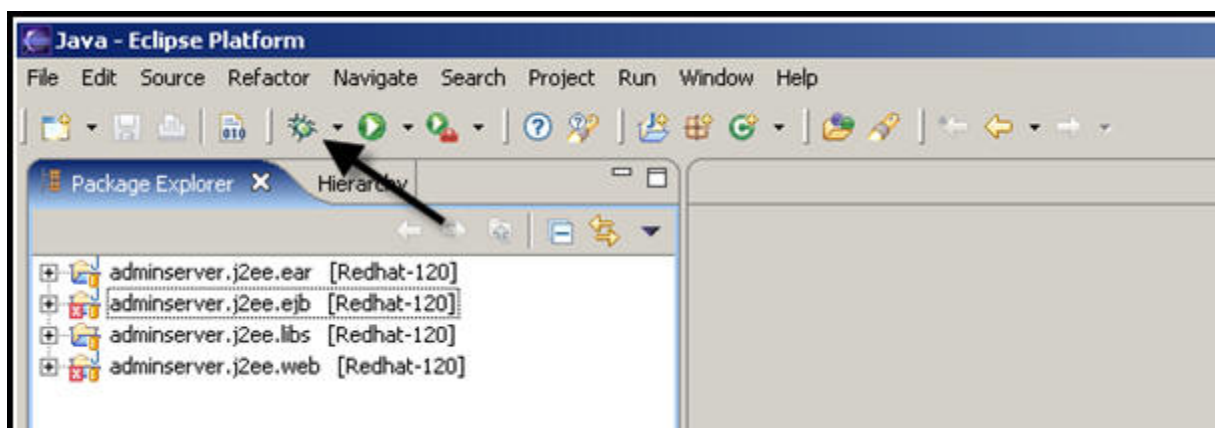
1. Copy the *run.sh* file in jboss/bin folder and rename it *debug.sh*.
2. Add the following command to the debug.sh file:

```
# Setup Debugging
JAVA_OPTS="$JAVA_OPTS -Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=4000"
```

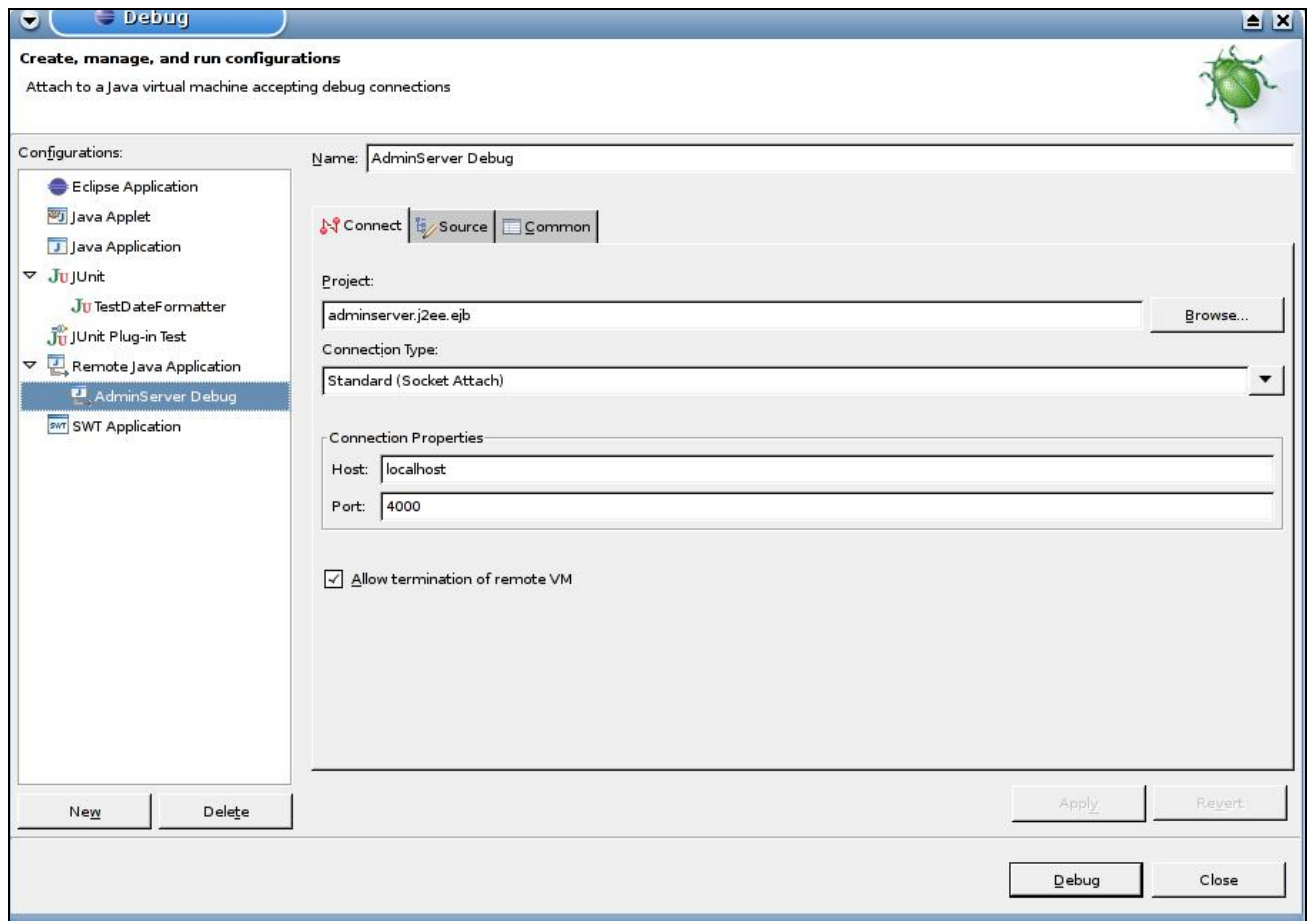
1. Save the file. You will now be able to run this when debug mode is desired.

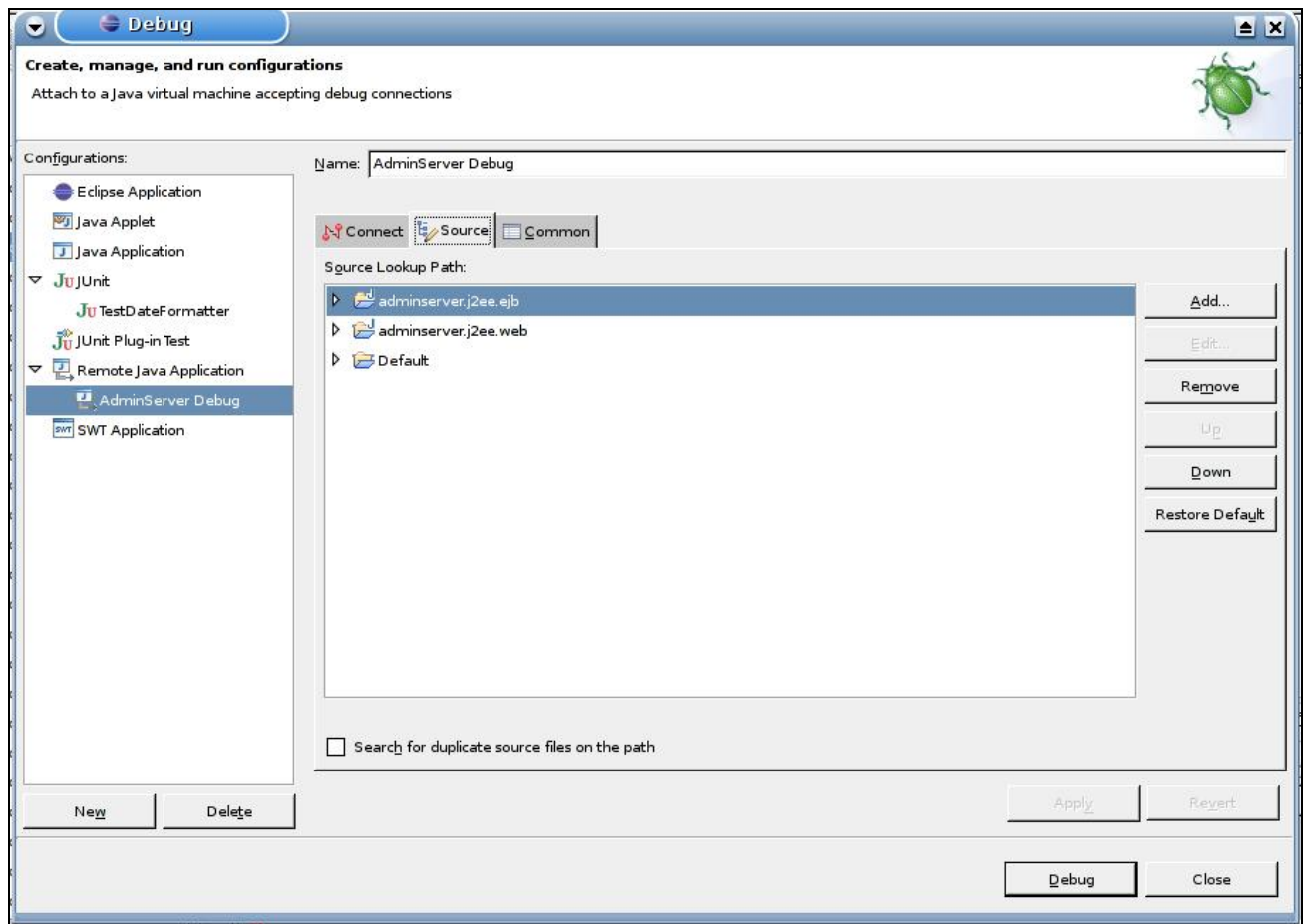
Open Eclipse

1. Click the down arrow next to the debug icon on the main button group.

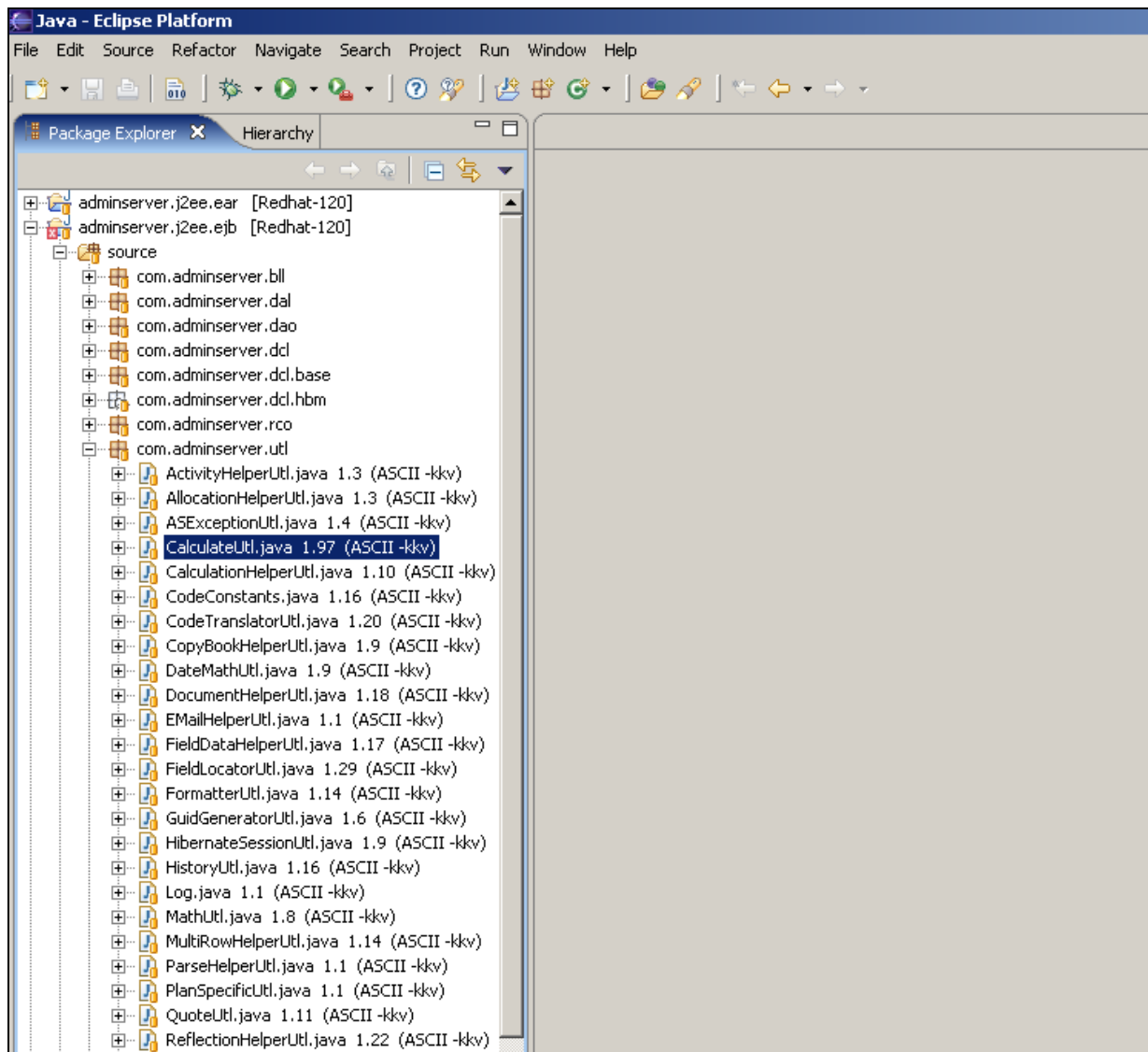


2. Select **Debug**.
3. Select **Remote Java Application** and select **New** on the left.
4. Fill out the fields as shown in the following two screenshots.

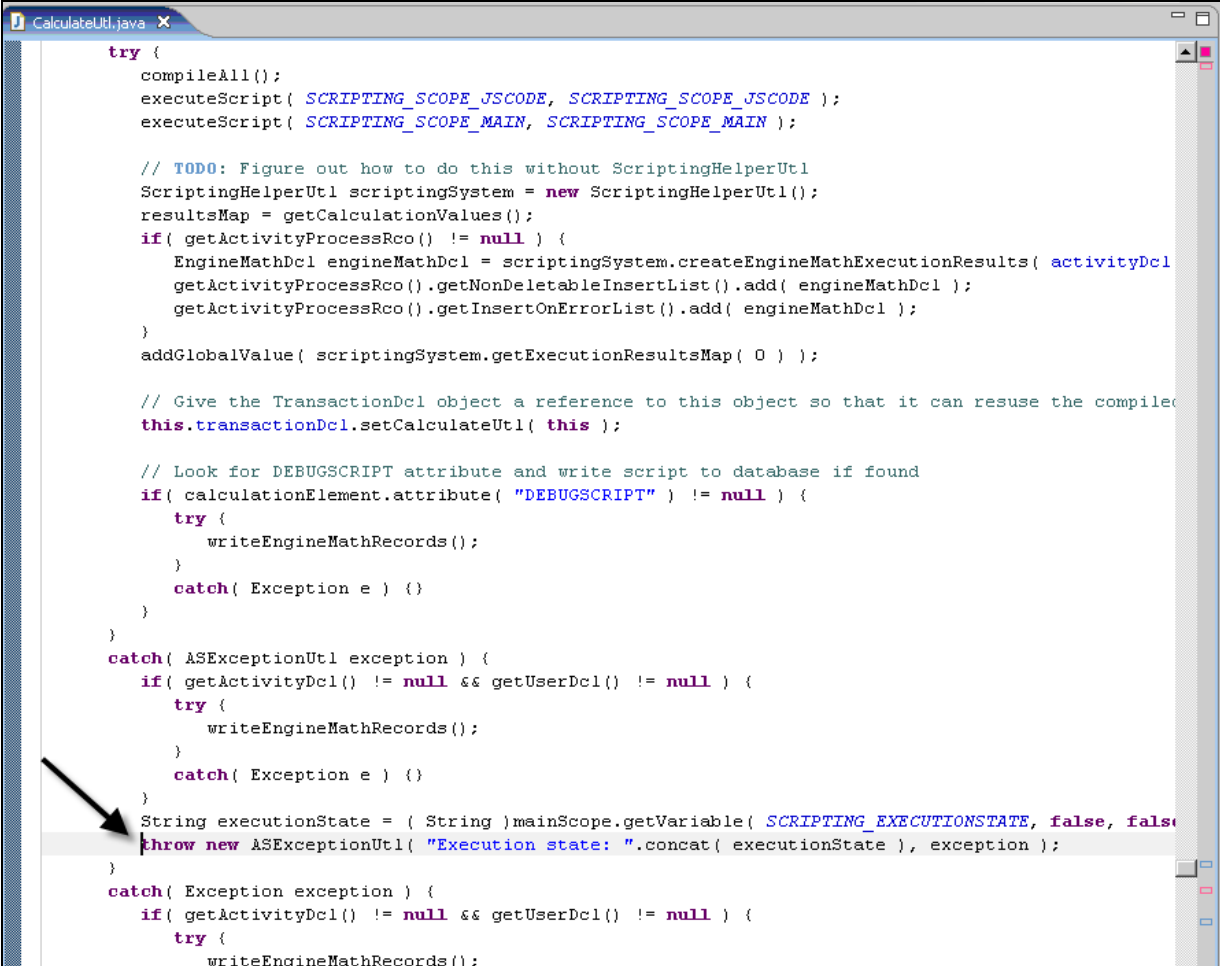




5. Select **Close**.
6. Follow the path in the stack trace and select the file name.



7. Double-click to open the file. When the file opens, scroll to the line number in the stack trace and bring focus to the line.



```

CalculateUtl.java
try {
    compileAll();
    executeScript( SCRIPTING_SCOPE_JSCODE, SCRIPTING_SCOPE_JSCODE );
    executeScript( SCRIPTING_SCOPE_MAIN, SCRIPTING_SCOPE_MAIN );

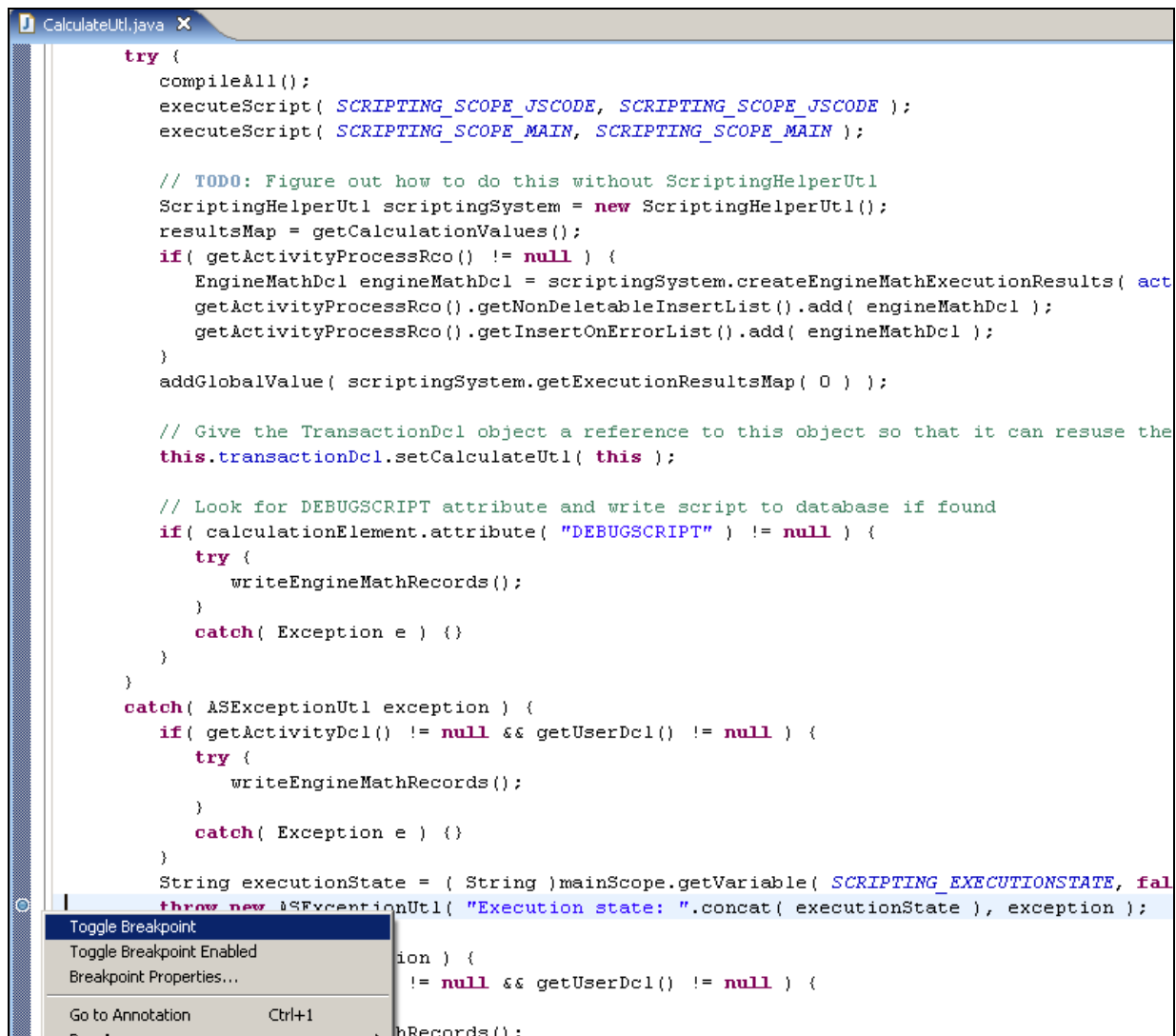
    // TODO: Figure out how to do this without ScriptingHelperUtl
    ScriptingHelperUtl scriptingSystem = new ScriptingHelperUtl();
    resultsMap = getCalculationValues();
    if( getActivityProcessRco() != null ) {
        EngineMathDcl engineMathDcl = scriptingSystem.createEngineMathExecutionResults( activityDcl
        getActivityProcessRco().getNonDeletableInsertList().add( engineMathDcl );
        getActivityProcessRco().getInsertOnErrorList().add( engineMathDcl );
    }
    addGlobalValue( scriptingSystem.getExecutionResultsMap( 0 ) );

    // Give the TransactionDcl object a reference to this object so that it can reuse the compiled
    this.transactionDcl.setCalculateUtl( this );

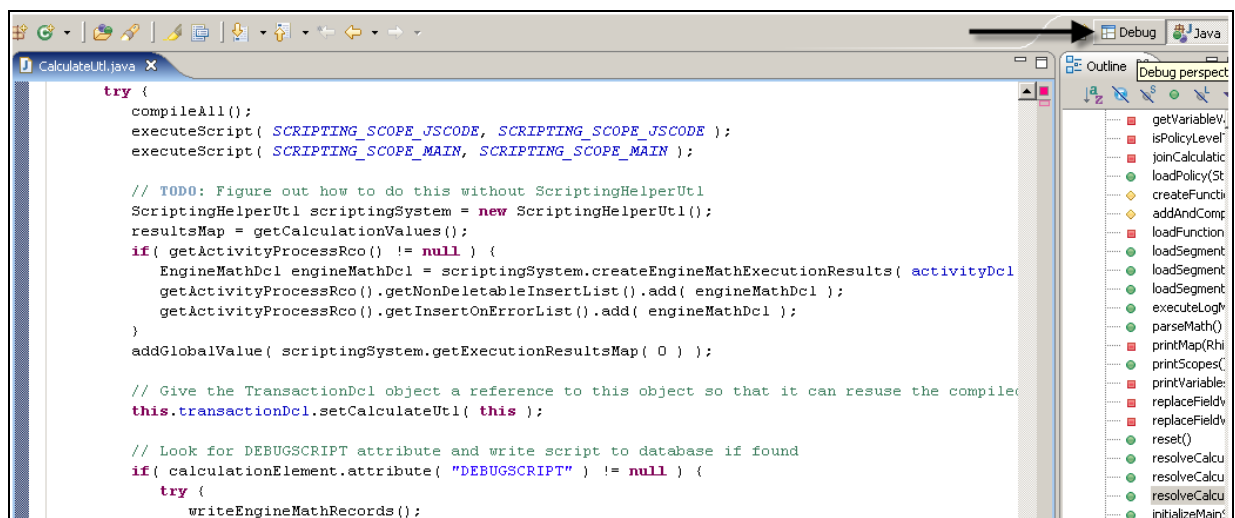
    // Look for DEBUGSCRIPT attribute and write script to database if found
    if( calculationElement.attribute( "DEBUGSCRIPT" ) != null ) {
        try {
            writeEngineMathRecords();
        }
        catch( Exception e ) {}
    }
}
catch( ASEExceptionUtl exception ) {
    if( getActivityDcl() != null && getUserDcl() != null ) {
        try {
            writeEngineMathRecords();
        }
        catch( Exception e ) {}
    }
    String executionState = ( String )mainScope.getVariable( SCRIPTING_EXECUTIONSTATE, false, false );
    throw new ASEExceptionUtl( "Execution state: ".concat( executionState ), exception );
}
catch( Exception exception ) {
    if( getActivityDcl() != null && getUserDcl() != null ) {
        try {
            writeEngineMathRecords();
        }
    }
}

```

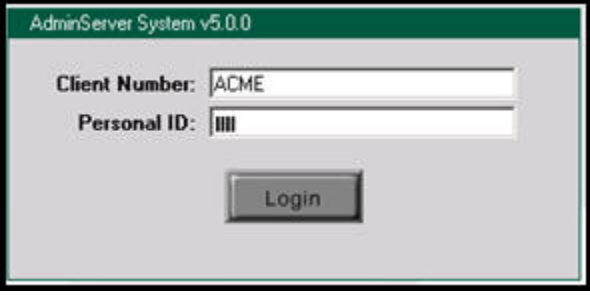
8. Right click on the line and select **Toggle breakpoint**.



9. Once the breakpoint has been set, select **Debug** in the upper right hand corner.



10. Close all browser sessions in which the OIPA system is open.
11. Open a new browser and log into the system.



The screenshot shows a web browser window titled "AdminServer System v5.0.0". The window has a light gray background. On the left side, there are two labels: "Client Number:" and "Personal ID:". To the right of "Client Number:" is a text input field containing the text "ACME". To the right of "Personal ID:" is a text input field containing three vertical bars "|||". Below these two input fields is a rectangular button with the text "Login" centered on it.

12. Process the transaction again and the code will step through debug, allowing you to follow the path of processes to get to that point.

SQL Server 2000 (if applicable)

Microsoft SQL Server Useful Utilities:

- Query Analyzer
- Enterprise Manager
- Profiler

Query Analyzer

Starting Query Analyzer

Go to Start/Programs/Microsoft SQL Server/Query Analyzer.

Or, from Run in the Windows Start menu, type:

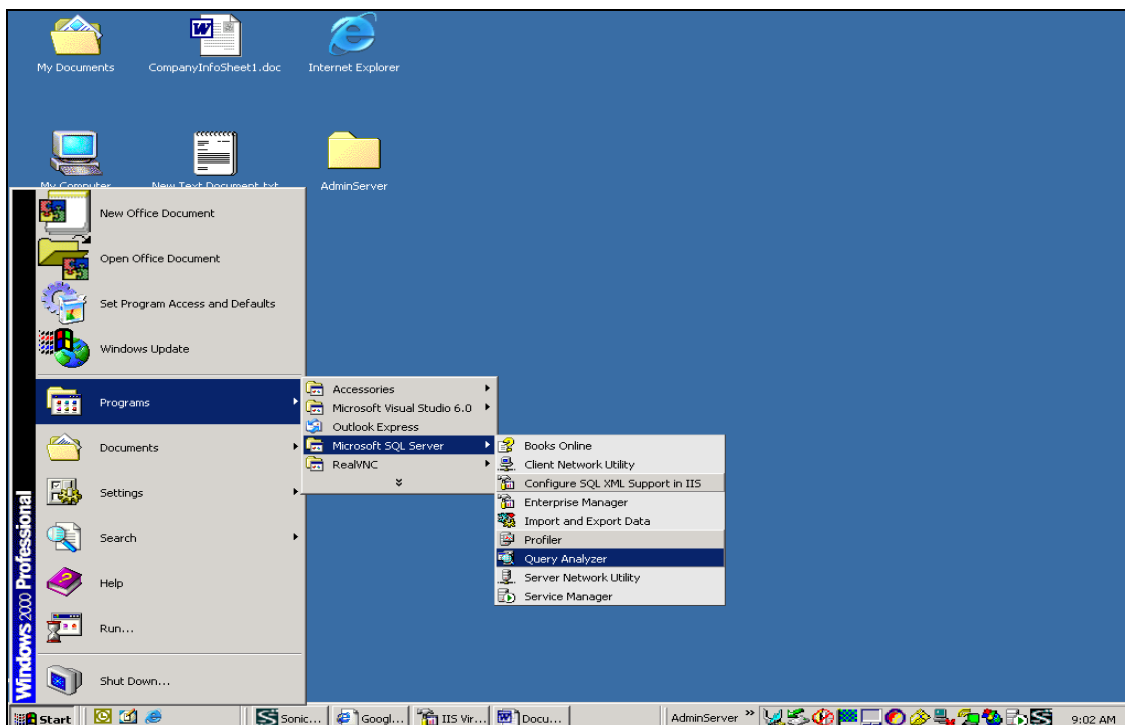
`iSqlW -SSqlServerName -dAdminServer -E`

Where:

SqlServerName is the name of the machine SQL Server is running on

AdminServer is the name of the database and

-E Uses Trusted connection (Windows login credentials)

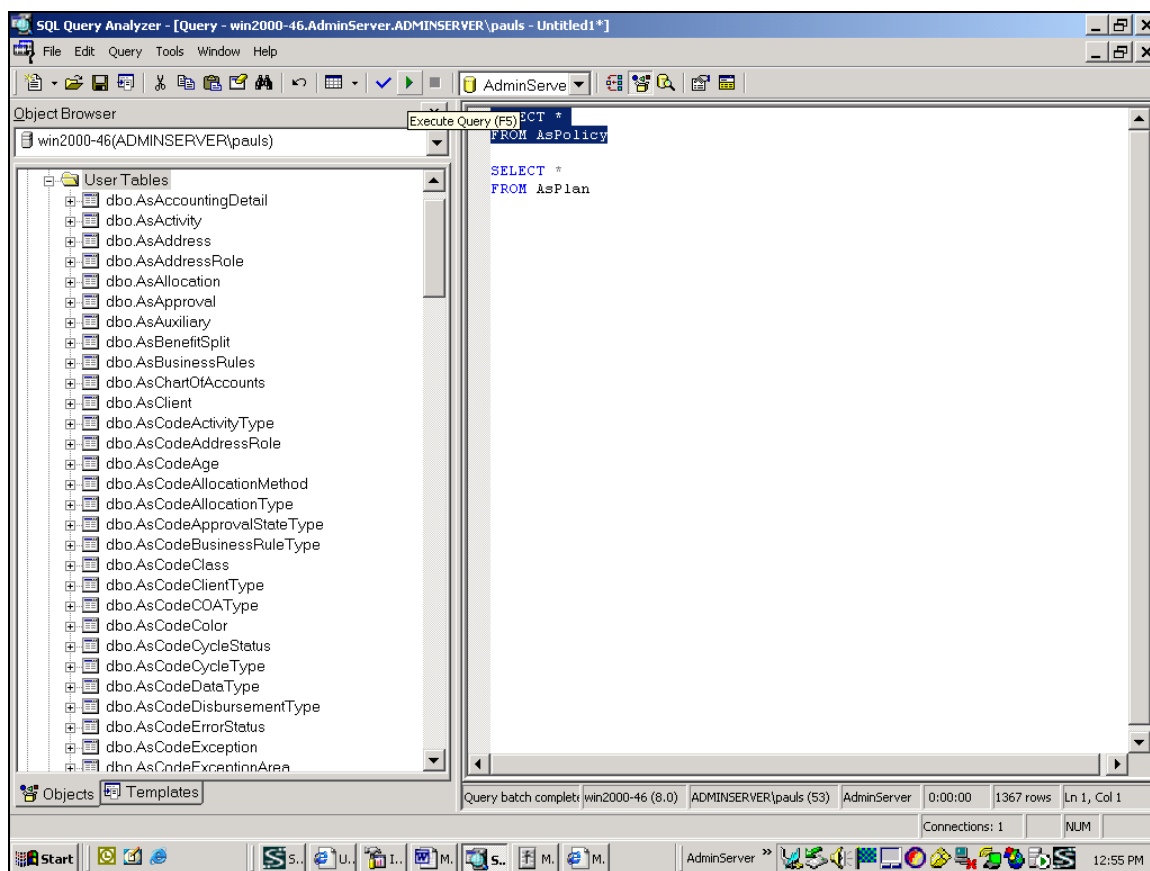


Ad-hoc querying (basics and important settings)

Type the following query in the Query Window in SQL Query Analyzer:

```
SELECT *
FROM AsPolicy
```

To execute the above query, select the query (highlight it) and push F5 or, to run all the queries in the Query Window, just press F5. The green play button in the toolbar will also execute the query.



Press F8 to close and open the Object Browser window, which lists all the tables, stored procedures, and functions in the OIPA database. The Object Browser also has a tab labeled Template that outlines most T-SQL statements.

The results window can be displayed as text (control T) or in a grid (control D).

Since the OIPA tables use the Text data type for XMLData columns, maximum characters per column should be set to 8192 - the largest number permitted by Query Analyzer. (See the Results tab under Tool and Options).

For more information on SQL Server:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/sqlserver2000.asp>

For more information on Templates:

<http://www.databasejournal.com/features/mssql/article.php/1560661>

Viewing the table structures

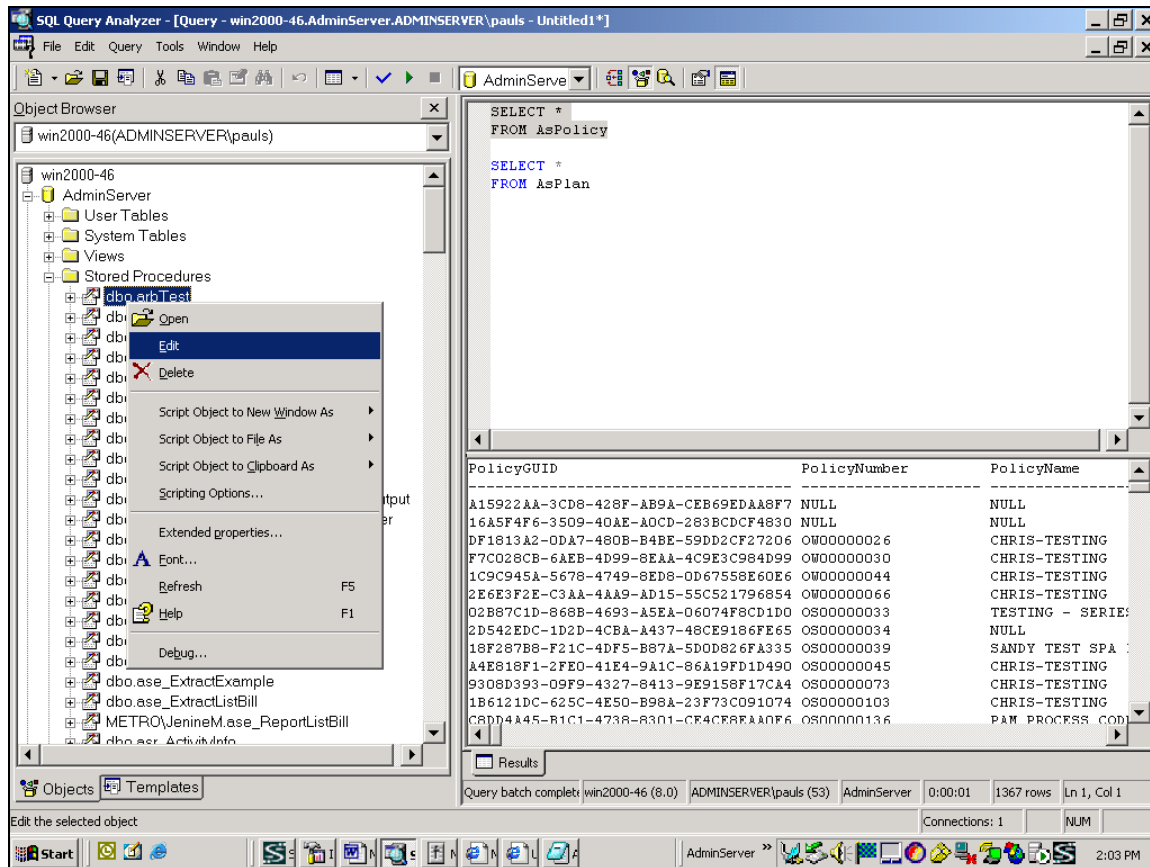
The definition of any of the OIPA tables can be viewed using Object Browser. The SQL Server system stored procedure call SP_HELP (Ex. SP_HELP AsClient) can also be used.

The screenshot displays the SQL Query Analyzer interface. On the left, the Object Browser shows the database structure for 'win2000-46'. The 'dbo.AsPolicy' table is expanded, showing its columns and their data types. The main query window contains a SQL query that joins 'AsPolicy', 'AsSegment', and 'AsPlan' tables, filtering for a specific 'PolicyNumber'. The results grid at the bottom shows 16 rows of data, with columns 'TransactionName' and 'RuleName'.

TransactionName	RuleName
1	LoanCapitalization
2	LoanCapitalization
3	LoanCapitalization
4	PolicyAnniversary
5	PolicyAnniversary
6	PolicyAnniversary
7	SystematicWithdrawalStart
8	SystematicWithdrawalStart
9	SystematicWithdrawalStart
10	SystematicWithdrawalStart
11	ListBill
12	ListBill
13	ListBill
14	ConservationLetter
15	ConservationLetter
16	ConservationLetter

Altering and creating stored procedures and functions

Using the Object Browser, expand the Stored Procedures or Functions folder. Select a stored procedure, right click on it, and then select **Edit**.



Some basic OIPA queries

Show the Policy and its Segment information:

```
SELECT *
FROM AsPolicy
JOIN AsSegment
  ON AsPolicy.PolicyGUID = AsSegment.PolicyGUID
JOIN AsPlan
  ON AsPlan.PlanGUID = AsPolicy.PlanGUID
WHERE PolicyNumber = '[PolicyNumber]'
/* AsCode has helpful information */
```

Join in the AsSegmentName:

```
SELECT COUNT(*)
FROM AsPolicy
JOIN AsSegment
  ON AsPolicy.PolicyGUID = AsSegment.PolicyGUID
JOIN AsSegmentName
  ON AsSegment.SegmentNameGUID = AsSegmentName.SegmentNameGUID
```

All the roles on a given policy:

```
SELECT *
FROM AsClient
JOIN AsRole
  ON AsClient.ClientGUID = AsRole.ClientGUID
AND AsRole.PolicyGUID =
(
  SELECT PolicyGUID
  FROM AsPolicy
  WHERE PolicyNumber = '[PolicyNumber]'
)
-- AsCodeClientType and AsCodeRole has helpful information
```

All the roles on a given policy with Address information:

```
SELECT *
FROM AsClient
JOIN AsRole
  ON AsClient.ClientGUID = AsRole.ClientGUID
AND AsRole.PolicyGUID =
(
  SELECT PolicyGUID
  FROM AsPolicy
  WHERE PolicyNumber = '[PolicyNumber]'
)
JOIN AsAddressRole
  ON AsAddressRole.ClientGUID = AsClient.ClientGUID
JOIN AsAddress
  ON AsAddressRole.AddressGUID = AsAddress.AddressGUID
```


All Activities on a given policy:

```
SELECT *
FROM AsTransaction
JOIN AsActivity
  ON AsTransaction.TransactionGUID = AsActivity.TransactionGUID
JOIN AsPolicy
  ON AsActivity.PolicyGUID = AsPolicy.PolicyGUID
  AND AsPolicy.PolicyNumber = 'OW00000026'
JOIN AsAllocation
  ON AsActivity.ActivityGUID *= AsAllocation.RelatedGUID
```

All Transactions with business rules attached:

```
SELECT AsTransaction.TransactionName, AsBusinessRules.RuleName
FROM AsTransaction
JOIN AsBusinessRules
  ON AsTransaction.TransactionGUID = AsBusinessRules.TransactionGUID
ORDER BY AsTransaction.TransactionGUID
```

Enterprise Manager

Introduction

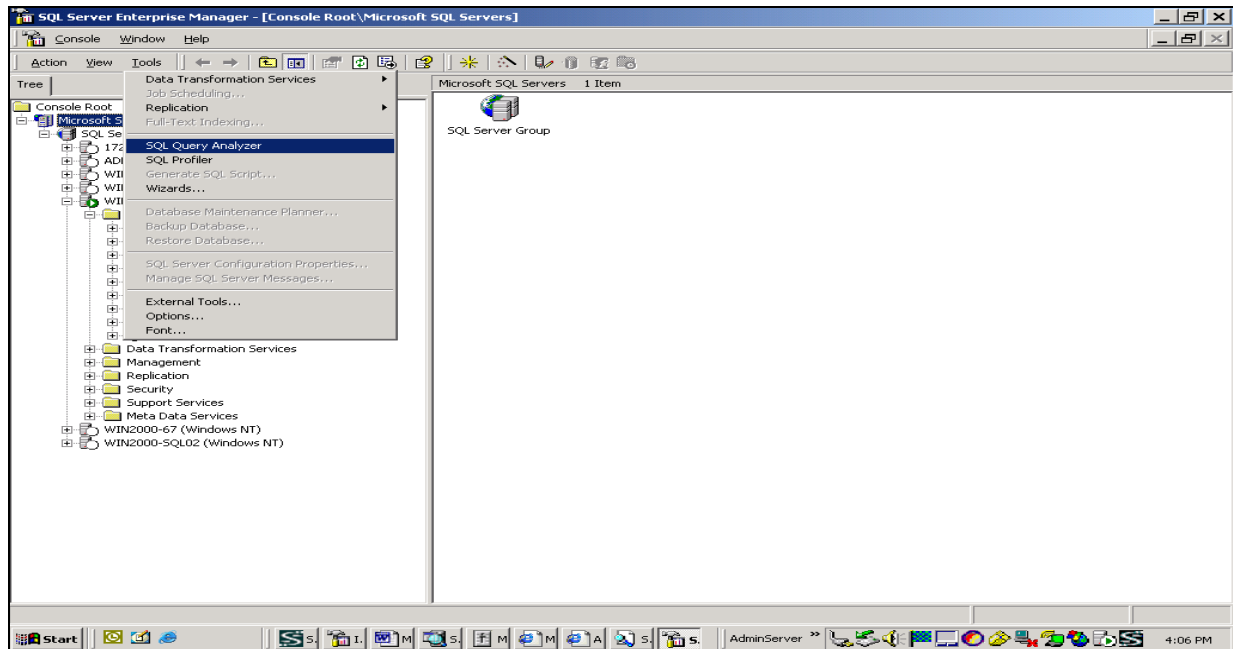
When first starting Enterprise Manager, the SQL Server with the OIPA database to the SQL Server Group must be added.

Below is a list of all the tables in the OIPA database on the machine named WIN2000-46:

Name	Owner	Type	Create Date
AsDetailSuspense	dbo	User	04/12/2003 6:25:01 PM
AsDisbursement	dbo	User	04/12/2003 6:25:01 PM
AsDocument	dbo	User	04/12/2003 6:25:01 PM
AsEngineMath	dbo	User	04/14/2003 10:16:33 AM
AsEngineSegment	dbo	User	04/12/2003 6:25:01 PM
AsErrorCatalog	dbo	User	04/12/2003 6:25:01 PM
AsExcelDetails	dbo	User	06/15/2003 3:23:08 PM
AsExchange	dbo	User	04/12/2003 6:25:01 PM
AsExtractLog	dbo	User	07/25/2003 10:31:57 AM
AsFilter	dbo	User	09/26/2003 2:49:16 PM
AsFund	dbo	User	04/12/2003 6:25:01 PM
AsFundGroup	dbo	User	04/12/2003 6:25:01 PM
AsFundWeight	dbo	User	04/12/2003 6:25:01 PM
AsHelpScreen	dbo	User	04/12/2003 6:25:01 PM
AsHierarchy	dbo	User	04/12/2003 6:25:01 PM
AsHistory	dbo	User	04/12/2003 6:25:01 PM
AsIllustration	dbo	User	04/12/2003 6:25:01 PM
AsImportLog	dbo	User	04/12/2003 6:25:01 PM
AsInquiryScreen	dbo	User	04/12/2003 6:25:01 PM
AsLinkRequirement	dbo	User	05/21/2003 9:34:45 AM
AsListBill	dbo	User	10/03/2003 11:54:52 AM
AsLookupRequirement	dbo	User	05/21/2003 9:34:45 AM
AsMapName	dbo	User	04/12/2003 6:25:01 PM
AsNetAssetValue	dbo	User	04/12/2003 6:25:01 PM
AsPlan	dbo	User	04/12/2003 6:25:01 PM
AsPocketPC	dbo	User	04/12/2003 6:25:01 PM
AsPolicy	dbo	User	04/12/2003 6:25:01 PM
AsPostallID	dbo	User	04/12/2003 6:25:01 PM
AsRate	dbo	User	07/19/2003 12:20:09 PM
AsRequirement	dbo	User	05/21/2003 9:34:45 AM
AsRole	dbo	User	04/12/2003 6:25:01 PM
AsRoleSequence	dbo	User	04/12/2003 6:25:01 PM
AsRollback	dbo	User	04/12/2003 6:25:01 PM
AsRollbackNUV	dbo	User	04/12/2003 6:25:01 PM
AsRollbackPolicy	dbo	User	04/12/2003 6:25:01 PM

Starting Query Analyzer

From the Tools menu, start SQL Query Analyzer or SQL Profiler.

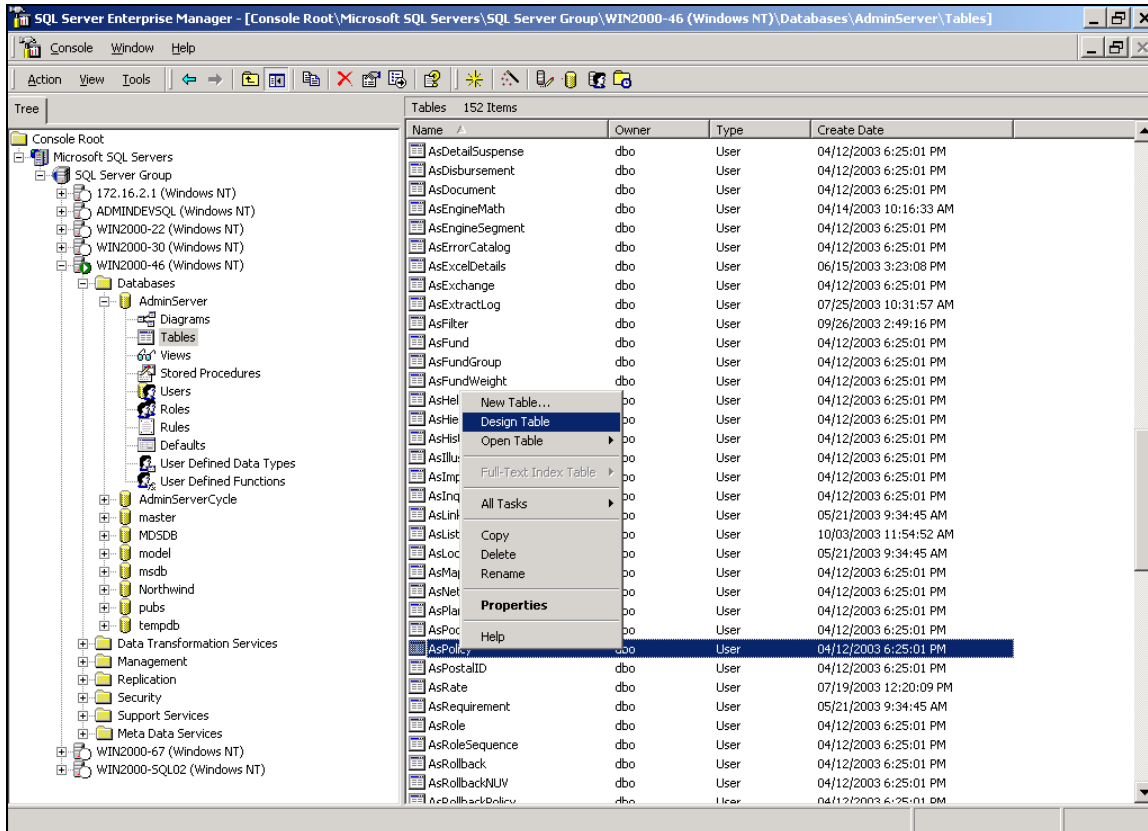


(SQL Profiler will be discussed later in this document.)

Altering and creating tables and indexes

Altering a Table

1. Right click a table.
2. Select **Design Table**.



The below Design Table window allows the user to alter AsPolicy without knowing the SQL syntax:

The screenshot shows the 'SQL Server Enterprise Manager - [2:Design Table 'AsPolicy' in 'AdminServer' on 'WIN2000-46']' window. The 'Columns' tab is selected, displaying a table with the following columns:

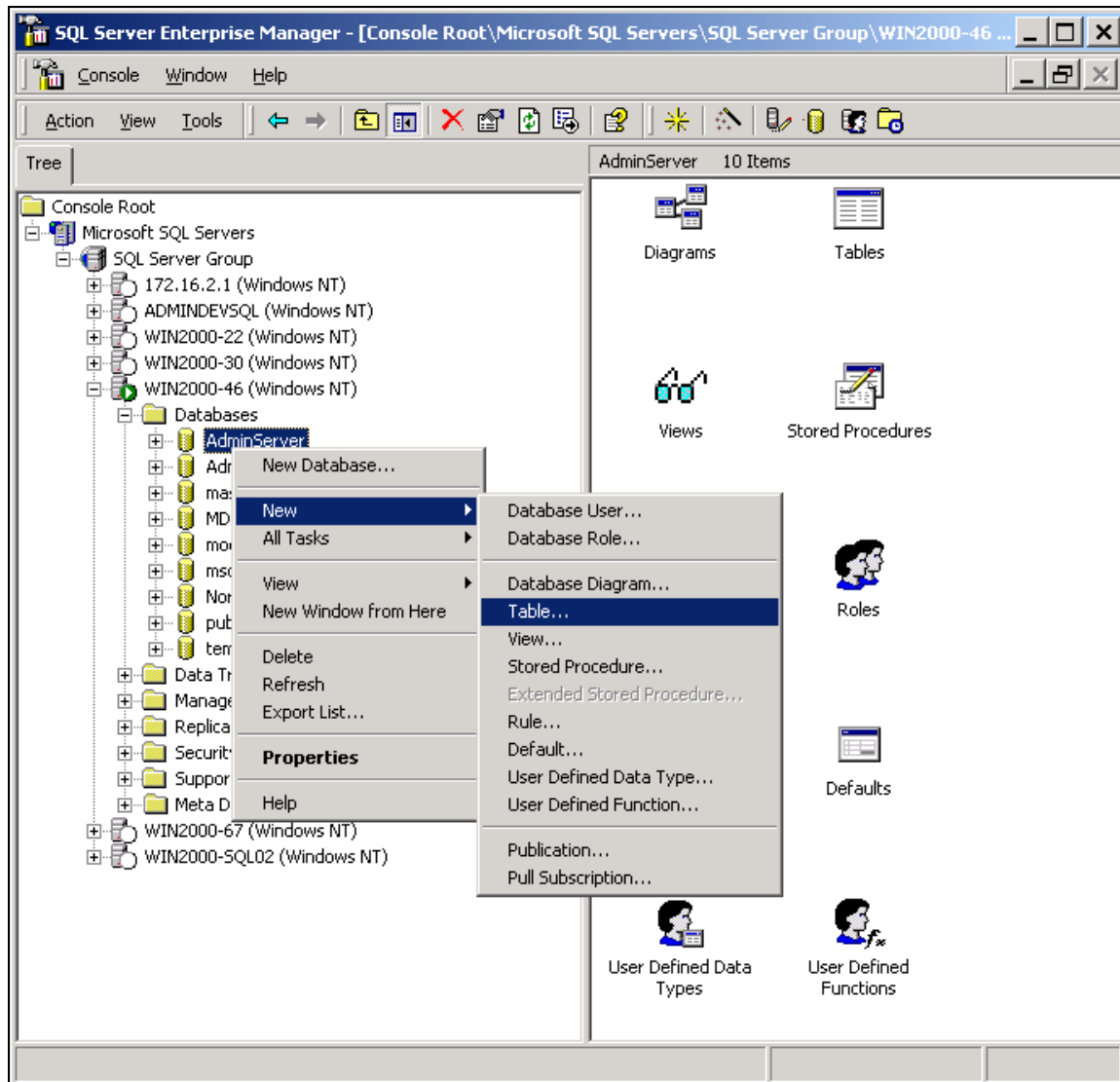
Column Name	Data Type	Length	Allow Nulls
PolicyGUID	uniqueident	16	
PolicyNumber	varchar	20	✓
PolicyName	varchar	50	✓
CreationDate	datetime	8	✓
IssueStateCode	varchar	2	✓
PlanDate	datetime	8	✓
StatusCode	varchar	2	✓
CompanyGUID	uniqueident	16	✓
PlanGUID	uniqueident	16	✓
XMLData	text	16	✓
UpdatedGMT	datetime	8	✓

Below the table, the 'Columns' tab is active, showing a list of properties for the selected column (PolicyGUID):

Description	
Default Value	(newid())
Precision	0
Scale	0
Identity	No
Identity Seed	
Identity Increment	
Is RowGuid	Yes
Formula	
Collation	

Creating a New Table

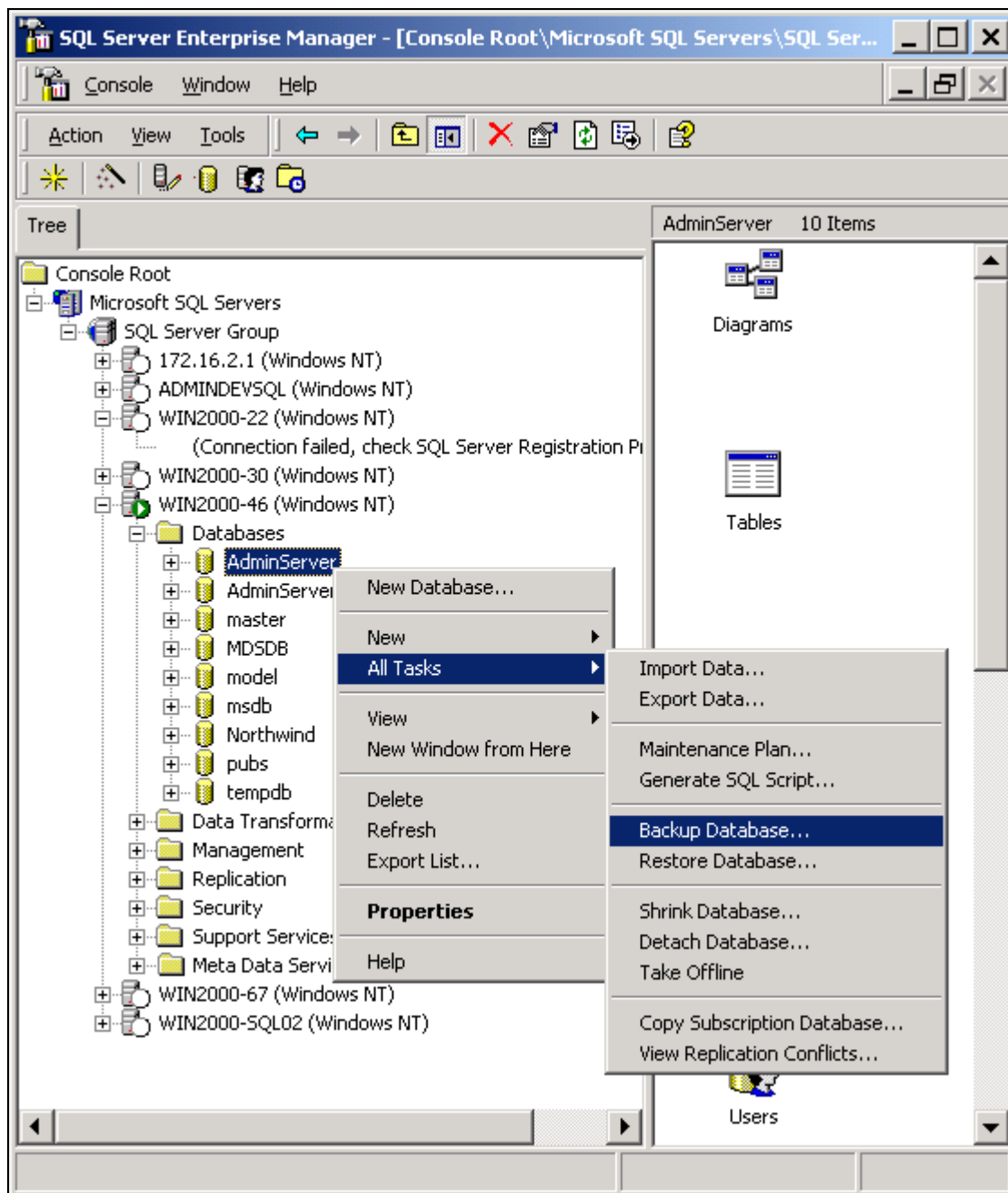
1. Right click on the database AdminServer.
2. Select New.
3. Select Table.



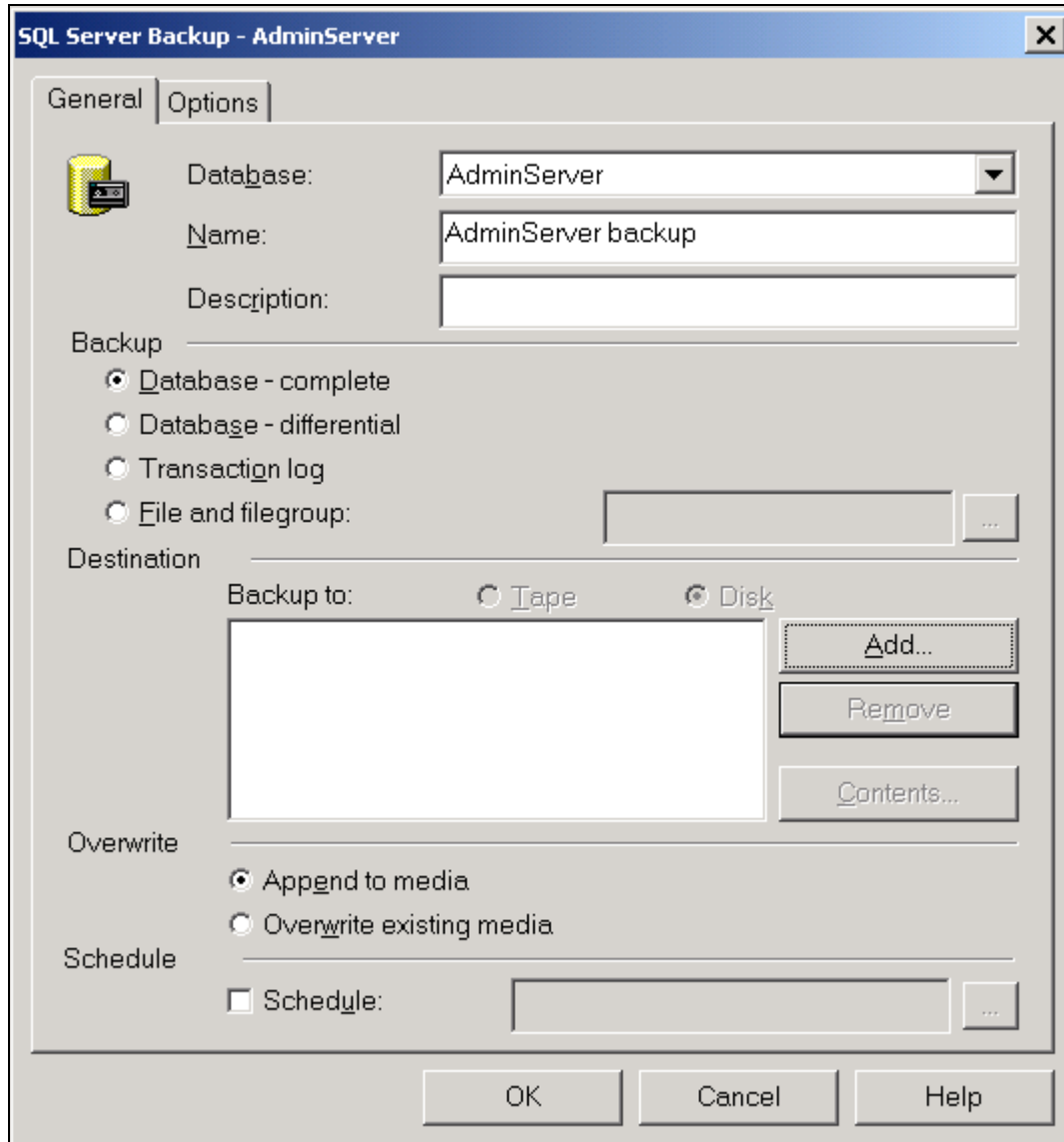
Backup and restoring

Backup:

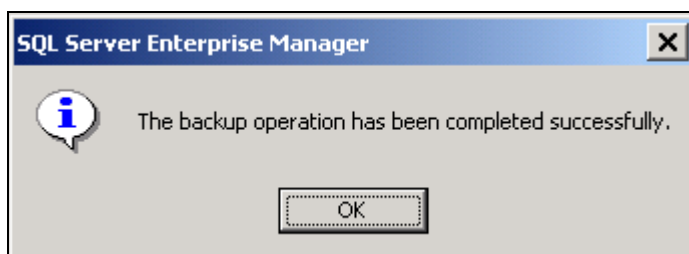
1. Right click the database
2. Select **All Tasks**.
3. Select **Backup Database**.



4. Give the backup a name and location (use the Add and Remove buttons on the General tab).
The Add button gives the backup a name and location.
5. Select **OK**.

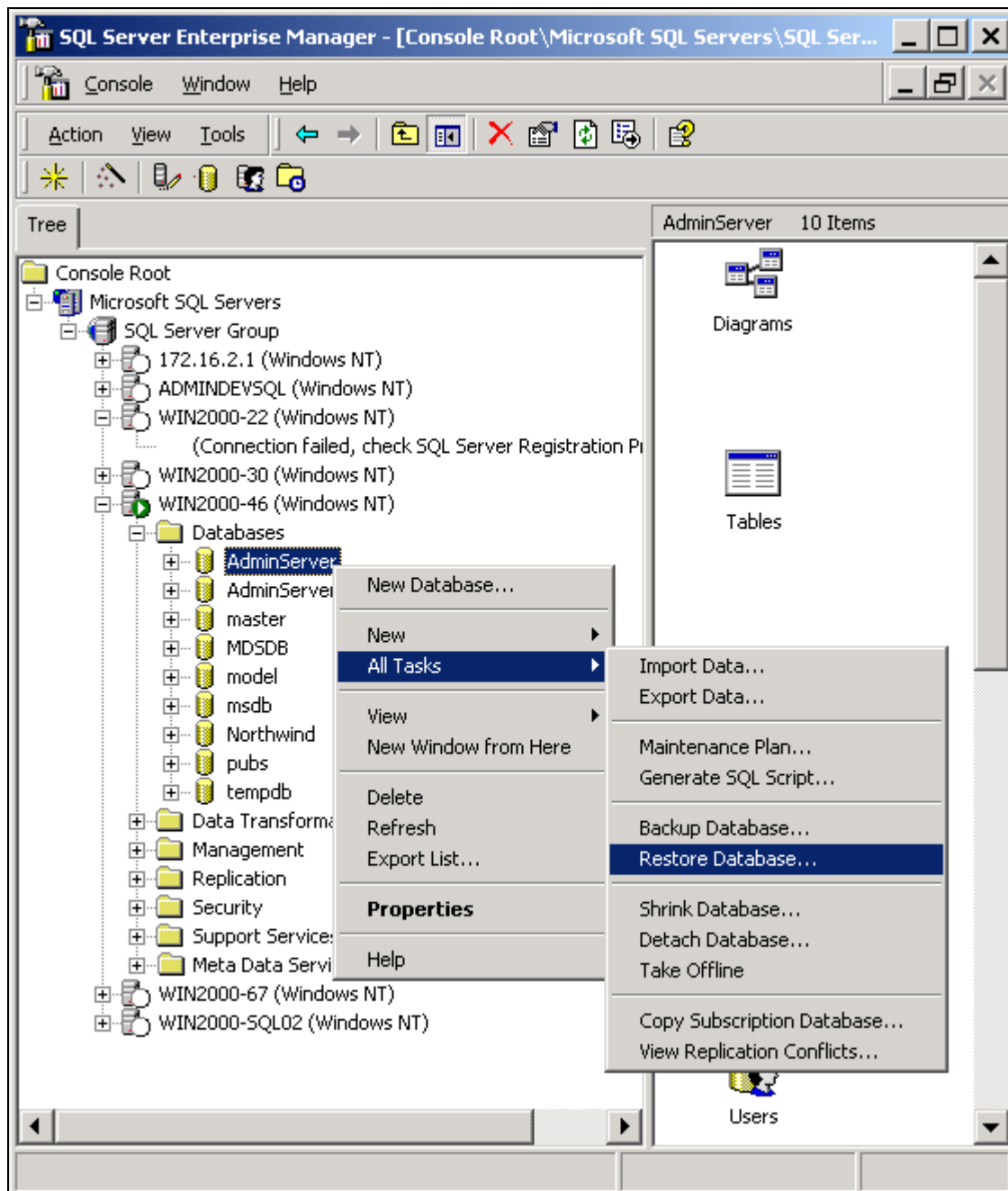


When the backup is complete, the following message will appear:

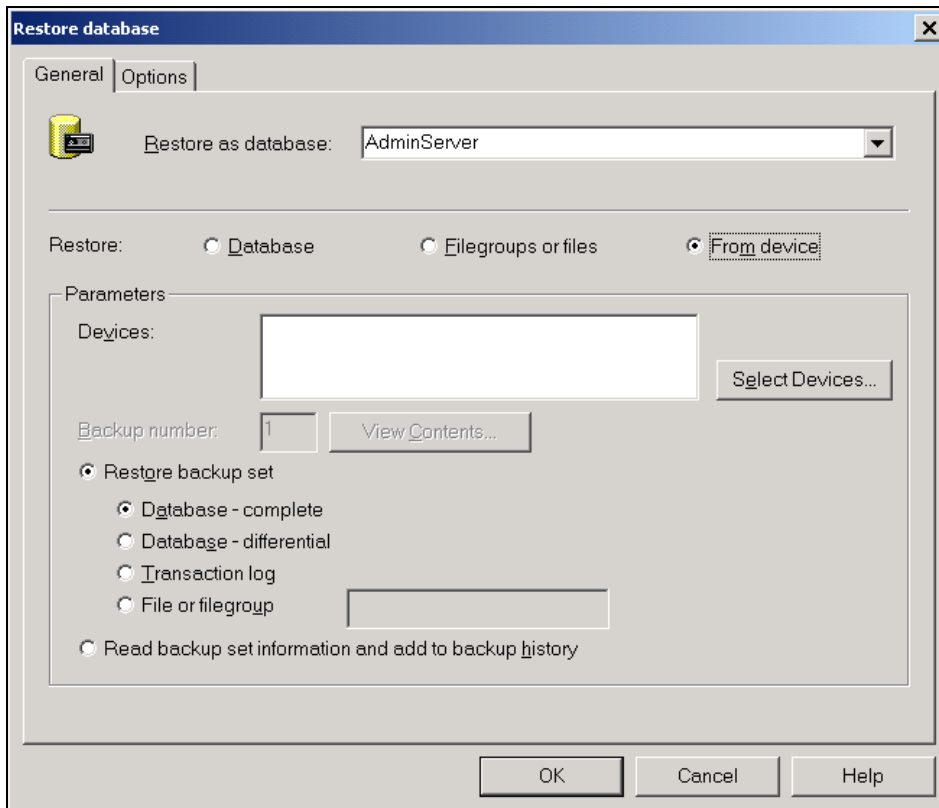


Restore a backup:

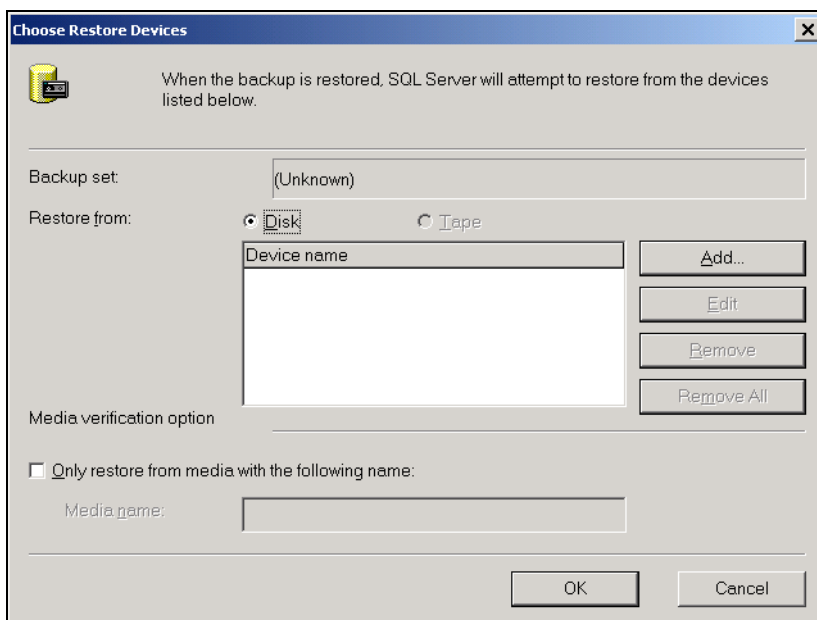
1. Right click the database and select **All Tasks**.
2. Select **Restore Database**.



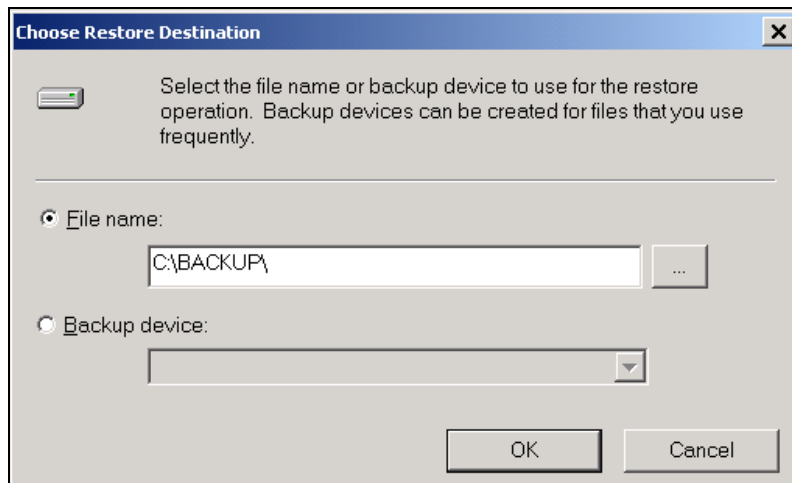
The Restore database window will appear.



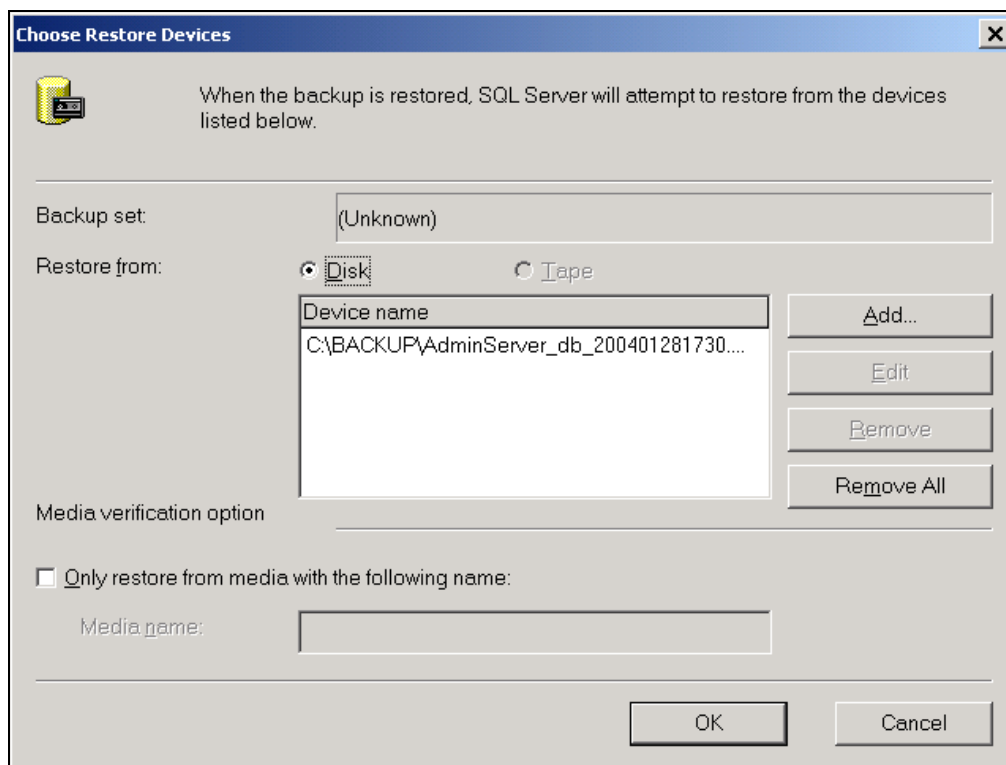
3. Select the radio button **From device**.
4. Select **Select Devices...** The window will appear.
5. Select **Add**.



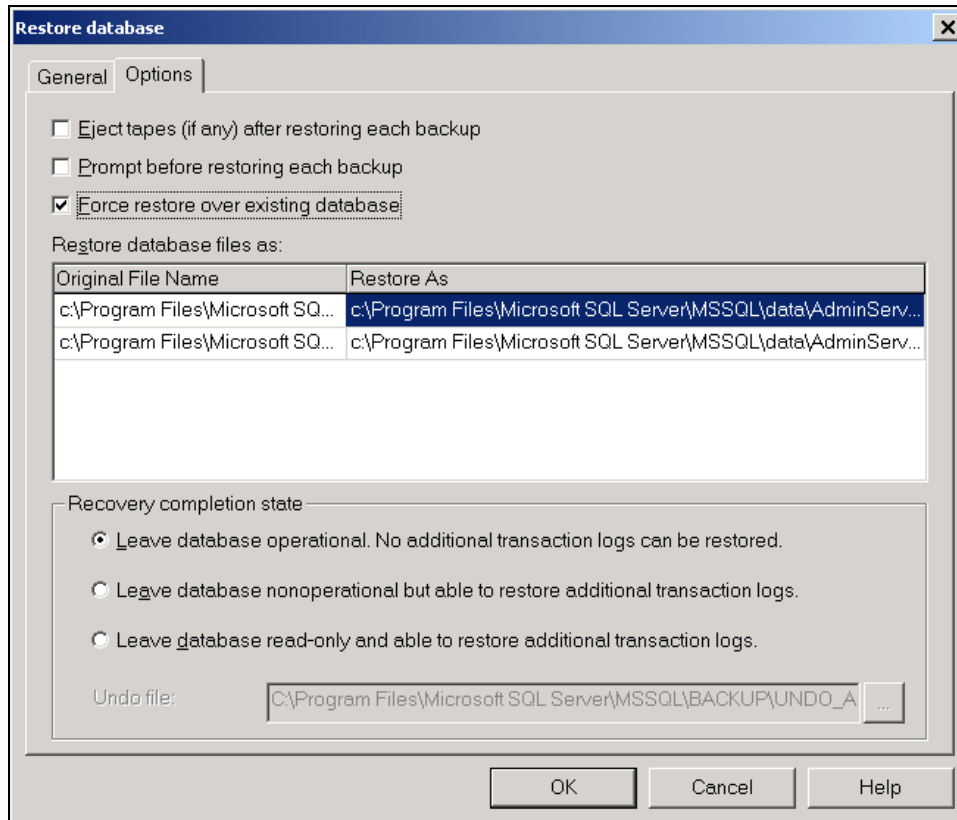
6. Choose the backup file to restore from.
7. Select **OK**.



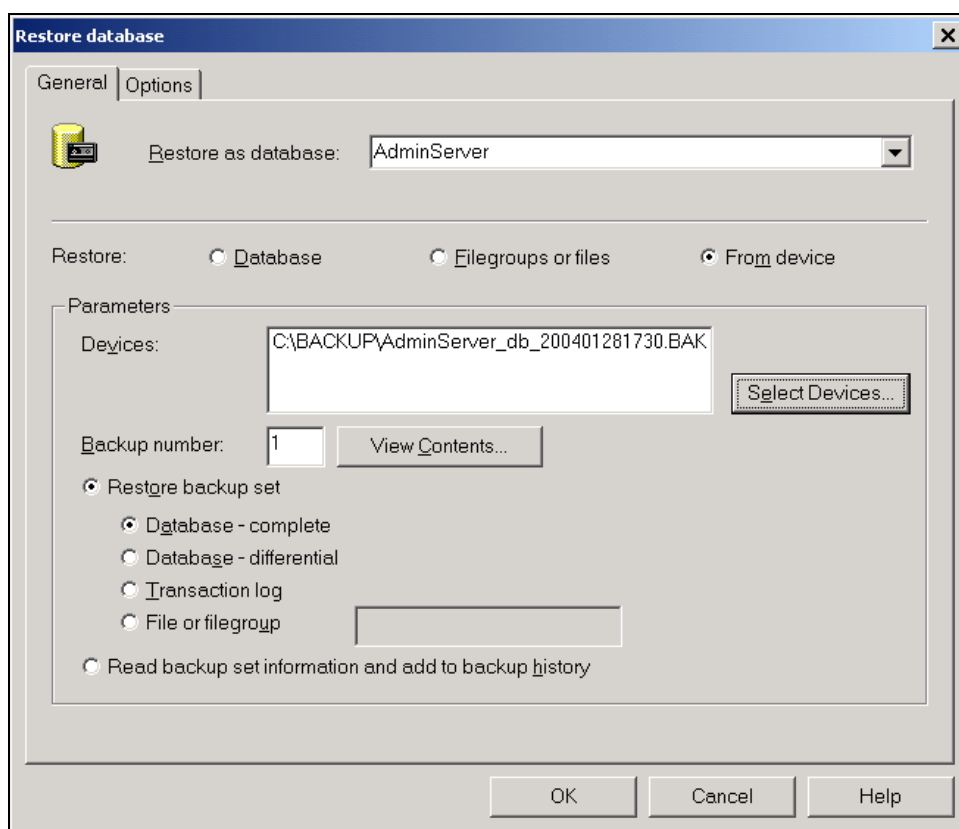
8. Click **OK** on the Choose Restore Devices window.



9. Select the **Options** tab.
10. Select **Force restore over existing database** and make sure the Restore As has the correct path for the data and log file for the SQL Server being restored to.



11. Select **OK** on the Restore database window.



The restore may take a few minutes to finish.

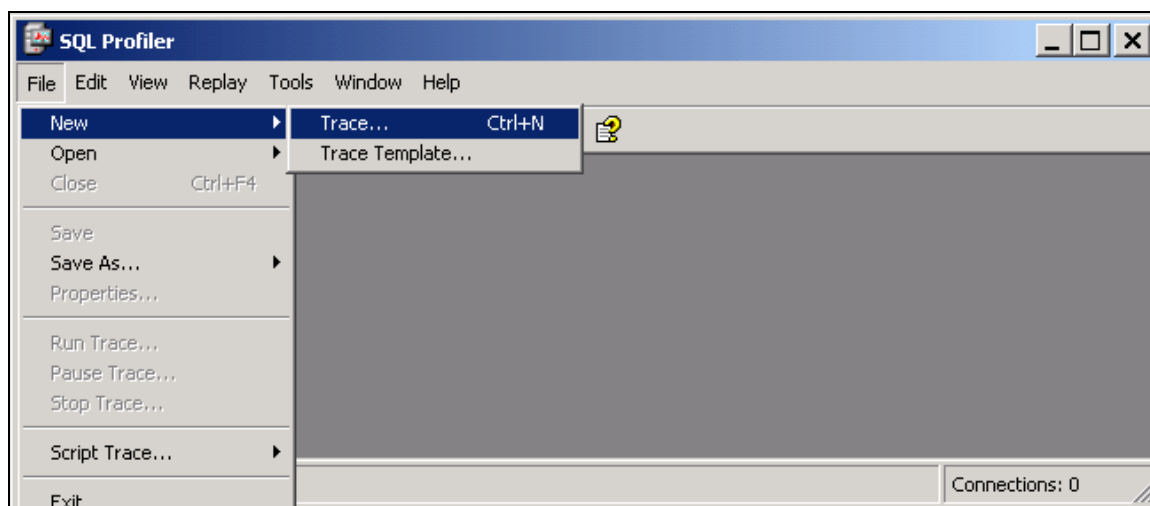
Profiler

As a debugging tool

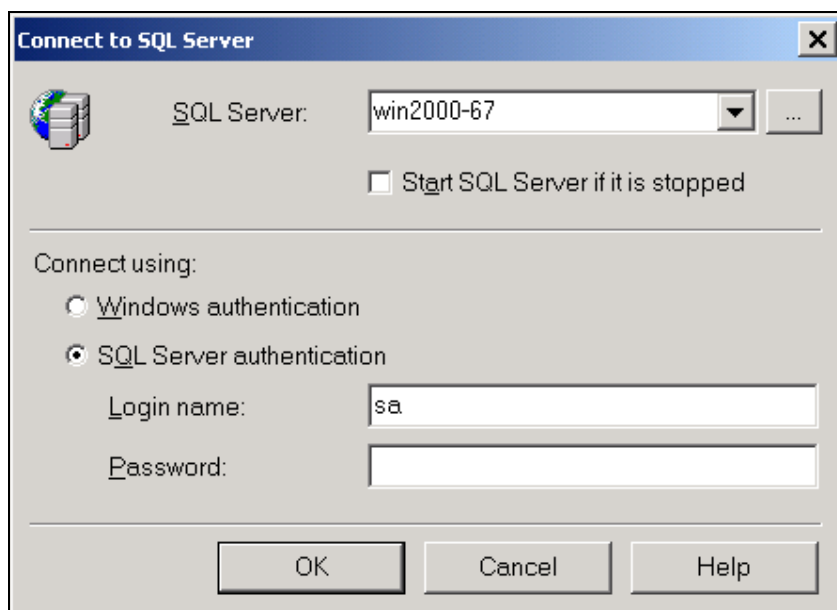
Profiler can be used to see the queries that are executed in a certain page (Ex. S3Policy.asp).

Start Profiler

1. Select **File**.
2. Select **New**.
3. Select **Trace...**



4. Choose the server and enter the login information.



5. On the Trace Properties window, select **Run**.
6. Choose Save to file or Save to table.

Trace Properties

General | Events | Data Columns | Filters

Trace name:

Trace SQL Server:

Use the following trace template:

Template name:

Template file name:

☐ Save to file:

Set maximum file size (MB):

☒ Enable file rollover

☐ Server processes SQL Server trace data

☐ Save to table:

☐ Set maximum rows (in thousands):

☐ Enable trace stop time:

A trace can be started, stopped, saved, or cleared at any time.

SQL Profiler - [Untitled - 2 (win2000-67)]									
File Edit View Replay Tools Window Help									
EventClass	Clear trace window	extData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Du
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT TOP 1 CompanyGUID, PlanGUID,...	Microsoft(R...	pauls	ADMIN...	0	195	0	0
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT AsCompany.* FROM AsSecurit...	Microsoft(R...	pauls	ADMIN...	0	95	0	0
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT TOP 1 CompanyGUID, PlanGUID,...	Microsoft(R...	pauls	ADMIN...	0	195	0	0
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT TOP 1 CompanyGUID, PlanGUID,...	Microsoft(R...	pauls	ADMIN...	0	10	0	0
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT TOP 5 AsPolicy.PolicyGUID, A...	Microsoft(R...	pauls	ADMIN...	109	574	0	10
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT AsCompany.* FROM AsSecurit...	Microsoft(R...	pauls	ADMIN...	0	95	0	0
RPC:Completed		exec sp_reset_connection	Microsoft(R...	pauls	ADMIN...	0	0	0	0
SQL:BatchCompleted		SELECT * FROM AsInquiryScreen WITH...	Microsoft(R...	pauls	ADMIN...	0	8	0	0
SQL:BatchCompleted		SELECT N'Testing Connection...'	SQLAgent - ...	SYSTEM	NT AU...	0	0	0	0
SQL:BatchCompleted		EXECUTE msdb.dbo.sp_sqlagent_get_pe...	SQLAgent - ...	SYSTEM	NT AU...	0	101	0	0
TraceStop									

SELECT TOP 1 CompanyGUID, PlanGUID, FundGUID, TransactionGUID, StateCode, ClientGUID, PolicyGUID, SegmentGUID, ActivityGUID, go
SELECT TOP 1 CompanyGUID, PlanGUID, FundGUID, TransactionGUID, StateCode, ClientGUID, PolicyGUID, SegmentGUID, ActivityGUID, go
SELECT TOP 5 AsPolicy.PolicyGUID, AsPolicy.PolicyNumber, AsPolicy.PolicyName, AsCodeStatus.ShortDescription, AsCompany.Compar go

Trace is stopped

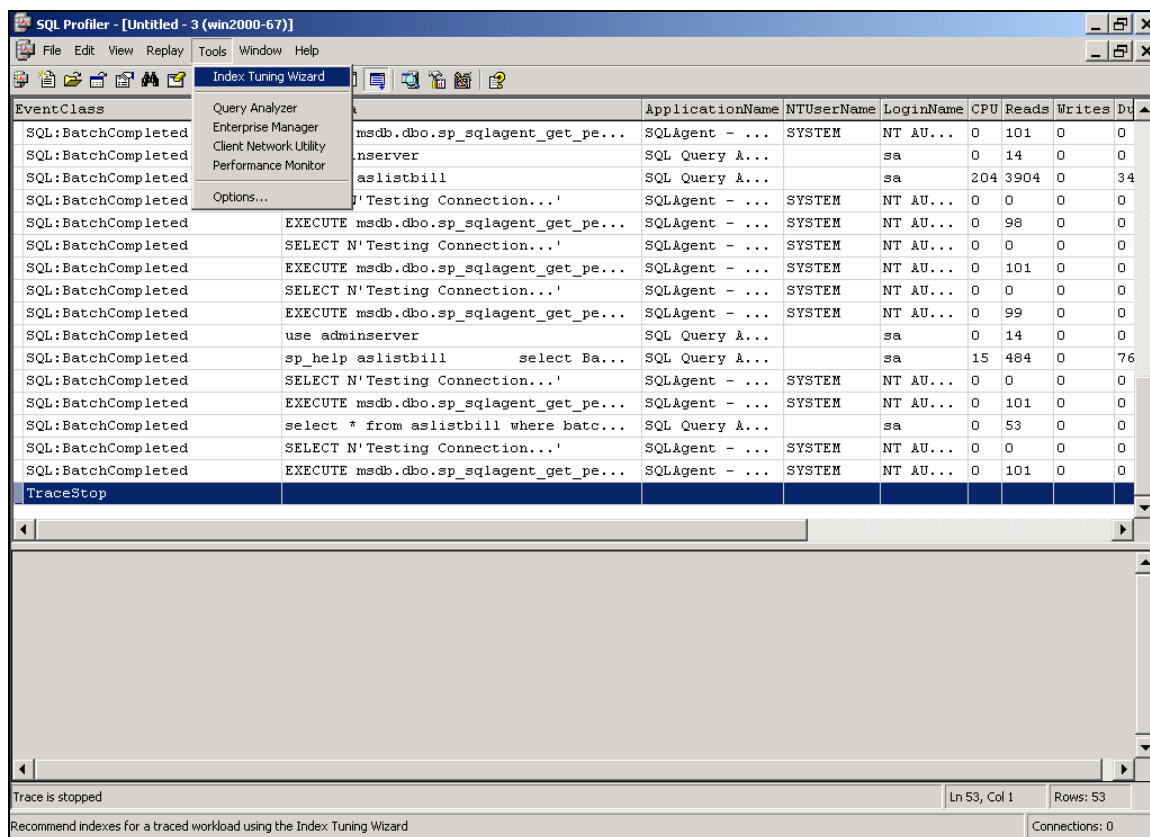
Ln 52, Col 2 Rows: 63 Connections: 0

Index tuning wizard

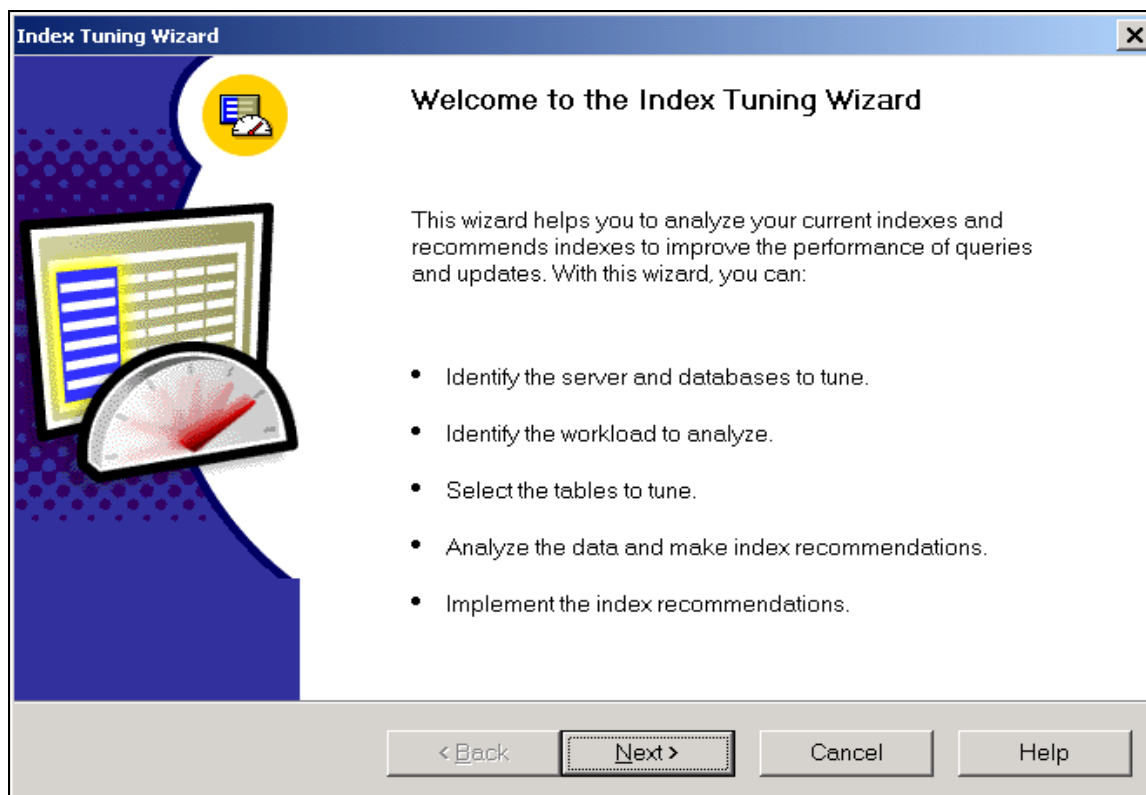
SQL Server Profiler and the index tuning wizard can be used to identify the tables in the database that may need indexes and will write the created index statements.

Index Tuning Wizard

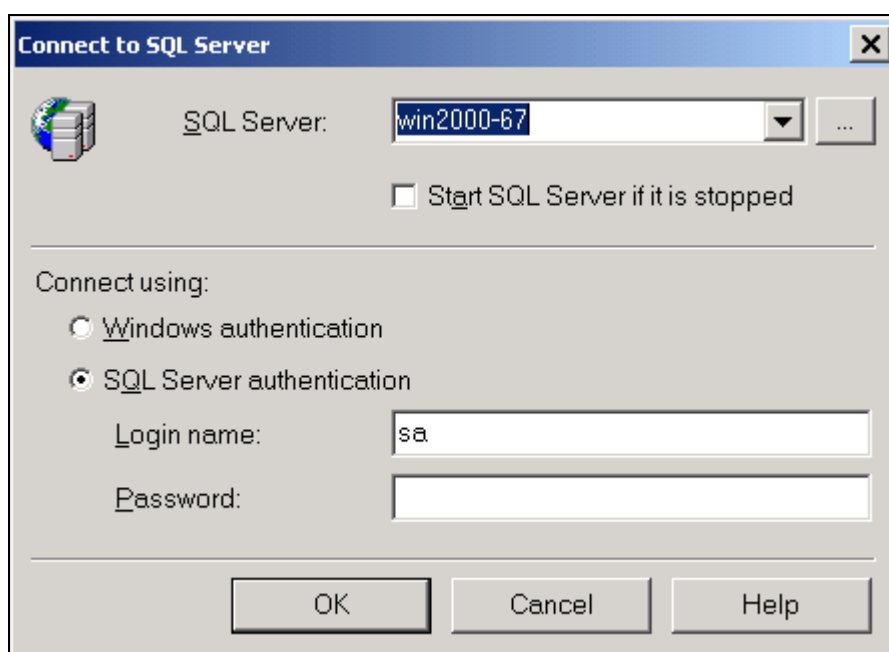
1. Start Profiler and run it, tracing the queries that are not performing well. When starting the Profiler trace, the Profiler must be configured to save the trace/log to a file(s) or to a database table.
2. When the Profiler has gathered enough sample SQL statements, select **Stop**.
3. Select **Tools**, then **Index Tuning Wizard**.



4. Select **Next** on the Index Tuning Wizard window.



5. Enter the server and user ID information.



6. On the Index Tuning Wizard window, select **Next**.

Index Tuning Wizard - win2000-67

Select Server and Database
Before the wizard can analyze your data, you must select the server and the database to tune.

Server: win2000-67

Database: AdminServer

☒ **Keep all existing indexes.**
Select this option to tune a workload of new or problem queries. Clearing this option will provide the maximum performance improvement for the workload being tuned. Indexes can be dropped or replaced if you do not keep existing indexes.

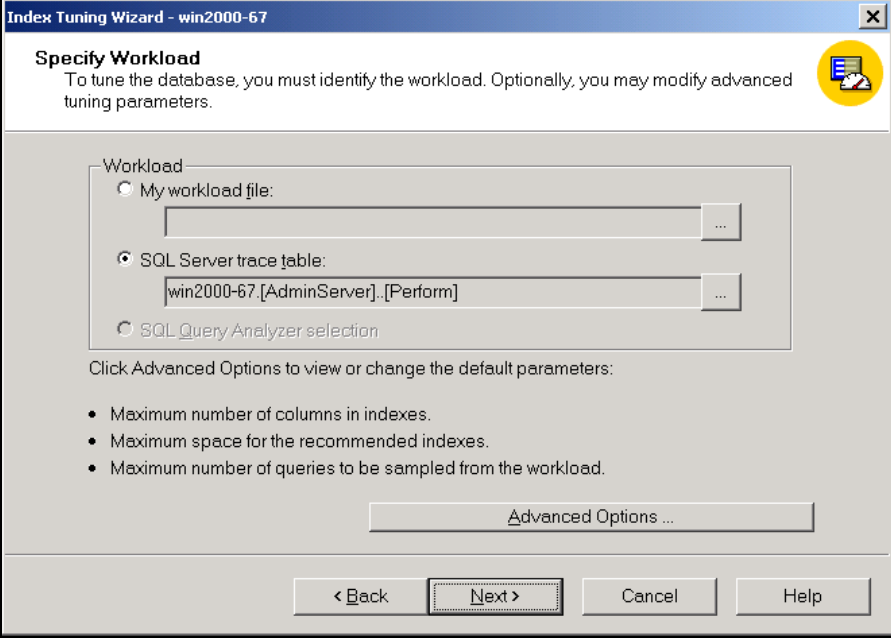
☒ **Add indexed views**

Tuning mode
A more thorough analysis requires more tuning time but results in a better recommendation of indexes

☐ Fast ☒ Medium ☐ Thorough

< Back Next > Cancel Help

7. Choose the file or trace table to which the poor performing queries were saved.
8. Select **Next**.



Specify Workload
To tune the database, you must identify the workload. Optionally, you may modify advanced tuning parameters.

Workload

☐ My workload file:

☒ SQL Server trace table:

☐ SQL Query Analyzer selection

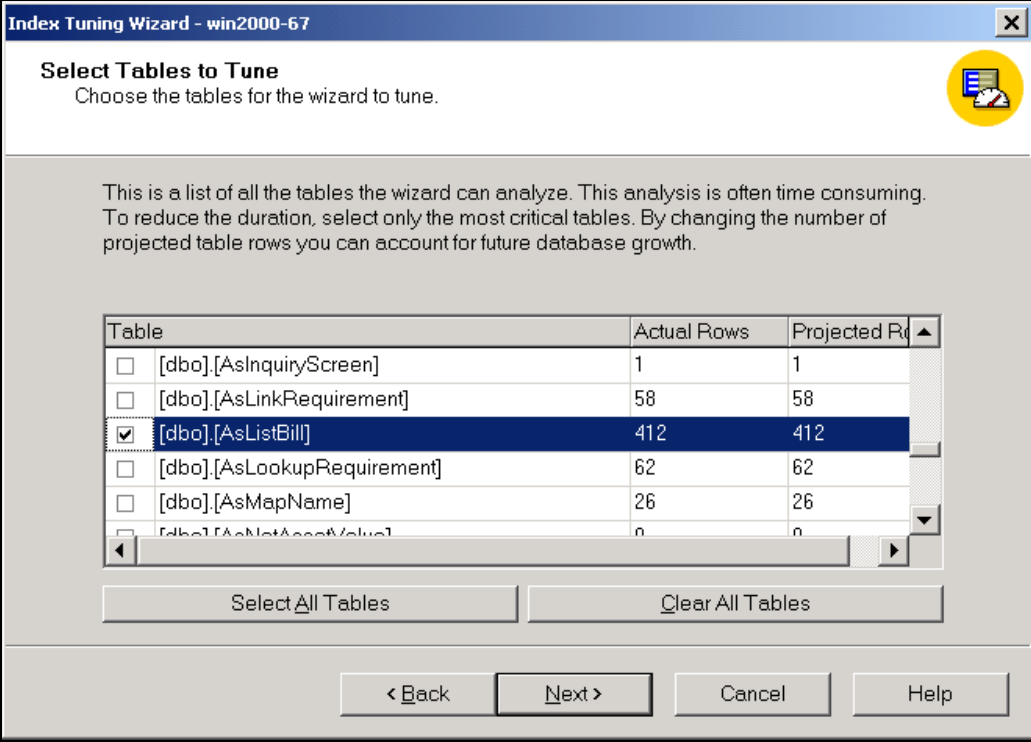
Click Advanced Options to view or change the default parameters:

- Maximum number of columns in indexes.
- Maximum space for the recommended indexes.
- Maximum number of queries to be sampled from the workload.

Advanced Options ...

< Back Next > Cancel Help

9. Select all the tables the tuning wizard should examine.
10. Select **Next**.



Select Tables to Tune
Choose the tables for the wizard to tune.

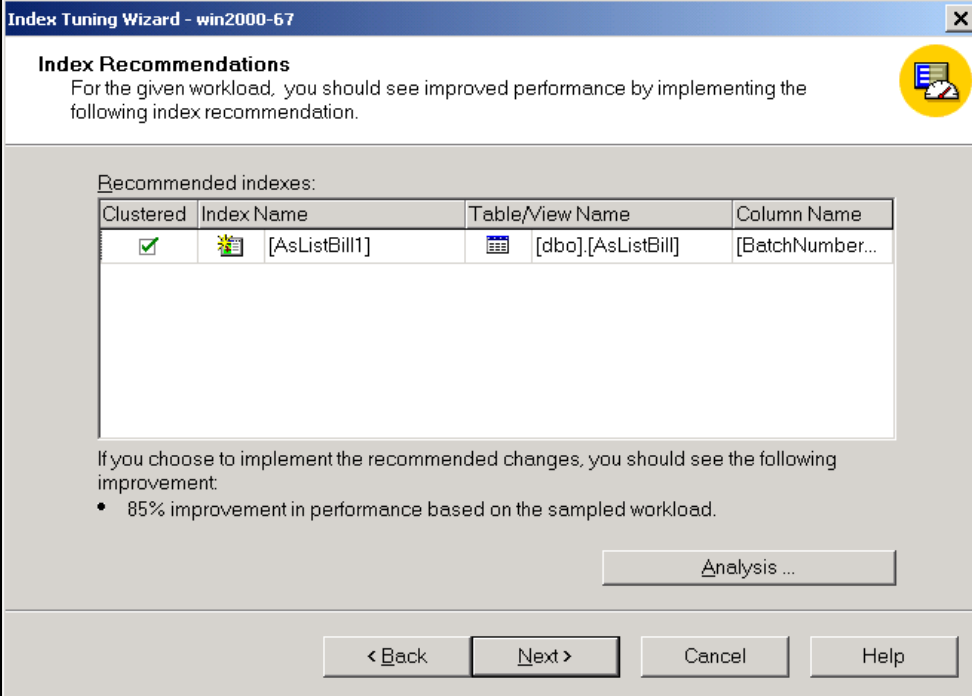
This is a list of all the tables the wizard can analyze. This analysis is often time consuming. To reduce the duration, select only the most critical tables. By changing the number of projected table rows you can account for future database growth.

Table	Actual Rows	Projected Rows
<input type="checkbox"/> [dbo].[AsInquiryScreen]	1	1
<input type="checkbox"/> [dbo].[AsLinkRequirement]	58	58
<input checked="" type="checkbox"/> [dbo].[AsListBill]	412	412
<input type="checkbox"/> [dbo].[AsLookupRequirement]	62	62
<input type="checkbox"/> [dbo].[AsMapName]	26	26
<input type="checkbox"/> [dbo].[AsNotAssessValue]	0	0

Select All Tables Clear All Tables



< Back Next > Cancel Help

11. When the tuning wizard finishes, look over the Analysis and select **Next**.



Index Recommendations
For the given workload, you should see improved performance by implementing the following index recommendation.

Recommended indexes:

Clustered	Index Name	Table/View Name	Column Name
<input checked="" type="checkbox"/>	 [AsListBill1]	 [dbo].[AsListBill]	[BatchNumber...]

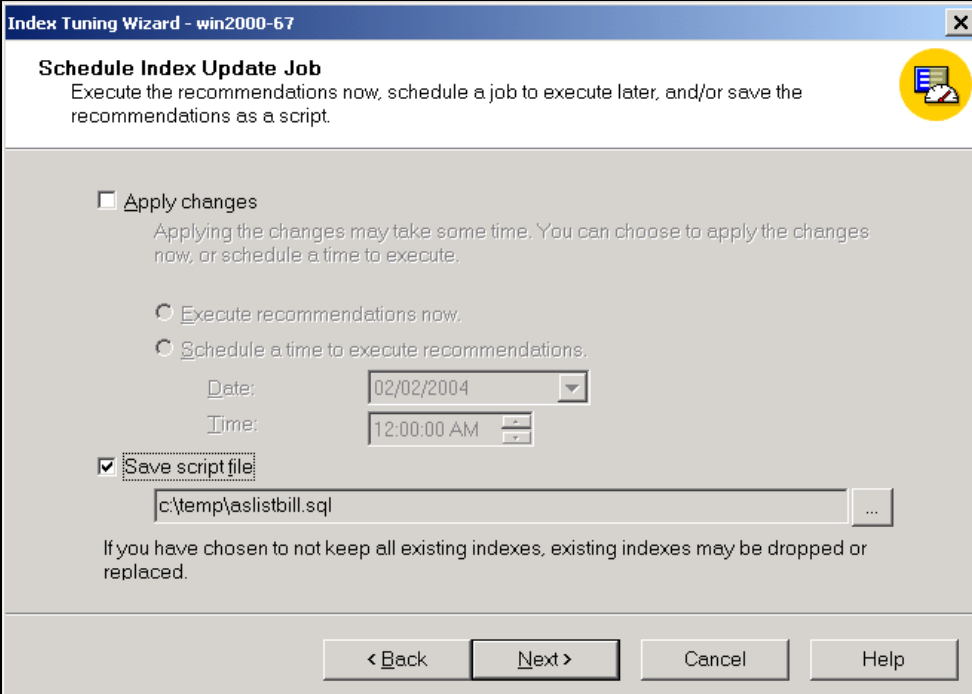
If you choose to implement the recommended changes, you should see the following improvement:

- 85% improvement in performance based on the sampled workload.

Analysis ...

< Back Next > Cancel Help

12. Save the results to a file and select **Next**.



Schedule Index Update Job
Execute the recommendations now, schedule a job to execute later, and/or save the recommendations as a script.

☐ **Apply changes**
Applying the changes may take some time. You can choose to apply the changes now, or schedule a time to execute.

☐ Execute recommendations now.

☐ Schedule a time to execute recommendations.

Date: 02/02/2004

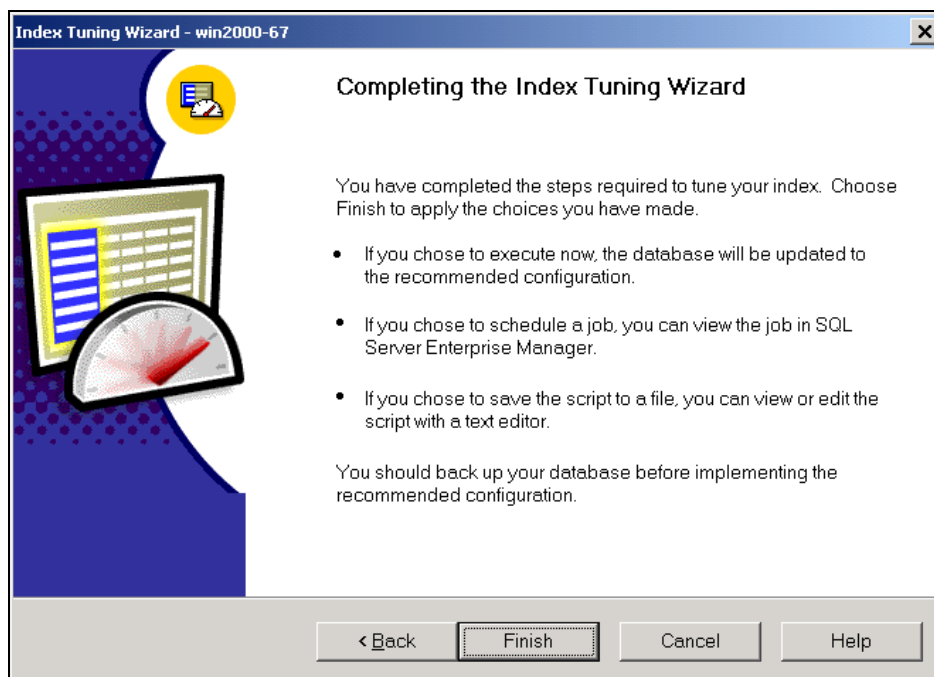
Time: 12:00:00 AM

☒ **Save script file**
c:\temp\aslistbill.sql

If you have chosen to not keep all existing indexes, existing indexes may be dropped or replaced.

< Back Next > Cancel Help

13. Select **Finish**.



14. Select **OK**.



The tuning wizard SQL script can now be reviewed. Run it and test the performance gains.

Source Code and Business Rule Source

Visual Source Safe is a source code management and version control system in which a team of developers can safely and easily share source code without the risk of overwriting someone else's changes. OIPA also uses it for version control of our web pages, source code, DLLs, and our executables.

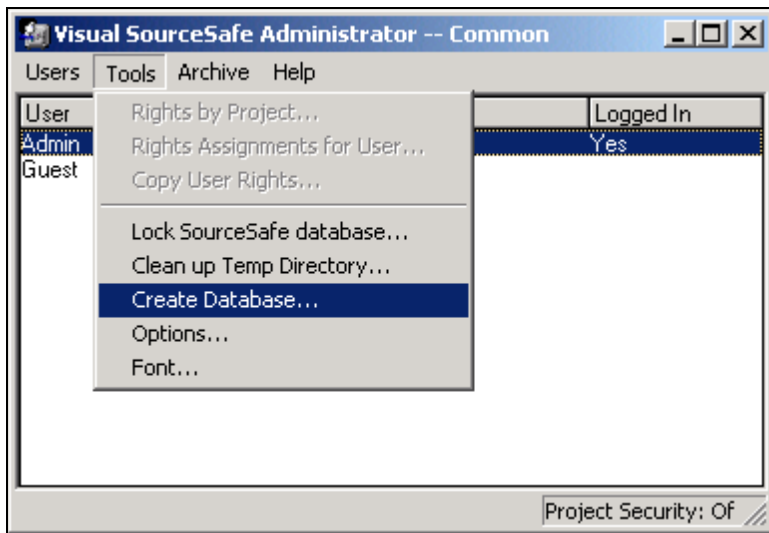
To place the OIPA application into VSS, one person will have to create a VSS DB on the network's domain or workgroup. This one VSS database should be shared by all the developers on a team.

Each of our programmers has a local copy (on their hard drive) of the OIPA system. These files are controlled by VSS.

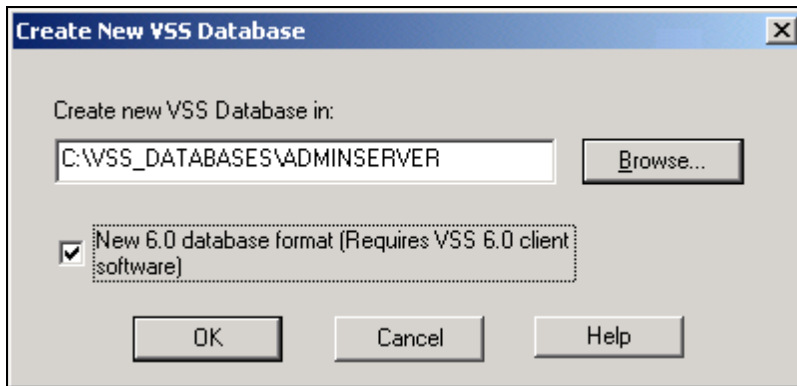
Creating a VSS database

Create VSS database

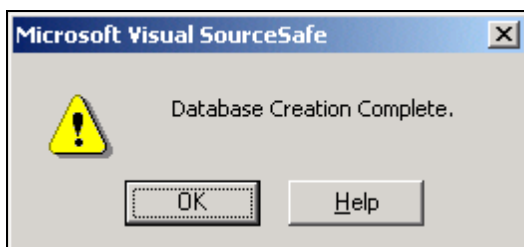
1. Start Visual SourceSafe 6.0 Administrator.
2. Select the Tools menu, then select **Create Database ...**



3. Choose a network location for the VSS database.
4. Select **OK**.



You should see the following screen.

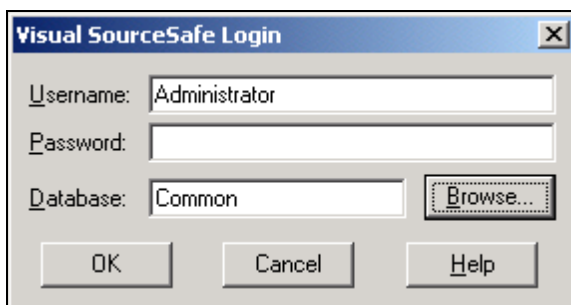


5. Click **OK**. A VSS database has been created.
6. At this time, enter the Users menu and add all applicable VSS users
7. Close Visual SourceSafe 6.0 Administrator.

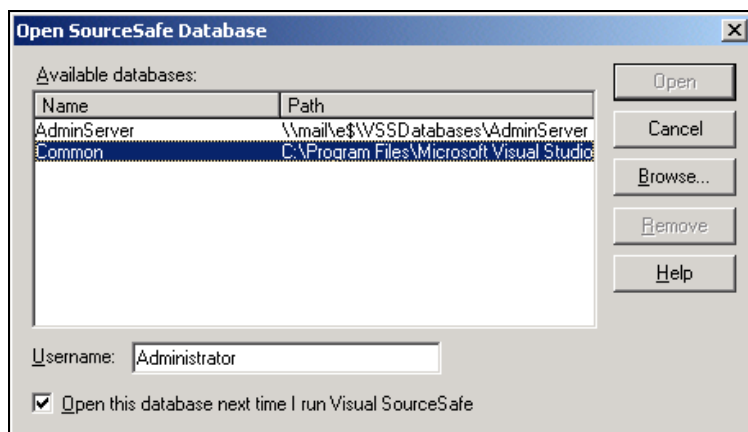
Changing VSS Database and Login

Change VSS Database and Login

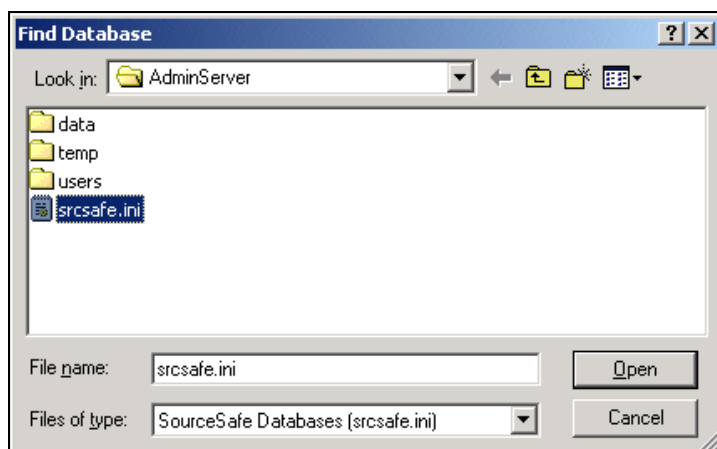
1. Start MS Visual SourceSafe 6.0.
2. Select **Browse....**



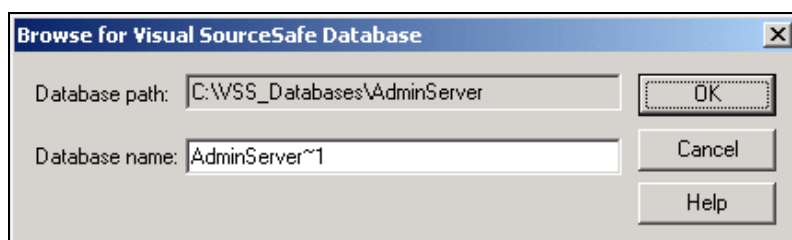
3. Select **Browse** and find the VSS database that was just created.



4. Select **Open**.

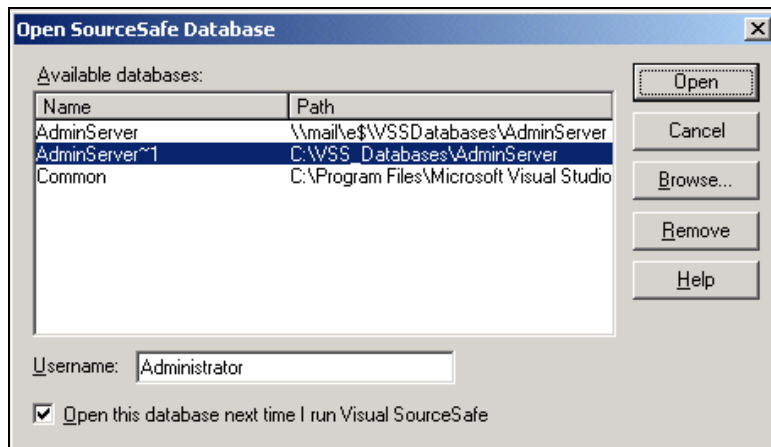


5. Select **OK**.

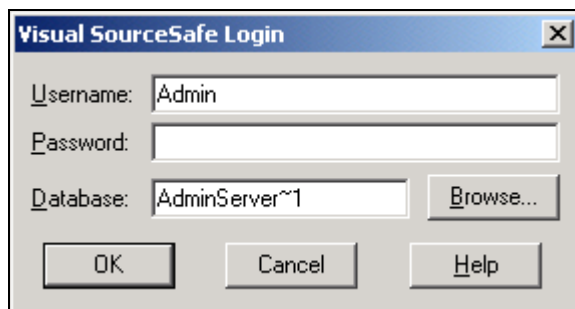


6. Select **Open**.

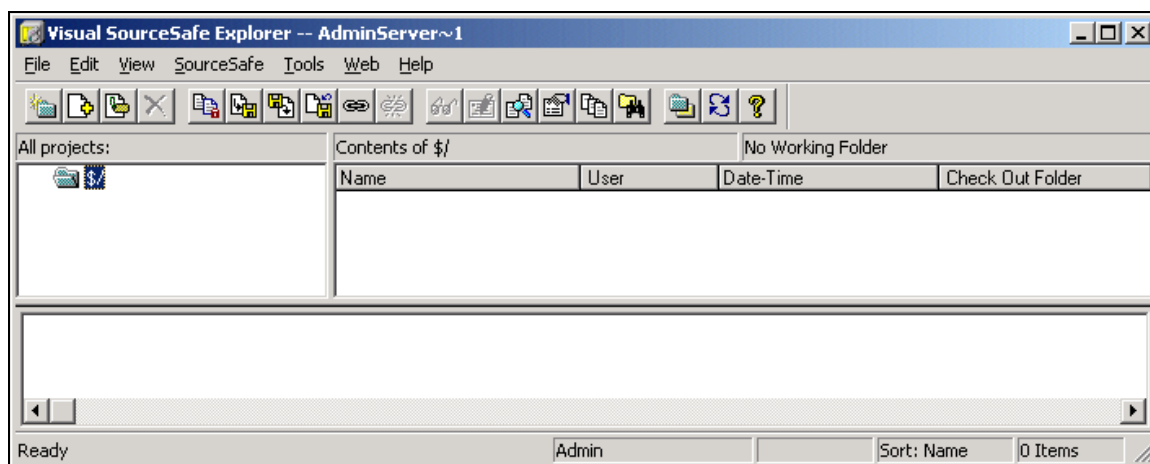
Note: Check box Open this database next time.



7. Enter the Password and select **OK**.



The following screen will open:



Local copy of the source code

A copy of the OIPA application should already be on the hard drive.

1.1.1.1.1 Getting Source in Eclipse

Put Source in Eclipse

1. Open Eclipse.
2. Go to the Window | Open Perspective | Other... menu item.
3. Choose **CVS Repository Exploring** and select **OK**.
4. Right click in the CVS Repositories tab view and choose New | Repository Location...
 1. Host: <your host name>
 2. Repository Path: /home/adminserver/cvsroot
 3. User: <your user name>
 4. Password: <your user account password>
 5. Connection Type: pserver
 6. Click Finish
5. Expand the tree for the new repository and locate:
 1. HEAD/adminserver.j2ee.ear
 2. HEAD/adminserver.j2ee.ejb
 3. HEAD/adminserver.j2ee.web
 4. HEAD/adminserver.j2ee.libs
6. Right click on each and choose Check Out

Creating Projects in VSS

Now a project must be created in VSS for every directory in the tree, starting with AdminServer.

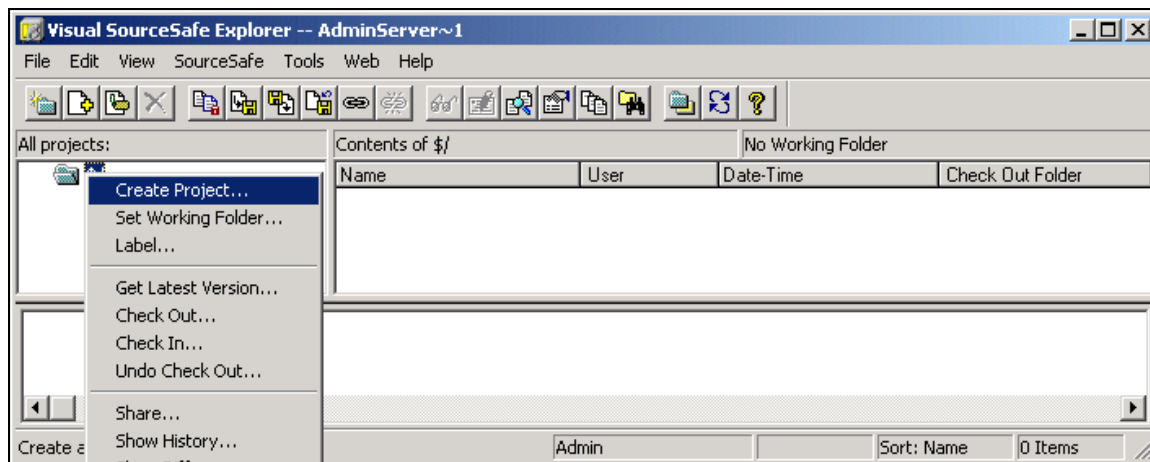
Examples of the first and second directories:

C:\Development\AdminServer => VSS project would be AdminServer

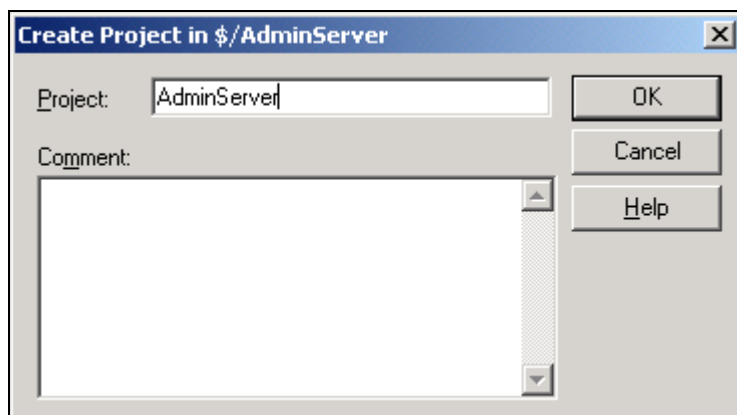
C:\Development\AdminServer\Bin => VSS project would be Bin created under AdminServer

Create Project in VSS

1. Right click the folder labeled \$/ and select **Create Project ...**



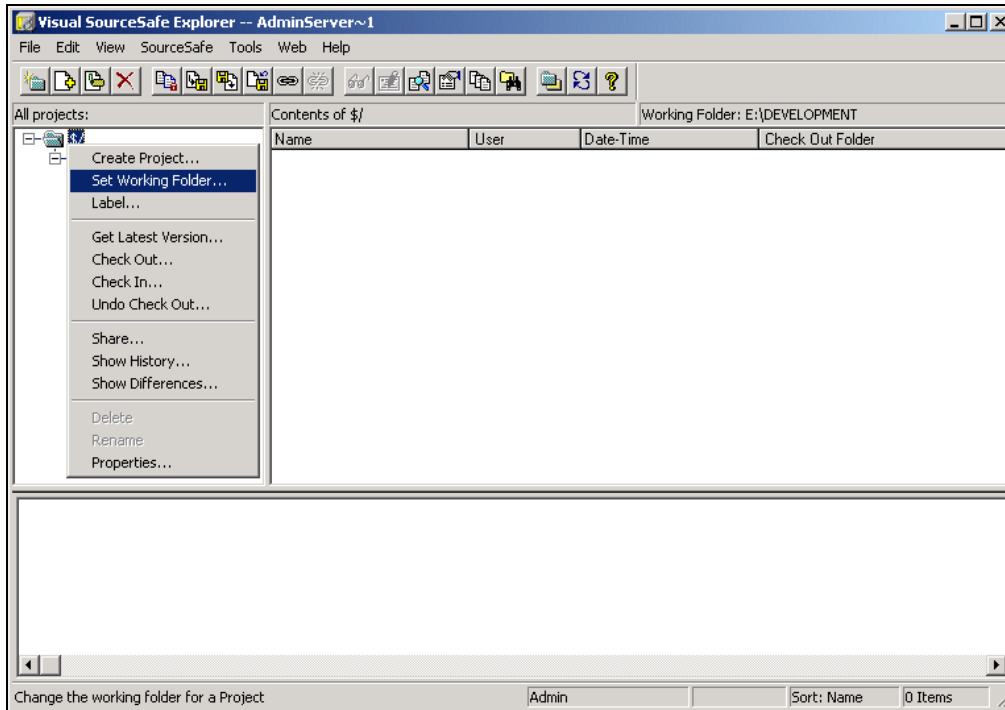
2. Give the Project the same name as the directory name.
3. Select **OK**.



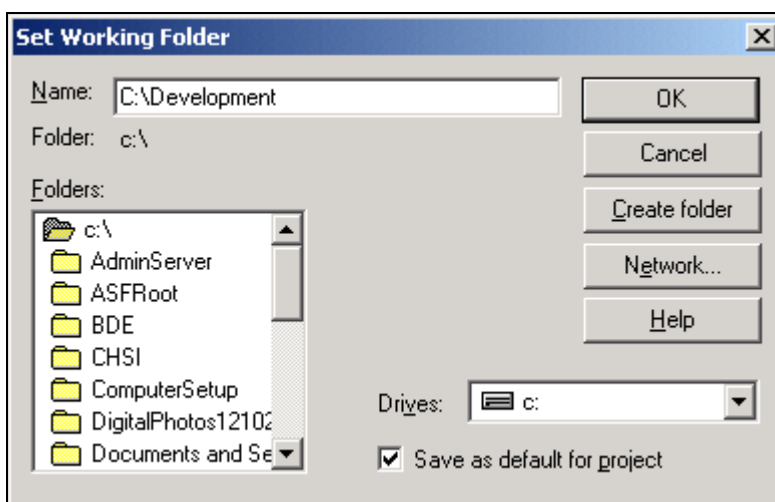
Set Working Folder

Set Working Folder

1. After creating all of the projects, right click the root project \$/ and select **Set Working Folder...**



2. Choose the directory with the source code.
3. Select **OK**.



GUID (Globally Unique Identifier)

All of OIPA's database tables (excluding the code tables) have unique keys using a GUID. Also, Windows stores all COM+ DLLs classes in the registry with a GUID called a Class ID. Using GUIDs, items can be uniquely identified (objects, rows in tables, and anything else) across all computers in the entire world at any given time.

The following is an example of a GUID used in the system:

GUID = {9435A58C-BB08-4C6E-9FD8-0985EFED2EC1}

The formula for this number is:

8 characters – 4 characters – 4 characters – 4 characters – 12 characters

GUIDs are commonly displayed by 32 character hexadecimal number formatting. This is the format used in Windows registry for all registered COM+ objects and also in SQL Server.

A GUID is unique by computer and by time. An Algorithm to generate a GUID uses 48 bits usually based on the physical address (or MAC) of its network card and another 60 bits that represent the amount of time from a fixed date and time (midnight on October 15, 1582). GUIDs are the same as **Universally Unique Identifiers** (UUIDs), which is the term used by the Open Software Foundation and the UNIX community.

- A GUID can also be expressed as a 128 bit integer = 4 (bits to store Hex) * 32 (size) or 16 bytes.
- GUIDs support approximately 3.4×10^{38} unique values.

GUIDs in SQL Server

The GUID has a data type called uniqueidentifier, which has a size of 16 bytes. NEWID() function should be used to generate a GUID. Uniqueidentifier data type is a string representing a GUID value. It can be specified in either a character string or binary format. OIPA uses only the character string format when working with GUIDs in SQL Server.

{6F9619FF-8B86-D011-B42D-00C04FC964FF} (Character string format)

XML: eXtended Markup Language

History of XML

In the late 1960s, data exchange between organizations became a necessity, and notations in data or about data describing the data became commonplace. Initially, data was transferred in the form of fixed length records. Soon, comma delimited methodology gained acceptance. However, as software and hardware systems became disparate and complicated, data exchange started to become unmanageable and there arose a need to standardize data interchange.

Though the first markup language, **Generalized Markup Language**, came into being around 1969, the wide use of markup languages became a reality in the late eighties with the invention and acceptance of **Standard Generalized Markup Language** (SGML).

The advent of the World Wide Web, having made HTML popular, allowed W3C to create a markup language as easy as HTML and as powerful as SGML. It was called **eXtended Markup Language** (XML).

Building blocks of XML

The following is a sample of an XML stream containing data about entities:

```
<Entities>
  <Entity TYPE="Corporation">
    <CompanyName>East India Co.</CompanyName>
    <TIN>19-0027510</TIN>
    <Addresses>
      <Address TYPE="HomeOffice" FORMAT="UK">
        <Street>5 Calvert Ave</Street>
        <City>London</City>
        <PostalID TYPE="UK">E2 7JP</PostalID>
      </Address>
      <Address TYPE="Colonial" FORMAT="US">
        <Line>The Robert Morris Building</Line>
        <Street>5 Chestnut Street</Street>
        <City>Philadelphia</City>
        <PostalID TYPE="US">19001</PostalID>
      </Address>
    </Addresses>
  </Entity>
  <Entity TYPE="Individual">
    <Name>
      <First>Samuel</First>
      <Last>Adams</Last>
```



```

        </Name>
        <SSN>100-72-7510</SSN>
        <Addresses>
            <Address TYPE="Home">
                <Street>745 Walnut Lane</Street>
                <City>Boston</City>
                <State CODE="MA">Massachusetts</State>
                <PostalID TYPE="Zip">2163</PostalID>
            </Address>
        </Addresses>
    </Entity>
    <Entity TYPE="Individual">
        <Name>
            <First>Benjamin</First>
            <Last>Franklin</Last>
        </Name>
        <SSN>133-72-0010</SSN>
        <Addresses>
            <Address TYPE="Home">
                <Street>456 Spruce Street</Street>
                <City>Philadelphia</City>
                <State CODE="PA">Pennsylvania</State>
                <PostalID TYPE="Zip">19100</PostalID>
            </Address>
        </Addresses>
    </Entity>
    .
    .
    .
</Entities>

```

An XML stream is built using tags and their data, along with attributes and their values.

Tag

A Tag is the primary delimiter of data in XML. Every tag has a starting manifestation and an ending manifestation. For example, <Name> is the start and </Name> is the end. Any amount of data or subsets of tags can exist inside the start and end tags.

Attributes

Attributes are used inside of a tag to provide indicators about the data and sub tags within the tag. There can be one or more attributes inside a tag. For example, address formats vary across the world. In order to facilitate that in data, an attribute called FORMAT is used. Any application using the XML data stream can be cognizant of the format of the address.

XPATH Query

When data is transferred between organizations, there may arise a need to search through and identify specific data. The creators of XML understood this need and provided a robust search methodology. XPATH, as it was called, is a powerful pattern-based search sub-language.

For instance, if King George III (under the Coercive or Intolerable Act) decided to punish the residents of Boston who were involved in dumping the tea that could have made 24,000,000 cups, he would need their social security numbers.

```
/Entities/Entity[@TYPE="Individual" and $any$ Addresses/Address/City="Boston"]/SSN
```

To understand this query, the process of building it step by step will help.

Step 1: XPATH for all entities:

```
/Entities/Entity
```

Step 2: XPATH for all entities who are individuals:

```
/Entities/Entity[@TYPE="Individual"]
```

Step 3: XPATH for all entities who are individuals and are associated with Boston:

```
/Entities/Entity[@TYPE="Individual" and $any$ Addresses/Address/City="Boston"]
```

Step 4: XPATH for just the SSN of the desired (or undesired) entities:

```
/Entities/Entity[@TYPE="Individual" and $any$ Addresses/Address/City="Boston"]/SSN
```

OIPA and XML

OIPA uses XML in several key areas. It uses XML as a conduit for data flow in the simplest form: to help define complex transaction processes, and to help define all business rules.

XML Standards

OIPA follows a strict standard in the structure of XML:

1. All tags are proper case (Hungarian notation).
2. All attributes are in upper case.
3. All sub tags are indented a tab space (3 spaces).
4. Tags have either sub tags or data – NOT both.
5. There is minimal, if any, use of abbreviations.

XML Payload and Scalability

The OIPA System views its persistent data in three different levels:

- Mapping Information Data
- Key and Business Critical Data
- Business Detail Data

Mapping Information Data

Mapping data refers to data that helps contain business rules used by rule engines; the backbone of the OIPA System. Mapping data is seldom used in updates or inserts and is usually read from the database to help the rule engines carry out their functionality. The mapped data elements are created by business analysts at the time of system set-up.

Key and Business Critical Data

A look at the data dictionary of the OIPA system shows that not all data in the system is stored in the XML elements. Data that represents critical business information such as policy numbers, agent numbers, last names, first names, and policy valuation information are not XML based. Though trivial, it is worth mentioning that key and index elements are not stored in XML either. If a data item is identified as something that will be required for a quick read, write, or something that might be involved in ad-hoc queries, it is stored as a relational element. Conversely, if it is not a 'critical' data item, it is stored as an XML element.

Business Detail Data

As the name suggests, business detail data is merely used for detail. It is generated in the system and maintained for helping users understand the underpinnings of the activities they request.

Most SQL products are beginning to understand the need for the persistence of XML data, and are providing data types and query functionalities for XML blobs.

An Example

A carrier may administer several different products. Policies associated with different products require different sets of data. While conventional systems modify the database to make way for new products. The OIPA system stores the policy detail in an XML format, thus making new product development simple in comparison. However, critical and common data such as policy number, effective date, and issue state are stored as relational elements. A rule also exists to help define these policy XML fields. This rule, along with other business rules, is stored in XML, allowing for a straightforward database design.

Developing a Sample J2EE Presentation Component and Business Logic Component

These instructions assume the developer has obtained the source code, set-up an IDE project and configured an application server. This example assumes an IDE of Eclipse and an application server of JBoss.

Presentation Component

Java Server Page

To create a Java Server Page (JSP), use your editor to create a new file. Save the file under the web project's jsp directory and name it Sample.jsp. For example, adminserver.j2ee.web\jsp\test\Sample.jsp.

Add the following code to the file:

```
<%@ page language="java"
import="java.util.LinkedList,com.adminserver.dal.*,com.adminserver.uip.*,com.adminserver.dcl.*"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/as-taglib.tld" prefix="as" %>

<html:html>
<head>
<title>AdminServer.com</title>
<link rel="stylesheet" href="../lib/style/AsCommon.css">
<script language="JavaScript"><jsp:include page="/lib/javascript/AsCommon.jsp"/></script>
</head>
<body>
<script LANGUAGE="JavaScript">
document.body.style.cssText = 'background-image:url(..images/<bean:write
name="companyCosmetics" property="tile"/>');
</script>

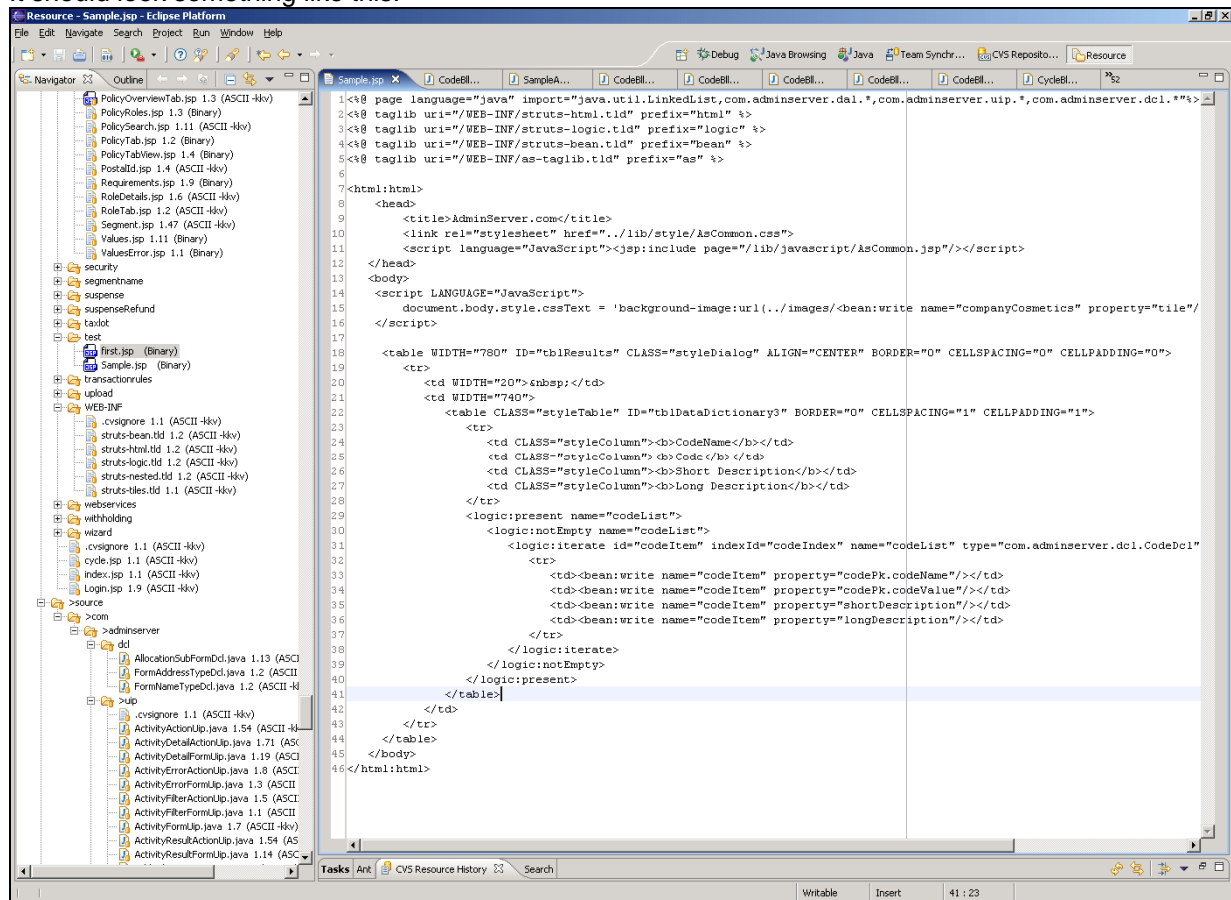
<table WIDTH="780" ID="tblResults" CLASS="styleDialog" ALIGN="CENTER" BORDER="0"
CELLSPACING="0" CELLPADDING="0">
<tr>
<td WIDTH="20">&nbsp;</td>
<td WIDTH="740">
<table CLASS="styleTable" ID="tblDataDictionary3" BORDER="0" CELLSPACING="1"
CELLPADDING="1">
<tr>
<td CLASS="styleColumn"><b>CodeName</b></td>
<td CLASS="styleColumn"><b>Code</b></td>
<td CLASS="styleColumn"><b>Short Description</b></td>
<td CLASS="styleColumn"><b>Long Description</b></td>
</tr>
<logic:present name="codeList">
<logic:notEmpty name="codeList">
<logic:iterate id="codeItem" indexId="codeIndex" name="codeList"
type="com.adminserver.dcl.CodeDcl">
```

```

<tr>
  <td><bean:write name="codeItem" property="codePk.codeName"/></td>
  <td><bean:write name="codeItem" property="codePk.codeValue"/></td>
  <td><bean:write name="codeItem" property="shortDescription"/></td>
  <td><bean:write name="codeItem" property="longDescription"/></td>
</tr>
</logic:iterate>
</logic:notEmpty>
</logic:present>
</table>
</td>
</tr>
</table>
</body>
</html:html>

```

It should look something like this:



Struts Action

OIPA uses the Struts framework to direct the presentation layer logic. To access the view implemented by Sample.jsp, you must create a Struts Action class. Locate the package hierarchy under adminserver.j2ee.web\source. Create a new class named SampleActionUip.java in the package com.adminserver.uip.

Use the following class definition:

```
public class SampleActionUip extends BaseActionUip
```

This Action needs some configuration information for the Struts ActionServlet to know how to deal with it. This is provided with XDoclet comments placed before the class definition as follows:

```
/**
 * @struts.action
 *     path = "/Sample"
 *     scope = "request"
 *
 * @struts.action-forward
 *     name = "display"
 *     path = "/test/Sample.jsp"
 */
```

These instructions are inserted into the struts-configuration file when the web application is built.

Next add the following import directives:

```
import java.util.Collection;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;

import com.adminserver.bll.CodeBllRemote;
import com.adminserver.bll.CodeBllUtil;
import com.adminserver.dcl.CompanyDcl;
import com.adminserver.dcl.UserDcl;
import com.adminserver.utl.ASEExceptionUtil;
```

Add the following method to the class:

```
public ActionForward execute( ActionMapping mapping, ActionForm form, HttpServletRequest request,
    HttpServletResponse response ) throws Exception {
    UserDcl userDcl = ( UserDcl )request.getSession().getAttribute( "userDcl" );
    CompanyDcl userCompanyDcl = ( CompanyDcl )request.getSession().getAttribute( "userCompany"
);
```

```

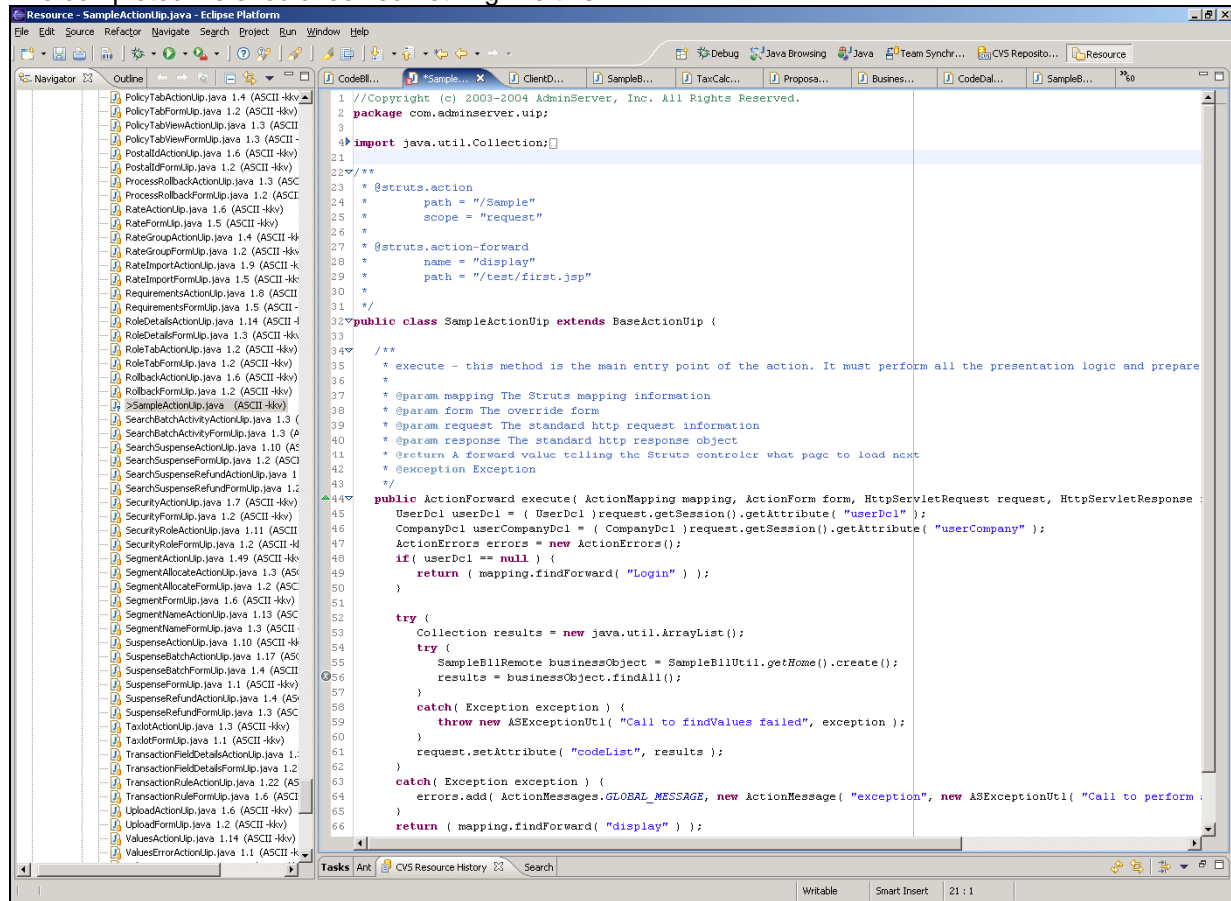
ActionErrors errors = new ActionErrors();
if( userDcl == null ) {
    return ( mapping.findForward( "Login" ) );
}

try {
    Collection results = new java.util.ArrayList();
    try {
        SampleBIIRemote businessObject = SampleBIIUtil.getHome().create();
        results = businessObject.findAll();
    }
    catch( Exception exception ) {
        throw new ASEExceptionUtil( "Call to findAll failed", exception );
    }
    request.setAttribute( "codeList", results );
}
catch( Exception exception ) {
    errors.add( ActionMessages.GLOBAL_MESSAGE, new ActionMessage( "exception", new
ASEExceptionUtil( "Call to perform search on Code beans failed", exception ) ) );
}
return ( mapping.findForward( "display" ) );
}

```

Some of these classes are in the business logic layer that we have yet to create. Until then the IDE will complain that these classes cannot be found. Ignore those messages for now.

The completed file should look something like this:



```

1 //Copyright (c) 2003-2004 AdminServer, Inc. All Rights Reserved.
2 package com.adminserver.uip;
3
4 import java.util.Collection;
5
6 /**
7  * @struts.action
8  *   path = "/Sample"
9  *   scope = "request"
10  *
11  * @struts.action-forward
12  *   name = "display"
13  *   path = "/test/first.jsp"
14  */
15 public class SampleActionUp extends BaseActionUp {
16
17     /**
18     * execute - this method is the main entry point of the action. It must perform all the presentation logic and prepare
19     *
20     * @param mapping The Struts mapping information
21     * @param form The override form
22     * @param request The standard http request information
23     * @param response The standard http response object
24     * @return & forward value telling the Struts controller what page to load next
25     * @exception Exception
26     */
27     public ActionForward execute( ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response ) {
28         UserDel userDel = ( UserDel request.getSession().getAttribute( "userDel" ) );
29         CompanyDel userCompanyDel = ( CompanyDel request.getSession().getAttribute( "userCompany" ) );
30         ActionErrors errors = new ActionErrors();
31         if ( userDel == null ) {
32             return ( mapping.findForward( "Login" ) );
33         }
34
35         try {
36             Collection results = new java.util.ArrayList();
37             try {
38                 SampleBillRemote businessObject = SampleBillUtil.getHome().create();
39                 results = businessObject.findAll();
40             }
41             catch( Exception exception ) {
42                 throw new ASExceptionUtil( "Call to findValues failed", exception );
43             }
44             request.setAttribute( "codeList", results );
45         }
46         catch( Exception exception ) {
47             errors.add( ActionMessages.GLOBAL_MESSAGE, new ActionMessage( "exception", new ASExceptionUtil( "Call to perform :
48
49         return ( mapping.findForward( "display" ) );
50     }
51 }

```

Sample Business Logic Component

Session EJB

The first step for the business logic layer is to create a Session EJB to provide the business logic API for our sample. We will create a Session EJB to look up the values of the AsCode lookup table. Locate the package hierarchy under adminserver.j2ee.ejb\source. Create a new class named SampleBillBean.java in the package com.adminserver.bl.

Add the following code:

```
package com.adminserver.bll;

import java.util.Collection;

import javax.ejb.CreateException;
import javax.ejb.SessionBean;

import com.adminserver.dal.CodeDal;
import com.adminserver.utl.ASExceptionUtl;
import com.adminserver.utl.CodeConstants;

/**
 * Code related methods
 *
 * @ejb.bean name = "SampleBll"
 *   description = "Sample EJB"
 *   jndi-name = "adminserver/Sample"
 *   local-jndi-name = "java:comp/env/SampleBll"
 *   type = "Stateless"
 *   view-type = "both"
 *
 * @ejb.resource-ref res-ref-name="ADMINSERVERDS"
 *   res-type="javax.sql.DataSource"
 *   res-auth="Container"
 *   res-sharing-scope="Shareable"
 *   jndi-name="java:ADMINSERVERDS"
 *
 * @websphere.resource-ref res-ref-name="ADMINSERVERDS"
 *   isolation-level="TRANSACTION_READ_COMMITTED"
 *
 * @ejb.ejb-ref view-type = "local"
 *   ejb-name = "SystemInformationBll"
 *   ref-name = "SystemInformationBll"
 *
 * @ejb.util
 *   generate = "physical"
 */
public class SampleBllBean extends BaseSessionBean implements SessionBean, CodeConstants {

    /**
     * The Bean's Create function. This is similiar to a construct and should
     * be used to initialize the class.
     *
     * @exception CreateException
     * @see com.adminserver.bll.BaseSessionBean#ejbCreate()
     */
    public void ejbCreate() throws CreateException {
        super.ejbCreate();
    }

    /**
     * @exception AsExceptionUtl
     * @ejb.interface-method
     */
}
```

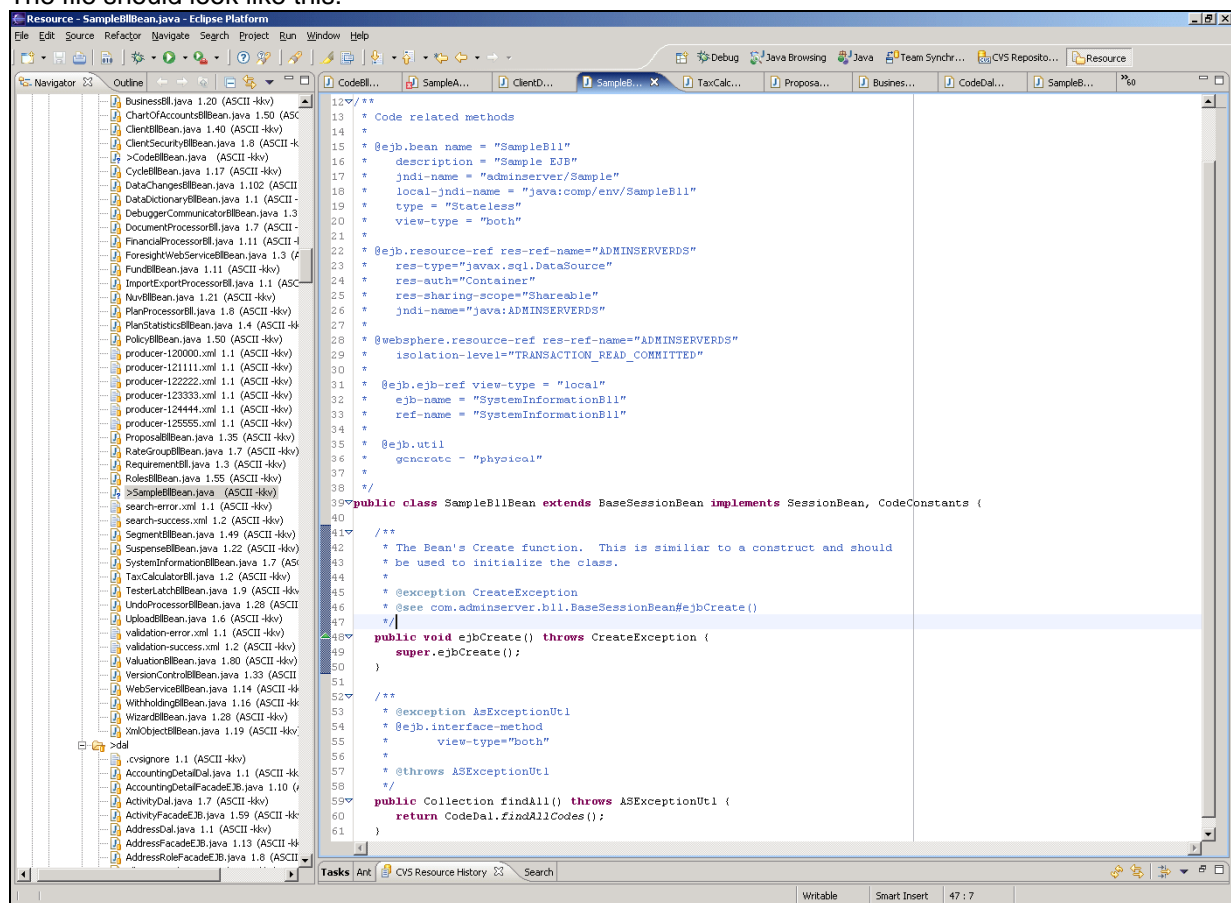
```

*    view-type="both"
*
* @throws AExceptionUtil
*/
public Collection findAll() throws AExceptionUtil {
    return CodeDal.findAllCodes();
}
}

```

The key elements are in the comments, which provide the information to generate the EJB infrastructure classes to allow the business layer to be accessed from the web client.

The file should look like this:



Data Access Layer

To retrieve data from the database, the EJB makes use of a class to abstract the mechanics of accessing the persistence layer. We create a data access layer object for the code table for this purpose. Create a new class named CodeDal.java in the package com.adminserver.dal.

We place in this file the code to access the database:

```
package com.adminserver.dal;

import java.util.Collection;

import org.hibernate.HibernateException;
import org.hibernate.Session;

import com.adminserver.utl.ASEExceptionUtl;
import com.adminserver.utl.HibernateSessionUtl;

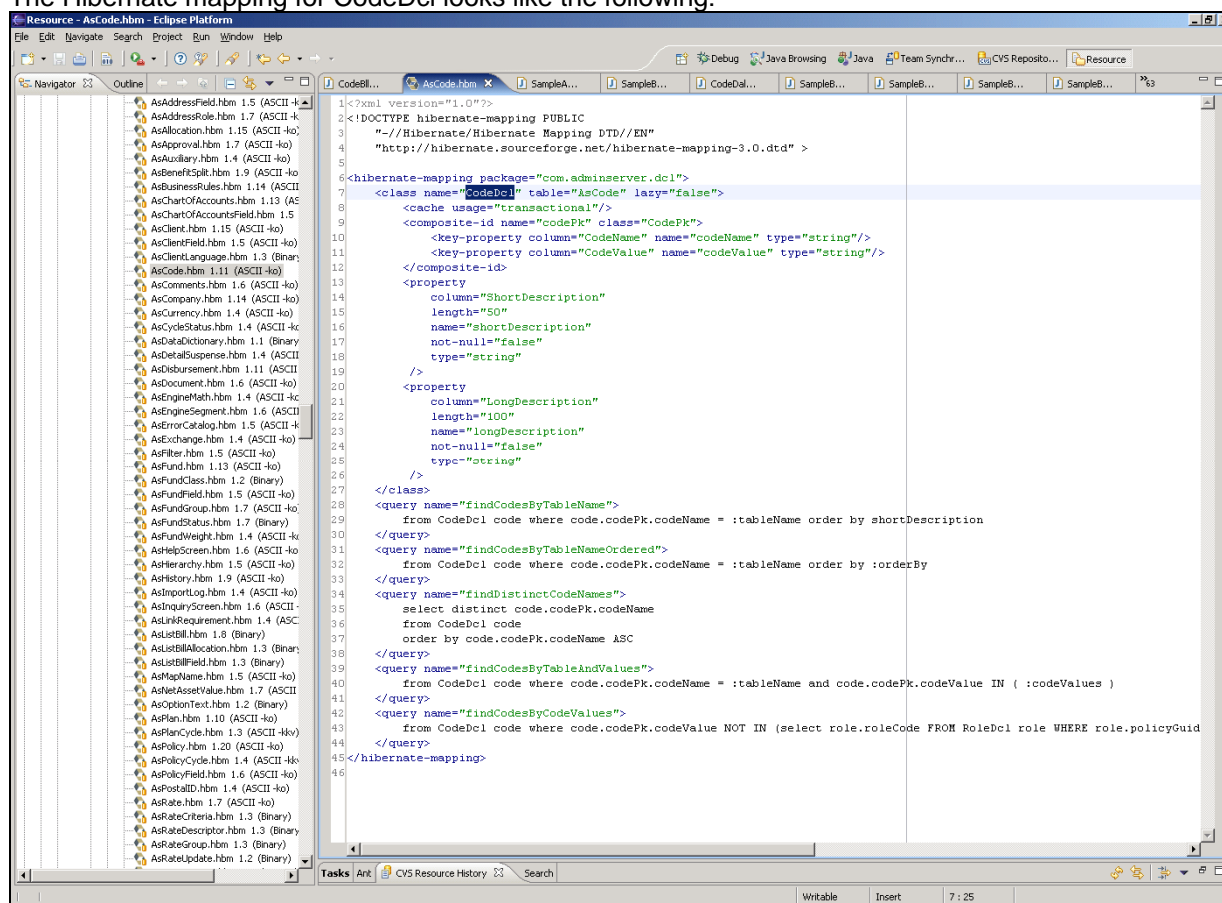
public class CodeDal {
    public static Collection findAllCodes() throws ASEExceptionUtl {
        Session session = null;
        try {
            session = HibernateSessionUtl.currentSession();
            return session.createQuery("from CodeDcl").list();
        }
        catch( HibernateException exception ) {
            throw new ASEExceptionUtl( "Call to findAllCodes failed", exception );
        }
        finally {
            HibernateSessionUtl.closeSession();
        }
    }
}
```

The data access layer can use any method to access the persistence layer. The standard method in OIPA is to use Hibernate.

Data Carrier and Hibernate Mapping

A normal java bean is used to collect the data from the database. In the case of this sample this class is com.adminserver.dcl.CodeDcl.java. The classes in this package are subclasses of auto-generated java beans created from Hibernate mappings.

The Hibernate mapping for CodeDcl looks like the following:



```

1<?xml version="1.0"?>
2<!DOCTYPE hibernate-mapping PUBLIC
3  "-//Hibernate/Hibernate Mapping DTD/EN"
4  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
5
6<hibernate-mapping package="com.adminserver.dcl">
7  <class name="CodeDcl" table="AsCode" lazy="false">
8    <cache usage="transactional"/>
9    <composite-id name="codePk" class="CodePk">
10      <key-property column="CodeName" name="codeName" type="string"/>
11      <key-property column="CodeValue" name="codeValue" type="string"/>
12    </composite-id>
13    <property
14      column="ShortDescription"
15      length="50"
16      name="shortDescription"
17      not-null="false"
18      type="string"
19    >
20    </property>
21    <property
22      column="LongDescription"
23      length="100"
24      name="longDescription"
25      not-null="false"
26      type="string"
27    >
28    </property>
29    <query name="findCodesByTableName">
30      from CodeDcl code where code.codePk.codeName = :tableName order by shortDescription
31    </query>
32    <query name="findCodesByTableNameOrdered">
33      from CodeDcl code where code.codePk.codeName = :tableName order by :orderBy
34    </query>
35    <query name="findDistinctCodeNames">
36      select distinct code.codePk.codeName
37      from CodeDcl code
38      order by code.codePk.codeName ASC
39    </query>
40    <query name="findCodesByTableNameAndValues">
41      from CodeDcl code where code.codePk.codeName = :tableName and code.codePk.codeValue IN ( :codeValues )
42    </query>
43    <query name="findCodesByCodeValues">
44      from CodeDcl code where code.codePk.codeValue NOT IN (select role.roleCode FROM RoleDcl role WHERE role.policyGuid
45    </query>
46  </hibernate-mapping>

```

Testing

Login to AdminServer. Enter the path mapped to the Action class. In this sample this was "/Sample" to create a URL like <http://localhost:8080/AsController/Sample>

The result should look like the following:

CodeName	Code	Short Description	Long Description
AsCodeDeathBenefitOption	01	Level - Option A	Level - Option A
AsCodeDeathBenefitOption	02	Increasing - Option B	Increasing - Option B
AsCodePhoneType	01	Home	Home
AsCodePhoneType	02	Cell	Cell
AsCodePhoneType	03	Business	Business
AsCodePhoneType	04	Fax	Fax
AsCodePhoneType	05	Other	Other
AsCodePhoneType	00	- Select phone type -	- Select phone type -
AsCodeStatus	57	Requirement Pending	Requirement Pending
AsCodeClientType	03	Producer-Individual	An Individual Producer/Representative/Agent
AsCodeClientType	04	Trust	Trust
AsCodeSegmentType	88	Test Policy Protector GDBR	Policy Protector GDBR Testing Tool - Please do not delete from Unit and Client
AsCodeClientType	06	Estate	Estate of Insured, Annuitant, or Owner
AsCodeRole	94	Endorsee	Endorsee
AsCodeClientType	07	UGMA/UTMA	Uniform Gift to Minors Act or Uniform Transfer to Minors Act
AsCodeRole	95	Second Contingent Owner	Second Contingent Owner
AsCodeRole	96	Partner	Partner
AsCodeRole	97	General Partner	General Partner
AsCodeClientType	08	Producer-Corporate	A Corporate Producer/Representative/Agent
AsCodeRole	98	Estate Of Owner	Estate Of Owner
AsCodeRole	61	Inactive-Producer	Inactive-Producer
AsCodeRole	62	Producer-Successor	Producer-Successor
AsCodeSegmentType	05	DeferredAnnuity	DeferredAnnuity
AsCodeActivityType	02	Reversal	Reversal
AsCodeActivityType	03	Undo	System generated reversal
AsCodeActivityType	04	Redo	System generated activity
AsCodeActivityType	05	Deleted	Deleted Activity
AsCodeAddressRole	01	EFT	Electronic Funds Transfer
AsCodeAddressRole	02	Home	Home
AsCodeAddressRole	03	Employer	Employer
AsCodeAddressRole	04	Other	Other
AsCodeAddressRole	05	Mailing	Mailing
AsCodeAddressRole	06	Phoenix Account	Phoenix Account
AsCodeAddressRole	07	Insurance Company	Insurance Company
AsCodeAdministrativeArea	01	New Business	New Business
AsCodeAdministrativeArea	02	Customer Service	Customer Service
AsCodeAge	01	Age Last	Age Last Birthday
AsCodeAge	02	Age Nearest	Age Nearest Birthday
AsCodeAge	03	Age Next	Age Next Birthday
AsCodeAllocationMethod	01	Percent	Percent
AsCodeAllocationMethod	02	Amount	Amount
AsCodeAllocationMethod	03	Units	Units
AsCodeAllocationMethod	04	Pro Rata	Pro Rata
AsCodeAllocationType	01	Plan Default	Plan Default
AsCodeAllocationType	02	Future Allocations	Future Allocations
AsCodeAllocationType	03	Activity	Activity Allocation
AsCodeAllocationType	04	Plan Default - Issue	Plan Default - Issue
AsCodeAllocationType	05	Segment Default	Segment Default
AsCodeAllocationType	06	Issue-Retail	Issue-Retail
AsCodeAllocationType	07	Issue-Insurance	Issue-Insurance
AsCodeAllocationType	08	Issue-Consults	Issue-Consults
AsCodeAllocationType	09	Rebalancing Program	Rebalancing Program
AsCodeAllocationType	10	AIF	AIF