Oracle9*i* Forms – New Features

An Oracle White Paper
April 2002

Oracle9*i* Forms – New Features

Oracle9*i* Forms – New Features

INTRODUCTION

Oracle Forms, as a product, has been around, with a variety of name changes, since 1985. The product continues to grow and evolve, and we are still going strong with the Oracle9*i* Forms Release! Over the last 17 years, Forms has carried its customer base forward enabling the adoption of the latest and greatest technologies, from dumb terminals through client/server and to the Web, while preserving investment in applications and skills. This is an achievement that few other tools can boast and one that Oracle is proud to continue.

This article will present the new features of the Oracle9*i* Forms release, which are focused into three major categories to help support our customer base moving forward:

- Supporting global deployment
- Improving development and runtime productivity
- Integration and Oracle9*i* Application Server (Oracle9*i*AS) platform support

As well as looking at what is being added to the product in the new release, it is worth considering what has been taken away. The end of the article will discuss some of the changes that might affect existing applications. However, the one major change that is worth mentioning right now is the fact that Oracle9*i* Forms Release only supports Web deployment. Client/server and character mode users will have to remain on Oracle Forms 6*i*, unless they wish to migrate to Web deployment as well.

SUPPORTING GLOBAL DEPLOYMENT

Web deployment of Forms applications gives customers the ability to install and maintain their Forms applications remotely from their actual application users. The Forms could be installed and running in a different building, city, or even country. The possibility of an application being run by users who are accessing the network through a variety of channels (LAN, WAN, internet) and from geographically spread locations raises some interesting issues that Oracle has tried to address in the new release.

Introducing the Forms Listener Servlet

Not such a new feature? You would be right in thinking you've seen this already. The Forms Listener Servlet feature was developed in Oracle9*i* Forms, but Oracle has back-ported it into later patches of Forms 6*i* (Patch 4 and above) simply because it is such a useful feature.

The Forms Listener Servlet runs within Oracle9*i* Application Server and manages

- The creation of a Forms Server runtime process for each client
- Network communications between the client and its associated Forms Server runtime process

All of the traffic between the browser client and the Forms Server goes via the Web server and the Forms Listener Servlet. Because the Web server acts as the network endpoint for the client, the other server machines and ports are no longer exposed at the firewall, as shown in Figure 1.

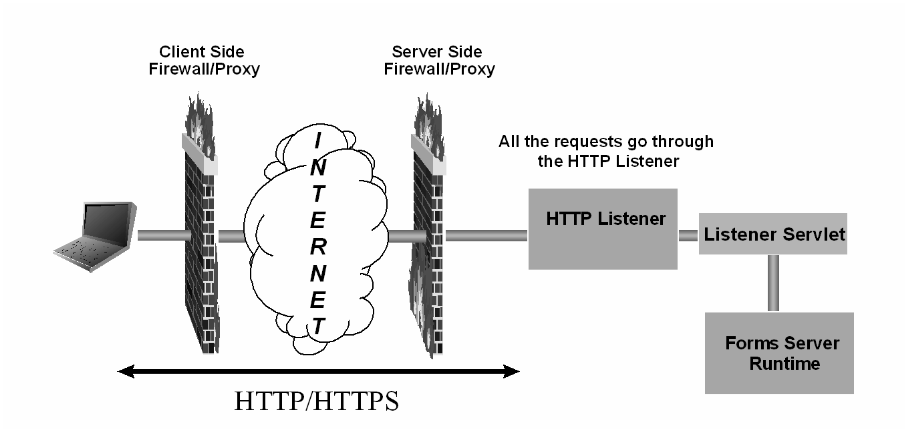

Client Side Firewall/Proxy
Server Side Firewall/Proxy
INTERNET
All the requests go through the HTTP Listener
HTTP Listener
Listener Servlet
Forms Server Runtime
HTTP/HTTPS

Figure 1. Forms Listener Servlet Architecture

The Oracle9*i* Forms Listener Servlet is designed to allow a more robust and standard deployment of Forms applications on the internet. When compared to the conventional Forms Listener, which can be used in Forms 6*i*, Listener Servlet provides the following benefits:

- Broader range of firewalls and proxies supported:
- Because the client browser always communicates with the Web server using HTTP or HTTPS (there is no direct connection between the client and the Forms Server runtime process), this architecture supports any firewall or proxy that can work with a standard servlet using servlet sessions.
- No protocol restriction (HTTP/1.1 or HTTP/1.0):
- Although the use of HTTP/1.1-compliant proxies provides better performance, this architecture also works well with HTTP/1.0-compliant proxies, too.

- No extra process to manage:
- Because this architecture eliminates the need for the Forms Listener process, the administrative tasks to start and stop the Forms Listener process are also no longer required.
- No nonstandard ports being used:
- As all traffic is channeled through the standard Web server HTTP or HTTPS ports with no separate ports required, sites with strict security and firewall configuration policies can successfully deploy Forms Server applications within the constraints that they have set for standard Web server access.
- No specific certificate to purchase/manage for SSL deployment:
- In the case of deployment using SSL (secure sockets layer), the HTTPS connection occurs between the client browser and Web server. Therefore, there are no specific security configuration requirements at the Forms Server level.
- Standard load balancing support:
- This architecture allows use of standard load balancing techniques, such as hardware-based load balancing, reverse proxy, and standard Apache JServ load balancing.
- Multiple runtime environments within a single server:
- The Forms Listener Servlet configuration allows you to set up multiple aliases for the Listener Servlet, each with its own environment settings defined. This allows different versions or languages of the same application to be run on the same server in a neatly partitioned way.
- Internet Explorer 5.x with native Java Virtual Machine (JVM) support:
- In addition to working with Oracle JInitiator, this architecture supports the use of Internet Explorer 5.x with native Microsoft JVM for internet deployment using HTTP and HTTPS connection modes.

In the new release, the Forms Listener Servlet is the only supported way of deploying Forms applications, providing a robust mechanism for deploying applications over the internet and across multiple network topologies.

Full documentation on the Forms Listener Servlet can be found on http://otn.oracle.com/products/forms.

Single Sign-on Support

Near the top of Oracle's list for enhancements in Oracle9*i* Forms was the ability for Forms to participate in a "single sign-on" environment. With the explosion of Web-based self-service and back office applications within companies, life without single sign-on is hard. Users have to remember multiple passwords for different applications and may have to enter them several times a day. This leads to security risks if all these passwords are written down and to support costs if they are not, as forgotten passwords have to be reset.

Single sign-on simplifies things by requiring the user to remember only one identity and one password. This only needs to be supplied when they sign on for the day. Thereafter, all applications for which they are registered are available to them using those credentials, which they have already established.

Oracle9*i* Forms, like many of the other tools within the Oracle9*i*AS Release 2 product suite, uses the Oracle Internet Directory (OID) as an LDAP server and the Oracle Login Server to handle the single sign-on through Oracle9*i*AS.

To enable single sign-on support for a Forms application, the application simply has to be registered with the Login Server. No code changes have to be made inside the Forms application, and the Forms application can still be used with an explicit logon and password if required.

Due to popular demand, single sign-on support will also be back-ported into the latest patches of Forms 6*i* in a limited form.

Improved Translation Facilities

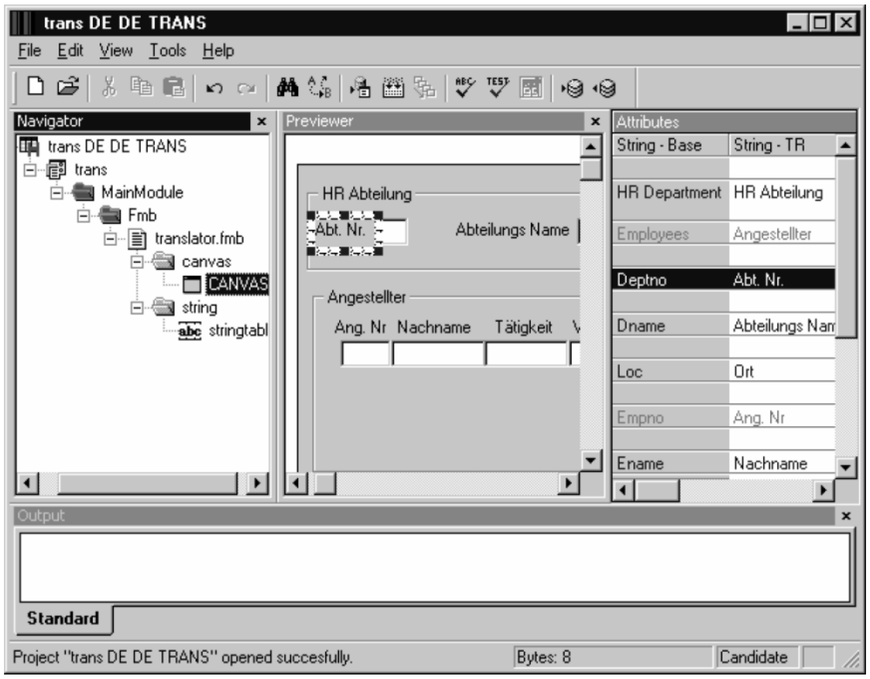
An important part of the global deployment of applications is the ability to have those applications running in multiple locales and languages. You may require this in a product that you market to multiple countries or even because legislation demands multi language applications.

With Oracle9*i* Forms Oracle is introducing a new product, TranslationHub, for translating Forms and Reports applications. TranslationHub replaces Oracle Translation Builder and Oracle Translation Manager, which have been supplied previously for this task.

The origins of TranslationHub lie in a tool that has been used internally by Oracle's own translation group for several years and is used for translating Oracle products, such as the RDBMS and Oracle E-Business Suite. Almost all user interfaces of Oracle products have been translated into up to 24 languages using this tool. TranslationHub is the first commercial release of the product, but given that the tool has been used internally by professional translators for several years, it can be seen as proven technology.

Some of the key features of TranslationHub are as follows:

- Provides translation in context with screen preview mode and context information for the translation
- Supports batch processing and translation
- Includes built-in team support
- Is built by translators for translators



String - Base	String - TR
HR Department	HR Abteilung
Employees	Angestellter
Deptno	Abt. Nr.
Dname	Abteilungs Nam
Loc	Ort
Empno	Ang. Nr
Ename	Nachname

Project "trans DE DE TRANS" opened succesfully. Bytes: 8 Candidate

Figure 2. TranslationHub

TranslationHub, as shown in Figure 2, provides the translator with multiple views of the module being translated, a WYSIWYG view to help provide context for the translation, hierarchical views based on the module structure, and terminology views where similar strings are grouped together.

Browser Language Detection

For a simple application translated into multiple languages, wouldn't it be good if all your end users could use the same URL to run the application, but automatically ended up seeing the translation in their language? Oracle9*i* Forms provides this feature as well! When a URL containing a config= parameter is submitted, the Forms Servlet will first look for the name of that parameter suffixed with the language code of the language that your browser is currently using. For instance, if a user in France issues the URL:

http://myserver/forms/forms.l90?config=sales

and the FORMSWEB.CFG file as shown in Listing 1,

```
[sales]
;Settings for the default application
splashScreen=/images/us/sales.gif
envFile = default.env
… etc.

[sales.de]
;Settings for the German version
splashScreen=/images/de/sales_de.gif
envFile = german.env
… etc.

[sales.fr]
;Settings for the French version
splashScreen=/images/fr/sales_fr.gif
envFile = french.env
```

Listing 1. FORMSWEB.CFG File

Then the **[sales.fr]** section would be used to define the application. Of course, a user in Germany would automatically use the [sales.de] application. If there were no match with the appended country code, then the application name on its own [sales] would be used.

Support for Character Semantics

Character semantics are a new way that database fields and PL/SQL variables can be defined. In versions of the Oracle Database prior to 9*i* and in existing versions of Forms, declaring a variable or field as CHAR(1) or VARCHAR(1) did not actually allocate space for one character, it allocated space for one BYTE. In standard Western European character sets, each character is one byte long, so it amounts to the same thing, but in multibyte and Unicode character sets, one character may well require more than one byte to store. For example, see Figure 3.

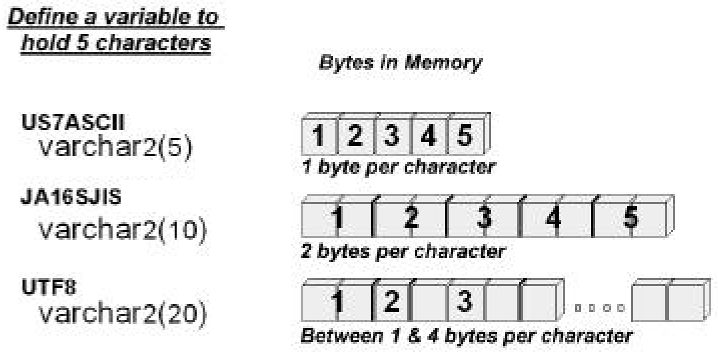


Figure 3. Comparison of storage requirements in various character sets

This is not really a problem for developers working in a fixed multibyte environment, as they simply increase the sizes of all fields and variables by a factor of two or three. Where it becomes a problem is in Unicode environments or in applications that need to be deployed to a mix of environments, some of which are single byte and some multibyte or Unicode.

Think about the kind of problem you might encounter. You have a business rule that specifies that a product ID must be no longer than 5 characters. If you are working in a single byte character set, then you make the field VARCHAR2(5) and the Forms item likewise. The user can only enter 5 characters and the business rule is fulfilled. Similarly, if you are in a fixed width multibyte character set, you might make the field VARCHAR2(10) (assuming 2 bytes per character 5 x 2 bytes=10 bytes) and the Forms item the same. Again, the user will be constrained to adding no more than 5 actual characters even though 10 bytes of storage are used to hold those 5 characters.

However, in the example of using a Unicode character set or an application designed for both single and multibyte, it would be necessary to design for the worst case scenario and make the database field VARCHAR2(20) (5 x 4 bytes) for the Unicode example and VARCHAR2(10) for the single/multibyte case. In this case, it is possible (if you do not code extra checks) for the user to enter more single byte characters than the business rule allows for, because the item declaration handles the worst case scenario only.

Character semantics solves this problem altogether by introducing a new syntax for the declaration of columns in the database and variables in PL/SQL

```
Column_name VARCHAR2(5 CHAR)
```

or

```
Column_name VARCHAR2(5 BYTE) / Column_name VARCHAR2(5)
```

The BYTE variant keeps the 8*i* behavior of one character really being one byte. The CHAR version, however, specifies that the column or variable should hold up to 5 characters in the current character set regardless of the number of bytes required. Thus the business rule will be observed without additional coding.

This new declaration syntax can be used for both database columns and PL/SQL CHAR and VARCHAR2 declarations. (Both in the database and in Forms PL/SQL). Forms items also support an additional property, **Data Length Semantics**, which can switch the field to use BYTE or CHAR semantics for applying properties such as Max Length and Query Length.

The default for Data Length Semantics is BYTE, to maintain compatibility with existing applications that are upgraded to Oracle9*i* Forms. However, it will be possible to define the default Length Semantics to use at compile time using the Environment variable **NLS_LENGTH_SEMANTICS**.

Timezone Support

Another problem experienced by applications that are widely deployed geographically is the simple concept of time. When users record a time in a Forms field, what exactly do they mean? The time where they are? The time where the Forms Server is running? Or the time on the Database Server?

This is not an important consideration when all three are in the same building, but imagine the scenario where a call center employee based in England is on the phone to a customer reporting the time of a credit card theft while on holiday in Hawaii, with the data stored in a database in New York.

It makes sense to record the time of the theft in the local time in Hawaii; but, if that then needs to be matched up with a fraudulent purchase made over the internet to a Web site in Singapore, it is clear why a universal concept of time is important.

Oracle9*i* Forms supports the ability to define the time zones of both the Forms Server and the Database Server so that times are correctly stored in the database.

Thus, if a time of 5:00p.m. is recorded from a Forms client running in British Summer Time (BST) into a database running in GMT (or Universal Coordinated [UTC] as it is more correctly called), then the time will be recorded as 4:00p.m. GMT. If the record is queried back, it is still seen in the local time zone, e.g., 5:00p.m. BST.

Likewise, if a colleague in San Francisco queries the same record with the local time zone set to Pacific Daylight Time (PDT), they will see the time as the correct time to them— 9:00a.m. PDT. (4:00p.m. GMT == 5:00p.m. BST == 9:00a.m. PDT).

Time zone support is something that many customers have already implemented using a variety of schemes, usually based on simple time offsets, e.g., PDT == GMT - 7.

However, time zones turn out to be much more complicated than they first appear. The following issues must be considered:

- Daylight Savings
- When does daylight savings start and end (in some countries it differs from year to year)?
- What about the ambiguous time once a year when we switch back from summertime/daylight savings, and we get one hour twice, once in summertime and once in standard?
- What about countries that have changed time zones in the past?

Oracle9*i* Forms uses a comprehensive knowledge base of timezone information, both current and historical, to handle any conversion.

The automatic conversion of time zones only applies to items defined as DATETIME.

Time zone support is configured by the use of environment variables on the Forms Services machine. These define the name of the timezone data file, the timezone that the database server is in and optionally the timezone that the browser client is using. The Forms client can also define the local time zone either as an explicit URL parameter, or by simply using the current timezone in which the browser is running.

The client timezone can also be changed during runtime with a call to Set_Application_Property().

In addition to the ability to configure the Forms session to run in a particular timezone, a new built-in called **ADJUST_TZ**() is also provided, which can be used to manually convert times between specified timezones using the same rules.

IMPROVING DEVELOPMENT AND RUNTIME PRODUCTIVITY

We've seen how Oracle9*i* Forms is instituting a bundle of improvements to make global deployment simpler, but let's not forget the day-to-day realities of using the tool. How have they changed and improved with Oracle9*i* Forms?

Run Directly in Browser

Previous versions of the Form Builder environment have supported a Web preview mode as well as a client/ server runtime execution. The preview mode was based on the Java Appletviewer, and though it gave a good idea of how the form would look when Web-deployed, it was not able to show certain things like Pluggable Java Components (PJCs) and Java Beans. Within Oracle9*i* Forms, when the application is run from within the builder, it is invoked within your favorite browser. This gives a true reflection of how it will look once deployed, with components such as PJCs fully functional.

N Tier "Remote" Debugging

As there is no longer a client/server runtime for use from the Form Builder, some changes have been made to the Forms debugger as well. If fact, Oracle has taken the opportunity to give it a total rewrite.

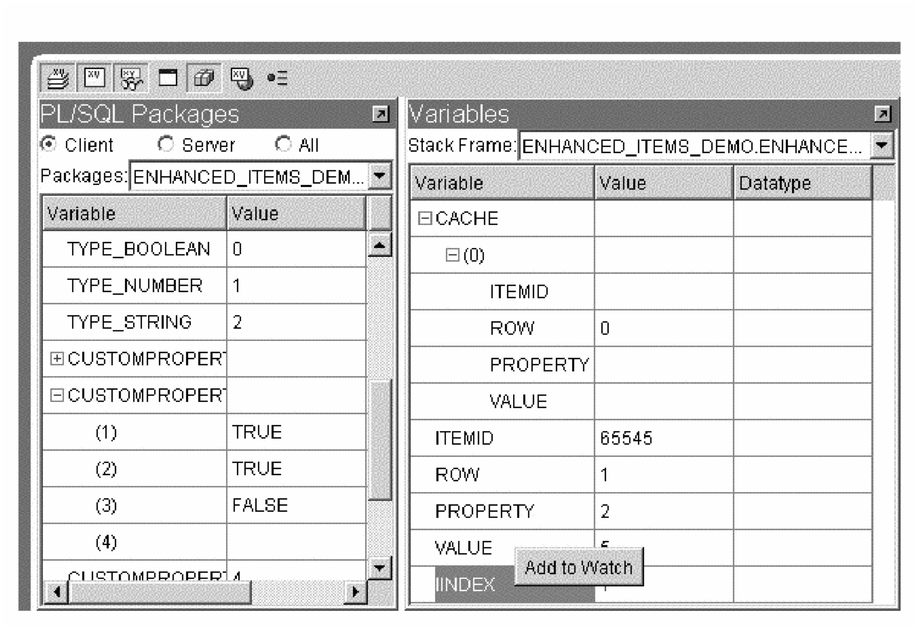
Debugging is now a builder-based, rather than a runtime-based, activity. You set breakpoints and step through the code while inside the builder, editing your module. Any changes that need to be made to the code can be made there and then!

Rather than having to run a special runtime executable (e.g., ifdbg60.exe) to get debugging capabilities, the builder has the debugging functionality and attaches to the normal runtime dynamically.

You can either run a form directly in Debug Mode from the builder, or more impressively, attach dynamically to a **remote runtime session** over your intranet. Imagine being able to work cooperatively with an end user, observing the code and stack in their running application as they perform an operation that causes a problem!

The builder provides a debug console from which a variety of subwindows can be undocked or docked at will. These windows display information such as:

- The execution stack
- Forms variables and globals
- Forms module values
- Watched variables
- PL/SQL package state


PL/SQL Packages

(● Client) ○ Server ○ All

Packages: ENHANCED_ITEMS_DEM...

Variable	Value
TYPE_BOOLEAN	0
TYPE_NUMBER	1
TYPE_STRING	2
⊞CUSTOMPROPERT	
⊟CUSTOMPROPERT	
(1)	TRUE
(2)	TRUE
(3)	FALSE
(4)	
CUSTOMPROPERT	4

Variables

Stack Frame: ENHANCED_ITEMS_DEMO.ENHANCE...

Variable	Value	Datatype
⊟CACHE		
⊟(0)		
ITEMID		
ROW	0	
PROPERTY		
VALUE		
ITEMID	65545	
ROW	1	
PROPERTY	2	
VALUE	5	
IINDEX	1	

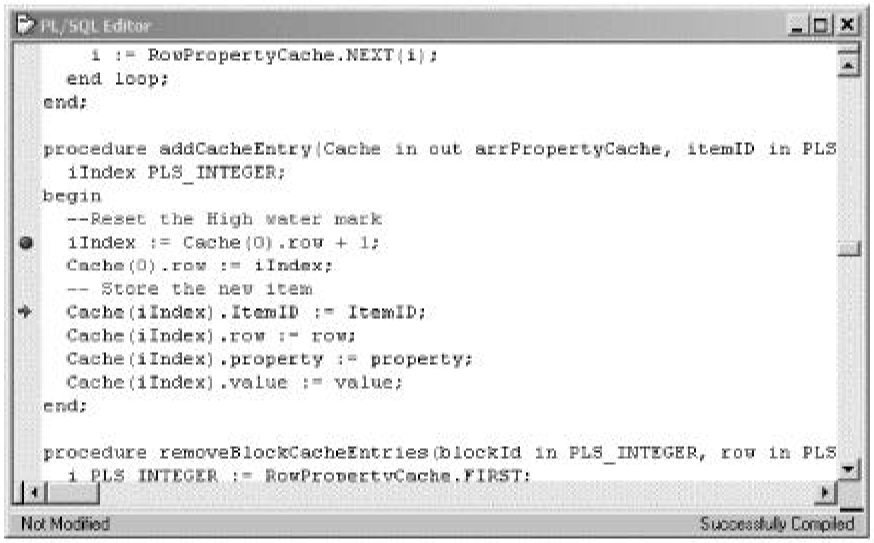
Figure 4. Two Docked Panels in the Debug Console Showing Package and Local Scope Variable Information

Notice from the screenshot in Figure 4 that not only are "simple" PL/SQL variables shown in the debugger, but for complex variables such as PL/SQL records, tables or even SQL Cursor handles, the object is represented as a tree into which you can drill down. So if you want to check what the values stored within the third record in a PL/SQL table of records currently are, you can drill down and take a look!

It is also possible to inspect variables that are declared in package headers and bodies and not in any specific function or procedure within that package. This is something that is not possible in the Forms 6*i* debugger.

Another key feature is the ability to set watch points on variables or expressions that are of particular interest. This can really be useful when bounds checking or trying to track down illusive exceptions. Such "watched" variables are shown in their own debug panel so you can keep an eye on those values that are really important to you.

Breakpoints and the current line execution are shown directly in the PL/SQL editor.



```
    i := RowPropertyCache.NEXT(i);
  end loop;
end;

procedure addCacheEntry(Cache in out arrPropertyCache, itemID in PLS
  iIndex PLS_INTEGER;
begin
  --Reset the High water mark
  iIndex := Cache(0).row + 1;
  Cache(0).row := iIndex;
  -- Store the new item
  Cache(iIndex).ItemID := ItemID;
  Cache(iIndex).row := row;
  Cache(iIndex).property := property;
  Cache(iIndex).value := value;
end;

procedure removeBlockCacheEntries(blockId in PLS_INTEGER, row in PLS
  i PLS_INTEGER := RowPropertyCache.FIRST;
```

Figure 5. Stepping Through the Live Code in the PL/SQL Editor

As well as placing breakpoints directly into the PL/SQL code as shown in Figure 5, you can also “Break on Exception” and nominate one, several, or all exceptions to raise a breakpoint if they occur. For instance, an intermittent NO_DATA_FOUND (ORA-1403) exception would be simple to track down by setting a breakpoint on that exception and then examining the execution stack when it happens.

As well as debugging a single form, the new debugger will step into PL/SQL code on the Database Server and will seamlessly handle the debugging of multiple Forms and libraries. If the executing code calls or opens a module that is not currently open in the builder, then the debugger will even find and open the source file for you!

The new Forms debugger is really a major step forward in productivity, and the ability to debug remote sessions provides an invaluable tool for maintaining Web-deployed applications.

The Forms API goes Java!

The Forms application-programming interface (API) was introduced with Forms 5.0. This provides a supported way for developers to query and manipulate their Forms modules in batch, without having to open the module in the Forms builder. As powerful as the API is, it has one drawback—it is a C-based API, and so is perceived very much as a “Power User” tool.

In Oracle9*i* Forms Oracle is keeping the C API, but also introducing a new Java version, the JDAPI (Java Development API). As well as being a language that everyone wants to learn, Java actually makes the use of the API simpler to understand and a lot more compact. Of course as an additional bonus, the Oracle9*i* Developer Suite comes with a Java Development Environment, in the form of Oracle9*i* JDeveloper, so you already have everything you need to build utilities with the Java API.

Just to show how much simpler it is, Listing 2 shows a sample of C code using the Forms C API, which is just used to get the name of a Form:

Listing 2. Getting the Name of a Form Module Using the C API

```
 /* Create Forms API context */
ctx_attr.mask_d2fctxa = (ub4)0;

if ( d2fctxcr_Create(&ctx, &ctx_attr) != D2FS_SUCCESS )
{
  fprintf(stderr, "Error creating Forms API context\n");
  exit(1);
}

/* Load the form module into memory */
if ( d2ffmdld_Load(ctx, &form, argv[1], FALSE) !=
D2FS_SUCCESS )
{
  fprintf(stderr, "Failed to load form : %s\n", argv[1]);
  exit(1);
}

/* Get the name of the form module */
if ( d2ffmdg_name(ctx, form, &form_name) != D2FS_SUCCESS )
{
  fprintf(stderr, "Error getting the name of the form.\n");
}
else
{
  /* print the name of the form, then free it */
  printf("The name of the form is %s\n", form_name);
  free(form_name);
}

/* Destroy the in-memory form */
```

```
if ( d2ffmdde_Destroy(ctx, form) != D2FS_SUCCESS )
{
  fprintf(stderr, "Error destroying the form module\n");
}

/* Close the API and destroy context */
d2fctxde_Destroy(ctx);
```

Listing 3 shows the same function written in the JDAPI, with the same error-handling capabilities.

Listing 3. Getting the Name of a Form Module Using the Java API

```
try
{
  // Open the module - this implicitly starts JDAPI
  FormModule form = FormModule.open(formName);
  // Call the getName method on the new Form object
  System.out.println("Form Name is " + form.getName());

  // Clean Up
  Jdapi.shutdown();
}
catch (JdapiException jdex)
{
  System.out.println(jdex.toString());
}
```

The Java API contains much less code and is simpler to read and understand (even with comments!).

XML anyone?

The Forms API is a really useful tool that is great for carrying out tasks such as dependency analysis, diff-ing, and bulk changes. There are even tools on the market using the API to provide exactly these services. But, in an attempt to make such operations even simpler and more accessible in Oracle9*i* Forms, Oracle will also be providing a Forms to XML converter. This utility will dump Forms modules into a documented XML format (see Listing 4), which can then be edited or transformed into custom output and reports using stylesheets.

The XML utility is a two-way operation. You can recreate a Forms module from the XML representation, making at an ideal alternative storage format to the FMT file format that is also provided

Listing 4. Sample XML Representation of a Form

```
<Module version="90000500"
        xmlns="http://xmlns.oracle.com/Forms">
  <FormModule Name="ONEBLOCK"
              MenuModule="DEFAULT&SMARTBAR"
              Title="MODULE1">
    <Coordinate RealUnit="3"
                DefaultFontScaling="true"
                CoordinateSystem="1"
                CharacterCellWidth="5"
                CharacterCellHeight="14"/>
    <Block Name="EMP"
           RecordsDisplayCount="5"
           QueryDataSourceName="emp"
           ScrollbarWidth="9"
           ScrollbarLength="135">
    .....
    </Block>
    <ObjectGroup Name="OBJECT_GROUP">
      <ObjectGroupChild Name="EMP" Type="Block"/>
    </ObjectGroup>
  </FormModule>
</Module>
```

Accessibility

With Oracle9*i* Forms, the builder and runtime are compatible with the Job Access With Speech (JAWS) screen reader. Of course other accessibility requirements are addressed, such as usability without a mouse and compatibility with high contrast color schemes.

Miscellaneous Runtime Improvements

A final few features worth mentioning under this productivity section are bundled under runtime improvements.

Cancelable List of Values

You will see a slight change in long List of Values (LOVs) if running in nonblocking mode in Oracle9*i* Forms. For a list that is being populated with thousands of rows, a progress counter will appear at the bottom of the LOV telling you how many records have been brought back so far. Importantly, you can also cancel the LOV population to stop it from getting any more rows and just select from those you have already brought back in to the LOV.

One-Time Where Clause

One-Time Where Clause is a new block property designed to replace the functionality for which most developers still use the Pre-Query trigger. The One-Time Where clause is used to add temporary conditions to the select statement for the block, in addition to the hard-coded where clause and any Query By Example data. As soon as the query has executed, Forms automatically cleans up the One-Time Where Clause so that it will apply to one and only one execution of the query.

Where the One-Time Where Clause is useful is in its use of bind variables. Unlike manipulation of the Query conditions through the Pre-Query trigger, the One-Time Where Clause supports the use of bind variables in the supplied Where condition. Importantly, this can promote database cursor reuse on the server, reducing the server-side resources required by the application.

Getting the Forms Version number

One of the top enhancement requests from customers was the ability to get the version number of the Forms runtime in PL/SQL. Well, here it is: Get_Application_Property(VERSION);

INTEGRATION AND ORACLE9*I* APPLICATION SERVER RELEASE 2 SUPPORT

Let's examine how Oracle9*i* Forms can integrate with the world of Java and the Oracle9*i*AS platform.

Support for 1.3 JDK for the Forms Java Client

With the Oracle9*i* Forms release we are upgrading the Forms Java Client to be able to use 1.3 of the Java Runtime Environment (JRE). In most cases this will make little difference to customers, but those who are writing Pluggable Java Components or trying to integrate the Forms Java Client with other Java applets or Java Beans will be able to use the full power of the 1.3 Java APIs rather than being restricted to the 1.1 APIs that you can use in 6*i*.

In line with this change, a 1.3 JRE based version of JInitiator will also be provided.

Support for 1.3 JRE in the Middle Tier

The Java Importer feature, which we introduced in 6*i*, continues to be a key part of the Forms integration strategy, providing simple-to-use Java integration from PL/SQL. Again, the Java Importer can be used with 1.3 of the Java Runtime, making it possible to integrate Forms applications with the latest external programs and services.

Use the power of the Java Importer to integrate Forms with:

- Web Services / WSDL
- SOAP
- CORBA
- XML

And so on . . .

Enhanced JavaBean Support

As well as allowing the 1.3 Java APIs to be used for JavaBean integration, Oracle has also spent some time making the whole integration of JavaBeans into your code a lot simpler.

In the 6*i* release, in order to integrate a JavaBean into the Forms Client, you have to write some wrapper code in Java. This code acts as a bridge between the standard interface that Forms exposes to pluggable components (the so called IVeiw interface) and the interfaces exposed by the JavaBean itself. In most cases this code is simple, but since it is in Java and not PL/SQL, it is one more thing to maintain.

Oracle9*i* Forms adds a new form of JavaBean integration using a PL/SQL-based interface, the **FBEAN** PL/SQL package.

Using FBEAN, you can programmatically register a JavaBean at runtime, get and set properties and execute methods, all from within PL/SQL, without having to write any extra Java code! Even if your JavaBean is complex and uses data types that cannot readily be translated from the basic PL/SQL types, the FBEAN interface is extensible using Java and can be tailored to your requirements.

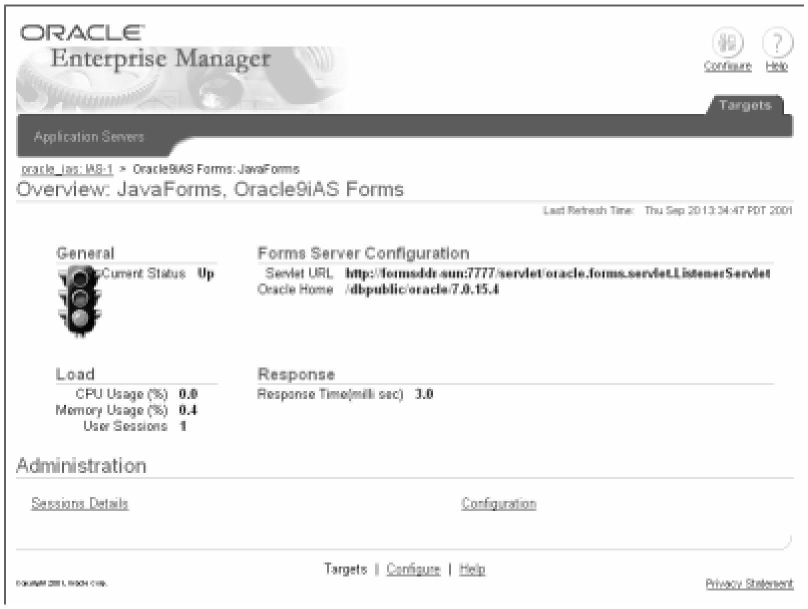
The enhanced JavaBean support also provides full event integration between the JavaBean and the Form, making the embedded component fit seamlessly into the user interface.

The Enhanced JavaBean support will also mange heavyweight JavaBeans (those based on the AWT classes). Heavyweight controls do not co-exist easily in a lightweight Java framework such as that used by Forms. You will see problems when trying to move windows containing heavyweight controls where the heavyweight control does not move with the window or does not get clipped correctly. Fortunately the enhanced bean support handles these issues and ensures that these operations work correctly, without the need for writing code.

Oracle Enterprise Manager (EM) Integration

Oracle Enterprise Manager provides a way of managing all of your Oracle components from a single console. The Oracle9*i* AS Forms Services can be managed through the new browser-based user interface for Oracle Enterprise Manager that will be provided with Oracle9*i*AS.

Through the console, the Forms Services can be monitored for information such as response time, number of connected sessions, CPU, and memory usage (see Figure 6).



oracle_ias: IAS-1 > Oracle9iAS Forms: JavaForms

Overview: JavaForms, Oracle9iAS Forms

Last Refresh Time: Thu Sep 20 13:34:47 PDT 2001

General

Current Status Up

Forms Server Configuration

Servlet URL http://formsddr-sun:7777/servlet/oracle.forms.servlet.ListenerServlet
Oracle Home /dbpublic/oracle/7.0.15.4

Load

CPU Usage (%) 0.0
Memory Usage (%) 0.4
User Sessions 1

Response

Response Time(milli sec) 3.0

Administration

Sessions Details Configuration

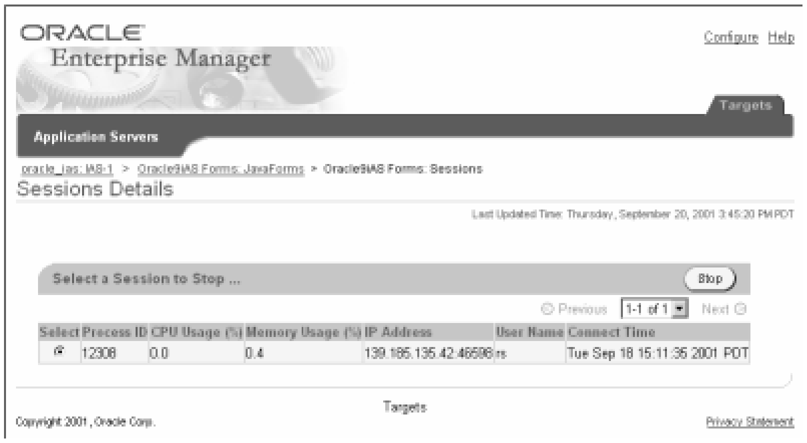
Targets | Configure | Help

Copyright 2001, Oracle Corp.

Privacy Statement

Figure 6. Managing Oracle9*i*AS Forms Services with Oracle Enterprise Manager

You will also be able to drill down into a particular session (see Figure 7) and even kill it (providing you have the correct credentials) right from your browser.



oracle_ias: IAS-1 > Oracle9iAS Forms: JavaForms > Oracle9iAS Forms: Sessions

Sessions Details

Last Updated Time: Thursday, September 20, 2001 3:45:20 PM PDT

Select a Session to Stop ...

1-1 of 1

Select	Process ID	CPU Usage (%)	Memory Usage (%)	IP Address	User Name	Connect Time
●	12308	0.0	0.4	139.185.135.42:46598	rs	Tue Sep 18 15:11:35 2001 PDT

Targets

Copyright 2001, Oracle Corp.

Privacy Statement

Figure 7. Viewing Session Information Through Enterprise Manager

Leverage the Features of the Oracle9*i*AS Platform

Of course Forms does not live in isolation; it is part of the Oracle9*i*AS platform. As such, it can leverage many of the services provided by that platform. Some of these services we have discussed already, but it's worth summarizing them again:

- Oracle9*i*AS Containers for J2EE (OC4J). As well as deploying into the Apache JServ engine, Oracle9*i*AS Forms Services will be deployable within OC4J offering improved performance and scalability.
- Oracle9*i* Gateways for non-Oracle database access. Access non-Oracle databases from any operating system that supports the Gateways, not just the Win32 platforms.
- Single sign-on support. Oracle9*i* Forms provides single sign-on and LDAP integration for free with no coding required in your Forms applications.

UPGRADE AND OBSOLESCENCE

As discussed in the introduction, Forms has been around for many years and continues to evolve. Over the years it has naturally accumulated some excess baggage, which we have retained for the purposes of backwards compatibility. With Oracle9*i* Forms, Oracle has taken the opportunity to finally remove some of these obsolete features from the product. This mostly relates to character mode features such as full screen menus.

The changes that have been made are too numerous to relate here, but a white paper *(Oracle9i Forms: Features Obsolescence)* that documents the changes in detail is available on the Oracle Technology Network (http://otn.oracle.com/products/forms). Oracle will also be releasing a utility to detect and where possible replace any obsolete feature usage that you may have in your modules.

SUMMARY

The good news is that Oracle Forms as a product is continuing to grow and evolve. The features and improvements introduced in the forthcoming Oracle9*i* Forms build on the product's existing strengths, positioning it for truly global Internet use and fully leveraging the capabilities of the Oracle Internet platform.