OpenROAD®

Release Summary

4.1

Computer Associates™

Contents

Chapter 1: OpenROAD 4.1/0109 Enhancements

Chapter 2: OpenROAD 4.1 New Features

Chapter 3: OpenROAD 4.1 Demo Overview

Chapter 4: Application Server Support for OpenROAD 4.1

Chapter 5: Reporter Enhancements

Chapter 6: Upgrading from OpenROAD 3.5 to 4.1

Chapter 7: Virtual Key Values

Chapter 1

OpenROAD 4.1/0109 Enhancements

This guide describes new features or enhancements for OpenROAD.

This chapter outlines the following features found in the 4.1/0109 maintenance release of OpenROAD:

- Installer Enhancements
- Speed Keys Supported
- New System Classes
- New Events
- New Attributes
- New Methods
- Application Server
- Workbench
- OpenROAD Runtime
- Demos
- VASA
- ASOLIB
- UNIX

Installer Enhancements

The following installer enhancements have been made in this release:

- Installation of the Application Server
- Upgrade of Enterprise Access
- Installation of JDBC
- Running VASA in a Runtime-only Installation
- Determining Whether a Reboot Is Recommended
- Installation of Thin Client Demos

Installation of the Application Server

The files necessary to run an Application Server client have been made part of the OpenROAD Runtime component. The Application Server component need only be selected if a server is in fact needed. The server is only supported on Windows NT and Windows 2000. If it were previously installed on Windows 95 or Windows 98 using the OpenROAD 4.1 GA installer, it will be removed by the MR installer.

The Application Server installer (asreg.exe) will now detect whether the Application Server has previously been installed. Access permissions will not be modified if the Application Server has been previously installed. SPO permissions have been corrected so that only SYSTEM will be granted SPO Launch permissions by asreg.

The Domain Portal (DPO) is no longer utilized. It will not be installed or registered in a new installation, and it will be removed and unregistered in existing installations.

Note: Asreg is now only run on Windows NT and Windows 2000. There is no need to run this program on Windows 9x.

Upgrade of Enterprise Access

Enterprise Access to Oracle, Sybase, Informix, and SQL Server can now optionally be upgraded to versions EA 2.1 or EA 2.6 using the OpenROAD 4.1 MR installer, if a prior version of that product has been installed.

Installation of JDBC

In the OpenROAD 4.1 GA installer, the files necessary to install JDBC were provided. However, the product was not installed automatically. As of the OpenROAD 4.1 MR installer, JDBC will be installed along with Ingres/Net.

Running VASA in a Runtime-only Installation

The Visual ASA (VASA) shortcut created by the InstallShield installer now executes via the OpenROAD Runtime executable w4glrun.exe, rather than w4gldev.exe. This enables VASA to be started in a Runtime-only installation.

Determining Whether a Reboot Is Recommended

The installer was designed to advise the user as to whether a reboot of their system was recommended after running the installer, specifically if a locked file were found or the autoexec.bat file needed to be read. Occasionally the user was not prompted to reboot when a locked file was found (usually on applying a patch), or was incorrectly advised to reboot when Ingres was installed on a system without an autoexec.bat, specifically, Windows 2000 or Windows NT. These cases have been corrected.

Installation of Thin Client Demos

Previously some of the files used by the web-based demos were distributed as part of the thin client demo component. This has been corrected.

Speed Keys Supported

The OpenROAD keyboard map file is a text file with a specific format. The file describes 23 OpenROAD system-defined keys, and 36 user-defined keys. The file contains one line for each key with 4 entries for each line. The entries are arranged in four columns separated by tabs. All of the columns are mandatory.

Since the contents of some columns require specific values, it is best to copy one of the existing keyboard map files and use this file as a template if you wish to modify the file or create a new one.

The following table describes the columns:

Column	Description
1	Index that OpenROAD uses to map the key. It is read into memory and must always match the text label in column 4. Do not change the entries in this column. **Note:** Do not change the entries in this column.
2	Text that appears in the pull-down menu for the speed key.
3	Virtual key name for key stroke that occurs.
4	Internal symbol name used by OpenROAD, which must match the index in column 1. **Note:** Do not change the entries in this column.

The following virtual key definitions can be used to define key strokes:

VK_BACK	VK_TAB	VK_RETURN	VK_SHIFT	VK_CONTROL
VK_MENU	VK_PAUSE	VK_CAPITAL	VK_ESCAPE	VK_SPACE
VK_PRIOR	VK_NEXT	VK_END	VK_HOME	VK_LEFT
VK_UP	VK_RIGHT	VK_DOWN	VK_SCROLL	VK_0
VK_1	VK_2	VK_3	VK_4	VK_5
VK_6	VK_7	VK_8	VK_9	VK_A
VK_B	VK_C	VK_D	VK_E	VK_F
VK_G	VK_H	VK_I	VK_J	VK_K
VK_L	VK_M	VK_N	VK_O	VK_P
VK_Q	VK_R	VK_S	VK_T	VK_U
VK_V	VK_W	VK_X	VK_Y	VK_Z
VK_NUMPAD0	VK_NUMPAD1	VK_NUMPAD2	VK_NUMPAD3	VK_NUMPAD4
VK_NUMPAD5	VK_NUMPAD6	VK_NUMPAD7	VK_NUMPAD8	VK_NUMPAD9
VK_MULTIPLY	VK_ADD	VK_SEPARATOR	VK_SUBTRACT	VK_DECIMAL
VK_DIVIDE	VK_F1	VK_F2	VK_F3	VK_F4
VK_F5	VK_F6	VK_F7	VK_F8	VK_F9
VK_F10	VK_F11	VK_F12	VK_F13	VK_F14
VK_F15	VK_F16	VK_F17	VK_F18	VK_F19
VK_F20	VK_F21	VK_F22	VK_F23	VK_F24
VK_NUMLOCK	VK_LSHIFT	VK_RSHIFT	VK_LCONTROL	VK_RCONTROL
VK_LMENU	VK_RMENU	VK_INSERT	VK_DELETE	

If a key is defined in the mapping file it will activate if assigned to an actual menu item. For example, if the VK_RETURN key is mapped to SK_USER1 and SK_USER1 is not used, then it will not change any behavior. On the other hand, if it is assigned to a menu item attached to the frame, then it will override the default behavior and raise that event block.

The same mapping is now used for all platforms. The older definitions in the PC keyboard files will still work. Any keyboard file that used UNIX definitions will have to be changed to use the new virtual key definitions.

Now 104 different key definitions are available. OpenROAD still has a limit of 59 slots available for speed key definitions which are active for the life of a session.

New System Classes

The following are new system classes:

- DecimalObject Class
- KeyDownData Class

DecimalObject Class

Description	The DecimalObject class provides an object version of a decimal value. In most circumstances, the simple decimal variable is the most efficient and useful form of a decimal value. However, sometimes a decimal object is useful. For example, because array row references in OpenROAD must be reference variables rather then simple scalar variables, you cannot have an array that contains scalar decimal elements. However, you can get the same functionality by using an array of DecimalObjects and referring to the Value attribute of the DecimalObject to get or set the data. To perform operations on a decimal object, one must first set the object's Precision and Scale attributes. After setting precision and scale, use the Value attribute as follows: DecimalObj.Value = 123.456
Inherits From	Object
Inherited By	None
Attributes	Precision Scale Value
Methods	None

Precision Attribute

Data Type	integer
4GL Access	RW
Description	The Precision attribute has a default value of 31. This attribute determines the total number of digits of the Value attribute.

Scale Attribute

Data Type	integer
4GL Access	RW
Description	The Scale attribute has a default value of zero. This attribute determines the total number of digits to the right of the decimal point of the Value attribute. The Scale attribute cannot exceed 15.

Value Attribute

Data Type	decimal
4GL Access	RW
Description	The Value attribute stores the value for a decimal object as a simple variable. You can set this attribute to store a decimal value in the object and access the attribute to retrieve the value.

KeyDownData Class

Description	The KeyDownData class contains information about the key that caused the KeyDown or ChildKeyDown event.
Inherits From	Object
Inherited By	None
Attributes	IsExtended Modifiers PreviousState ScanCode VirtualKey
Methods	None

IsExtended Attribute

Data Type	integer
4GL Access	R
Description	This attribute is TRUE if the key is an extended key and FALSE otherwise.

Modifiers Attribute

Data Type	smallint
4GL Access	R
Description	This attribute indicates which modifier keys are pressed at the time of the KeyDown or ChildKeyDown event. The value of the Modifiers attribute allows one to determine whether the key is modified by the Shift, Ctrl, or Alt keys, and also allows one to distinguish between the left and right modifier keys.

This attribute contains bits that indicate which modifier keys are down. Three bits indicate the state of the modifier keys without distinguishing between the left and right keys:

1	Shift
2	Control
4	Alt

In addition to these bits, there are other bits that allow one to determine the state of the left and right modifier keys.

16	Left Shift
32	Right Shift
64	Left Control
128	Right Control
256	Left Alt
512	Right Alt

Although the 4GL language does not provide for explicit bit testing, it can be done using the *mod* function. For example, the following code tests to see if the left control key is down (where KeyDown is a KeyDownData object):

```
if (mod(KeyDown.modifiers, 128) >= 64) then
  /* Left control key is down. */
else
  /* Left control key is up. */
endif;
```

PreviousState Attribute

Data Type	smallint
4GL Access	R
Description	This is the previous key state bit from the WM_KEYDOWN or WM_SYSKEYDOWN message.

ScanCode Attribute

Data Type	integer
4GL Access	R
Description	This attribute contains the scan code associated with the key.

VirtualKey Attribute

Data Type integer

4GL Access RW

Description Contains the virtual key code for the key. Note that the code distinguishes between the left and right shift, control, and alt keys. The virtual key codes for these keys are:

Key	Virtual Key Code
Left Shift key	160 (VK_LSHIFT)
Right Shift key	161 (VK_RSHIFT)
Left Control key	162 (VK_LCONTROL)
Right Control key	163 (VK_RCONTROL)
Left Alt key	164 (VK_LMENU)
Right Alt key	165 (VK_RMENU)

The developer can intercept a key down message and prevent further processing of the WM_KEYDOWN or WM_SYSKEYDOWN message by setting the VirtualKey attribute to zero in the KeyDown or ChildKeyDown event block, as in the following example:

```
/* Note: the virtual key code for the A key is 65. */

on ChildKeyDown =
declare
   KeyData = KeyDownData default NULL;
   VirtualKey = integer not null;
enddeclare
begin
    KeyData = KeyDownData(CurFrame.MessageObject );
    VirtualKey  = KeyData.VirtualKey;

    /*  Do no further processing of the A key  */

   if (VirtualKey = 65) then
     KeyData.VirtualKey = 0;
   endif;
end;
```

New Events

The following are new events:

- ChildKeyDown Event
- KeyDown Event

ChildKeyDown Event

Purpose

The ChildKeyDown event is triggered whenever OpenROAD receives a WM_KEYDOWN or WM_SYSKEYDOWN message. The event is not queued, but is delivered immediately. Information about the key is stored in the KeyDownData object, which is delivered to the 4GL program in the MessageObject attribute of FrameExec. The 4GL event code has the opportunity to stop further processing of the WM_KEYDOWN or WM_SYSKEYDOWN message by modifying the KeyDownData object. To discard the message, the developer should set the KeyDownData.VirtualKey to zero. The PrintScrn key does not cause a ChildKeyDown event and no other events are generated in conjunction with a ChildKeyDown event. In particular, the ChildKeyDown event does not cause either a SetValue or a ChildSetValue event.

See the Virtual Key Values chapter for a listing of virtual key codes and their values.

In the following example processing of the 'A' Key is stopped:

```
/* Note: the virtual key code for the A key is 65. */

on ChildKeyDown =
declare
   KeyData = KeyDownData default NULL;
   VirtualKey = integer not null;
enddeclare
begin
   KeyData = KeyDownData(CurFrame.MessageObject);
   VirtualKey  = KeyData.VirtualKey;

   /*  No further processing of the A key  */

   if (VirtualKey = 65) then
     KeyData.VirtualKey = 0;
   endif;
end;
```

Syntax

on ChildKeyDown

Attributes

The following attributes of the FrameExec class can be used in the ChildKeyDown event block:

Attribute	Description
MessageObject	A KeyDownData object that contains information about the key that caused the ChildKeyDown event.

See Also

MessageObject Attribute

KeyDown Event

Purpose

The KeyDown event is triggered whenever OpenROAD receives a WM_KEYDOWN or WM_SYSKEYDOWN message. The event is not queued, but is generated immediately. Information about the key is stored in the KeyDownData object, which is delivered to the 4GL program in the MessageObject attribute of FrameExec. The 4GL event code has the opportunity to stop further processing of the WM_KEYDOWN or WM_SYSKEYDOWN message by modifying the KeyDownData object. To discard the message, the developer should set the KeyDownData.VirtualKey to zero. The PrintScrn key does not cause a KeyDown event and no other events are generated in conjunction with a KeyDown event. In particular, the KeyDown event does not cause either a SetValue or a ChildSetValue event.

See the Virtual Key Values chapter for a listing of virtual key codes and their values.

In the following example processing of the 'A' Key is stopped:

```
/* Note: the virtual key code for the A key is 65. */

on KeyDown =
declare
   KeyData = KeyDownData default NULL;
   VirtualKey = integer not null;
enddeclare
begin
   KeyData = KeyDownData(CurFrame.MessageObject);
   VirtualKey  = KeyData.VirtualKey;

   /*  No further processing of the A key  */

   if (VirtualKey = 65) then
     KeyData.VirtualKey = 0;
   endif;
end;
```

Syntax on KeyDown

Attributes The following attributes of the FrameExec class can be used in the KeyDown event block:

Attribute	Description
MessageObject	A KeyDownData object that contains information about the key that caused the KeyDown event.

See Also MessageObject Attribute

New Attributes

The following classes have new attributes:

- EntryField Class
- FrameExec Class
- Object Class
- SessionObject Class

EntryField Class

Attributes ExitBehavior

ExitBehavior Attribute

Data Type integer

4GL Access RW

Description The ExitBehavior attribute controls the action of the tab key and/or carriage return key in an EntryField object. This attribute is ignored if the IsMultiLine attribute is set to FALSE. If the IsMultiLine attribute is set to TRUE the following values are valid for the ExitBehavior attribute:

Valid Value	Description
EB_NONE	Default behavior
EB_TAB	A <TAB> key will exit an EntryField object

Valid Value	Description
EB_ENTER	A <CR> will exit an EntryField object
EB_ALL	Both the <TAB> and <CR> keys will cause an exit of the EntryField object.

FrameExec Class

Attributes NextTargetField

NextTargetField Attribute

Data Type ActiveField

4GL Access R

Description The NextTargetField attribute is used to substitute for the TargetField attribute of FrameExec in cases when the user action is going to set the input focus to a field that is not yet present on the frame. The NextTargetField attribute is set only if the TargetField attribute is set to NULL.

The TargetField attribute can be set to NULL when:

1. The user clicks on an append row of a tablefield.
2. The user action will scroll the tablefield.

It is not mapped since the Exit event has not completed and the 4GL code might still do a "Resume" which would prevent it from completing. NextTargetField is set in this case. Not all of its attributes are reliable, and the 4GL code should not try to set any attributes of the field. However, attributes which never vary for the tablefield column will be reliable.

The NextTargetField attribute can be used to confirm that the target is in a tablefield and to determine which column of the tablefield is the target.

Object Class

Attributes InstanceIdentifier
InstanceReferences

InstanceIdentifier Attribute

Data Type	integer
4GL Access	R
Description	The InstanceIdentifier attribute contains a value that uniquely identifies this instance of an allocated object. The value is guaranteed to be unique among the set of instantiated objects for an OpenROAD session. It is valid only during the life of this object. When this object is freed, the value is available for reuse.

InstanceReferences Attribute

Data Type	integer
4GL Access	R
Description	The InstanceReferences attribute contains the current number of references to this instantiated object. A value of zero represents a permanent object that cannot be freed.

SessionObject Class

Attributes	ProcessWait ProcessWindow

ProcessWait Attribute

Data Type	integer
4GL Access	RW
Description	The ProcessWait attribute is used to specify whether commands executed by the "call system" statement block the execution of the currently running OpenROAD application until they complete. The default value is TRUE, indicating that the OpenROAD application will wait while executing statements via the "call system" statement. A value of FALSE will allow an OpenROAD application to continue executing 4GL statements after issuing a system command via the "call system" statement.

ProcessWindow Attribute

Data Type integer

4GL Access RW

Description The ProcessWindow attribute is used to specify whether commands executed by the "call system" statement run in a process that has a window. The default value is TRUE, indicating that executing statements via the "call system" statement will create windows. A value of FALSE indicates that system commands run via the "call system" statement will not have process windows.

New Methods

The following classes have new methods:

- LongByteObject Class
- StringObject Class

LongByteObject Class

Methods ConvertFromString

ConvertFromString Method

Description The ConvertFromString method loads the object from a StringObject that is assumed to hold a representation of binary data, which has been converted to text.

```
integer = LongByteObject.ConvertFromString(
          string = StringObject, format = integer );
```

The format parameter is required and specifies the data format of the text in the string parameter. Valid settings are:

Valid Settings	Description
EF_HEX	Each pair of hexadecimal characters in the string is expected to represent a single byte of binary data. For example, the string "00017F" is loaded as three bytes containing the values 0, 1, and 127.

Valid Settings	Description
EF_BASE64	The string is expected to represent binary data that has been converted to text using the Base64 Content-Transfer-Encoding1 algorithm. As an example, the string "AAF/" is loaded as three bytes containing the values 0, 1, and 127.

This method returns ER_OK if successful. If invalid characters are found outside the alphabet of the specified format or if any other errors occur, the method will return ER_FAIL.

StringObject Class

Methods ConvertFromBinary

ConvertFromBinary Method

Description The ConvertFromBinary method loads the object from the binary contents of a LongByteObject. This provides a way of converting binary data into a text string.

```
integer = StringObject.ConvertFromBinary(
          binary = LongByteObject, format = integer );
```

The format parameter is required and specifies how the binary contents of the LongByteObject are converted into text. Valid settings are:

Valid Settings	Description
EF_HEX	Each byte is converted into a two-character hexadecimal representation of that byte. For example, the three bytes containing the values 0, 1, and 127 are copied into a string with a value of "00017F". Using this format will increase the size of the data by 100 percent.
EF_BASE64	Base64 Content-Transfer-Encoding algorithm is used to convert binary data into a string. As an example, the three bytes containing the values 0, 1, 127 are copied into a string with a value of "AAF/". Using this format will increase the size of data by about 33 percent.

This method returns ER_OK if successful. If invalid characters are found outside the alphabet of the specified format or if any other errors occur, the method will return ER_FAIL.

Application Server

The following application server enhancements have been made in this maintenance release:

- Authorized Applications Only
- Forced Shutdown
- DPO
- SPO Launch Permissions

Authorized Applications Only

If the new RunAuthorizedAppsOnly option is selected in the VASA SPO configuration options, the SPO will refuse to Initiate any application images which have not been registered. An "Unauthorized application" error 0x8004E300 is returned on the Initiate call.

Forced Shutdown

When the OpenROAD Application Server service (orsposvc) is stopped, the SPO will now shut down regardless of how many other clients are connected.

Calls already in process are given a grace period to complete. Further calls from existing clients are refused (with a "Dispatcher disabled due to shutdown" error 0x8004E000). Attempts by new clients to connect are refused by COM (with an "Access denied" error 0x80070005). The SPO terminates as soon as all its ASO slaves have finished processing their current calls, or when the grace period expires, whichever comes first.

The ShutdownGracePeriod can be set in the VASA SPO configuration options, and it defaults to 15 seconds. If a call does not complete within the grace period, the ASO processing that call will be disconnected, and the SPO will terminate. When a disconnected ASO finally completes the call, the byref results are discarded and the ASO begins a normal shutdown sequence.

DPO

The DPO (OpenROAD.DomainPortal routing server) has been removed from this release (and will be automatically uninstalled and erased from existing installations) because it created a potential security risk. It has been replaced by a more direct method for overriding the COM default authentication level of the client machine and requesting unauthenticated connections.

If a Routing string of "unauthenticated" is specified on an Initiate call, an explicitly unauthenticated connection is established directly to the target server. Without this, COM would first attempt to establish an authenticated connection, and only after failing that would it fall back to an unauthenticated connection. Under some network configurations, those failed attempts can significantly impact performance. An explicitly unauthenticated connection also overrides the default authentication level of the client process.

The Initiate Routing string "OpenROAD.DomainPortal" is still supported for backward compatibility, but it no longer causes connections to be forwarded through a DPO server; it simply enables an explicitly unauthenticated connection, directly connecting to the target server. The new Routing string "unauthenticated" is equivalent, and is the preferred way to achieve that result.

SPO Launch Permissions

The installer now sets the SPO launch permissions to include only the SYSTEM account, which is the account under which the OpenROAD Application Server service (orsposvc) runs. This enables the lifetime of the SPO to be fully controlled by the orsposvc, by preventing other users from launching an SPO when the orsposvc is trying to shut down.

Workbench

The following workbench enhancements have been made in this maintenance release:

- Application Workbench Frame
- Connections Profile Edit Frame
- Script Editor Frame
- Reporter Query Editor Frame
- Reporter Component

Application Workbench Frame

Full Row Selection has been added under the Options menu. This allows the listview selection to span columns if selected.

Edit Connections Profile Frame

The -x flag has been removed. It can be accessed from Other flags if necessary.

Script Editor Frame

A search list OptionField has been added to the toolbar. This OptionField contains a list of the most recent search strings. To use, enter a string to find in the OptionField and then click on the Find Next icon.

A Match Case icon has been added to the toolbar. Use this to add case sensitivity to your searches. Toggle the icon on to do case-sensitive searches and off to do case-insensitive searches.

Reporter Query Editor Frame

Changed the Columns... menu item under the Edit menu to View Columns.… .

Reporter Component

The prompt before deleting a report now only happens once.

The delete button in the reporter catalog color now matches other buttons in the frame.

OpenROAD Runtime

The following OpenROAD runtime enhancements have been made in this maintenance release:

- ExtObjects
- Reporter
- TableFields

ExtObjects

One can now use the byref clause on parameters to pass and receive data in an ExtObject (ActiveX) method.

Reporter

New exported 3GL function **rep_print_document()** exposes the repprint.exe functionality.

New flags have been added to reprint.exe which allow the developer to specify the behavior of the confirmation popup. Specifying -cy has the current default behavior of asking for confirmation before printing. Specifying -cn suppresses the confirmation popup before printing. This allows unattended batch jobs that send output to the printer to run.

TableFields

Mousewheel scrolling is now supported on tablefields. To scroll using the mousewheel, hover the cursor over the tablefield to be scrolled and move the mousewheel.

Changes to a protofield's property option menu are reflected in the other rows belonging to the columnfield. Scripts associated with other rows continue to execute.

Demos

The following demo enhancements have been made in this maintenance release:

- UNIX
- Windows

UNIX

Use the same demo applications for 3GL and ESQLC procedures as on Windows.

Windows

Meeting Point and global.asa now use the new routing parameter.

VASA

The following VASA enhancements have been made in this maintenance release:

- SPO Details Pane
- ASO Details Pane
- Disable New Connections
- Auto-Suspend
- Stateless Application Housekeeping
- User-Defined Procedure - iiasohousekeep
- Support for RP_LOCAL

SPO Details Pane

The SPO Details pane has been enhanced to allow maintenance of the new SPO registry entries RunAuthorizedAppsOnly and ShutdownGracePeriod.

ASO Details Pane

Added maintenance of Routing registry entry which replaces the DPO (OpenROAD.DomainPortal routing server) authentication override mechanism. Also, visual enhancements have been made to the pane.

Disable New Connections

Added the ability to manually disable an AKA Detail registration, preventing new connections via the Name Server, but still allowing the ASO and any existing users to be monitored. A remote node can also be disabled manually.

Auto-Suspend

There is a new auto-suspend configuration option to automatically suspend an AKA Detail if housekeeping of that node fails. This provides automated failover causing new connections to be directed to other AKA Details with the same name. Remote nodes will also be suspended automatically if there are zero AKA Details available at the remote node (and the remote node is being monitored by the current VASA instance).

Stateless Application Housekeeping

Housekeeping of stateless applications is also now possible. This can be configured in VASA and a housekeeping log is displayed.

User-Defined Procedure - iiasohousekeep

A new procedure, **iiasohousekeep**, can be overridden by a user procedure with the same name. This enables user defined housekeeping functionality to be implemented.

Support for RP_LOCAL

Nodes can be configured with server type = RP_LOCAL. This means that the NameServer can be used in a test environment where the server application is included in the client application.

ASOLIB

The following ASOLIB enhancements have been made in this maintenance release:

- COM Errors
- NameServer
- XML - in
- XML - out
- Pre-process, post-process User-Written Routines for XML-in, XML-out
- Load Balancing

COM Errors

COM error codes are now logged in HEX to make them easier to interpret, and all REMOTESERVER values are also logged.

NameServer

The 4GL client NameServer component ASONameServer now accepts optional parameters specifying the Location and Routing of the NameServer. This means that the machine's default DCOM settings can be overridden when connecting to the NameServer from 4GL.

XML-in

Global Service Call Procedures (GSCPs) can now handle XML-in as well as XML-out. This means that some or all of the "in" parameters to a GSCP can be contained within an XML document.

XML-out

Global Service Call Procedures (GSCPs) XML-out generation will now traverse inherited userclasses.

Pre-process, post-process User-Written Routines for XML-in, XML-out

It is now possible to call a user-written routine to pre-process the XML-in when it is first received from the client. Similarly, a user-written routine can be used to post-process the XML-out before it is returned to the client. This enables user modification of the XML. For example, an XSL Transformation could be applied using a third-party XSLT control.

Load Balancing

Server farm connection load balancing has been improved to dynamically maintain connection counts for remote installations. Previously, remote installations were allocated connections on a round robin basis.

UNIX

Added VASA to the installation package.

Chapter

2

OpenROAD 4.1 New Features

OpenROAD 4.1 represents a logical evolution of the OpenROAD 4.0 product yet it provides a significant number of enhancements that are designed to allow the creation of better looking applications, allow the creation of N-Tier applications and support the development and deployment of OpenROAD applications on UNIX platforms. OpenROAD 4.1 is built on the OpenROAD 4.0 functionality. Migration from OpenROAD 4.0 to OpenROAD 4.1 is a simple process. By default, an existing application running under OpenROAD 4.1 will look very similar to one running under OpenROAD 4.0. The enhancements to the look and feel of OpenROAD can be exploited as they are desired.

Property Option Menu Enhancements

Create Property Option Menus at application runtime and associate them with a CompositeField.

TableField Enhancements

Add the listview look and feel to tablefields. The following features were added:

- Button Headers
- Resizable column
- Display a subset of columns (Horizontal Scrolling)

Keystroke Events on EntryFields

Allow application developers to monitor keystrokes in entryfields: New events KeyPress, ChildKeyPress, BufferChanged, and ChildBufferChanged events. Added new system class KeyPressInfo that contains keystroke information.

Auto-completion for OptionFields

Provided a new mode, OF_AUTOFIND, that provides an auto-completion feature that allows a user to select an item from the list by typing one or more initial characters of the desired item.

New Field Styles

Introduce a new style OS_STANDARD to provide the standard Microsoft style for ScalarFields and CompositeFields.

SizeGrip

Introduce a Microsoft-like SizeGrip in the lower right hand corner of resizable frames.

True Type Fonts

Currently, OpenROAD provides limited support for fonts. The introduction of True Type (native) fonts provides application developers with greater flexibility in designing OpenROAD frames. Among other things, True Type fonts allow developers to scale field sizes.

Button Styling

The new OS_STANDARD style now allows buttons to conform to the Microsoft standard.

Progress Bars

OpenROAD works with Microsoft Progress Bar ActiveX control. The ActiveX control that this feature is based on is only available under Windows.

Group Boxes

A Group Box contains a number of logically related objects. It consists of an outline with a beveled border and a label, which describes the contents of the box. Subforms and Flexibleforms can now have a groupbox label.

Wizard Frames

Enhance tabfolders so that a wizard frame can be created.

Icon Images

OpenROAD 4.0 support for icons is extremely limited. An icon file can contain multiple images. The OpenROAD 4.0 implementation only uses the first icon image in the icon file. It treats icons with transparent colors like a monochrome bitmap.

An icon may have a transparent area that is represented by the "transparent color." When an icon is placed on a form, OpenROAD 4.1 will honor the transparent color and allow colors below the transparent area of the icon to "show through."

24-bit Color Bitmaps

OpenROAD 4.0 supports color images up to 8-bit color resolution. OpenROAD 4.0 can read 24 bit color images, but internally it converts these to 8 bit images. When images are converted, a significant amount of color information is permanently lost resulting in a very grainy picture. The internals of the BitmapObject have been modified to remove this restriction.

System Color Remapping

Bitmaps can be rendered differently in the Microsoft development environment. They can either be *loaded* as is, which would typically be done with pictures and photographs used pictorially, or they can be *re-mapped* with system colors. Re-mapping is typically done if the bitmap is going to be used on a ButtonField, MenuField, ToolBar or anywhere where a rectangular image is boring and quite often unattractive. Microsoft defines exactly how re-mapping is to occur.

Three colors are defined that will be re-mapped:

Color	RGB Setting	Result
CC_SYS_BTNFACE	RGB(192,192,192)	mapped to the system button face Gray
CC_SYS_BTNSHADOW	RGB(128,128,128)	mapped to the system Shadow color Dark Gray
CC_SYS_SHADOW	RGB(233,233,233)	mapped to the system Highlight color Light Gray

These will then need to be re-mapped whenever the system colors are changed. This feature will allow the placement of what appears to be non-rectangular bitmaps on OpenROAD FormField objects.

RGB Color support

OpenROAD supports a user customizable color palette along with a non-customizable system palette. This places an arbitrary limit on the amount of colors available for application use. Specifying a palette index currently sets color attributes. Allowing color attributes to be set with either a palette index or a RGB value will lift the color limit.

A new system-defined function is provided to return an integer representing an RGB color value.

RGB Function

Purpose

Returns an integer number representing an RGB color value.

Syntax

RGB (red, green, blue)

Arguments	Data Type	Description
red	integer	0-255, inclusive, that represents the red component of the color
green	integer	0-255, inclusive, that represents the green component of the color
blue	integer	0-255, inclusive, that represents the blue component of the color

Returns

Returns an integer number representing an RGB color value

Drag and Drop

Support drag and drop (DAD). This is the ability to drag information from one activefield to another. DAD is not dragging one field to another to invoke some code; it is moving data contained in one field to another. DAD allows OpenROAD Application Developers to add this functionality to their application. With OpenROAD DAD, users will be able to specify the type of data that is being transferred (including user classes) and thus, users will have great flexibility in creating objects that can be used for transferring data.

For example, in a DAD enabled OpenROAD application, the user could select an imagefield representing a database table, and drag and drop it over a tablefield which gets populated with data from the database table.

OpenROAD DAD will support the following DAD features:

- Keyboard state during a drag operation
- Drag operation will be aborted if Esc key is pressed
- Cursor change when pressing the Control key during a drag operation

OpenROAD DAD will have the following restrictions:

- Drag and drop will not be allowed among several OpenROAD applications
- Drag and drop between several forms within the same application, however, will be allowed
- No support for dragging objects from OpenROAD to non-OpenROAD applications
- Dragging will be supported with the left (primary) mouse key only

Standard Toolbars Bitmaps

OpenROAD 4.1 provides a set of bitmap files that users can utilize for developing their applications. Application developers can now add new toolbar bitmaps to the palette and create and maintain their own galleries of toolbar bitmaps.

Flat Toolbars with Hot tracking

Another new Microsoft interface style that has become popular is the "flat" toolbar. In a flat toolbar, each button takes on a 3D look only when the mouse cursor tracks over it. This is known as "hot tracking."

A new button style attribute BS_TOOLBAR has been introduced to give this behavior to an OpenROAD button.

Spin Controls

OpenROAD 4.0 provided a field template called spin_control that creates an entryfield with an associated pair of buttons that allow the user to increment or decrement the field. This did not look like the native Microsoft UpDown control. OpenROAD 4.1 now has a spin control template that looks more like the Microsoft control.

Date Picker

OpenROAD has been certified to use the DateTimePicker ActiveX control provided by Microsoft. The ActiveX control that this feature is based on is only available under Windows.

CompositeField Enhancements

The OpenROAD 4.0 application development environment allows a developer to group various types of visual fields into a single field known as a CompositeField. The purpose of a CompositeField is twofold: (1) It allows the various fields within a CompositeField to be treated as named attributes of a parent class, thus inheriting many of the benefits of object-oriented programming and (2) it allows the creation of portable visual grouping of form fields that maintains inter-field geometry regardless of deployment platform.

When a CompositeField is used to represent the attributes of an OpenROAD userclass, a restriction comes into play: the CompositeField can contain only fields contained in the userclass. One cannot place "foreign" entryfields or buttonfields in a CompositeField based on a userclass and take advantage of both Object-oriented and portability benefits of CompositeFields. This restriction has been relaxed.

ActiveX Error Handling Enhancements

The OpenROAD 4.0 application development environment allows a developer to invoke methods and access attributes of an ExternalClass object containing an ActiveX component. In many instances the specification of an ExternalClass method invocation or attribute access does not contain a status indication, in that the operation either fails or succeeds, with the operation target either containing junk or a valid external class component. There was no way to differentiate that an ExternalClass object contained a valid object using the OpenROAD 4GL. Subsequent operations on invalid or junk objects lead to more problems, such as strange execution patterns, trashed data, application crashes, and lost data.

OpenROAD 4.1 now provide a way to indicate to a 4GL program that the last ExternalClass operation has failed and provides detailed information as to why the failure occurred.

Userclass Object Allocation Limit Increase

Add the ability to create virtually an unlimited number of userclass cbjects. The new limit is now 2 billion. The ability to create a large number of Userclass objects allows developers to design applications that leverage the strengths of Object-Oriented programming unencumbered by unreasonable limits. The old limit was 32,760.

Report Writer Conversion to OpenROAD

OpenROAD includes Reporter, a repository-based, report writer utility that allows reports to be viewed or printed. This report development tool feature highly sophisticated graphic capabilities. This feature provides a new utility to convert Ingres Report-Writer reports to Reporter reports.

StackField Separator (Screen divider)

Provide OpenROAD users with the ability to create forms that contain two or more adjacent subforms that can be re-sized using a divider between these subforms.

For example, in Windows NT Explorer, there are two areas: Folders, and Contents. The two areas can be re-sized by dragging the divider between those two areas.

Minimizing Informational Messages in Trace Windows

A new parameter was added to the application startup command string. This parameter notifies the application to disregard the informative messages.

Destroy a Single Component Flag in DestroyApp

Provide users of the DestroyApp utility with the ability to identify individual components of an application to be destroyed. Previous versions of the utility only allowed destruction of the entire application.

Reporter API Documentation

Reporter is a tool for designing reports in a graphical-layout manner similar to the design of frames in OpenROAD applications. Reporter generates a 4GL procedure to take care of the data retrieval and organization in report form, and a 4GL API to a set of 3GL routines for the actual generation of an intermediate output file. A separate process is responsible for the final interpretation of the intermediate file and sending output to a printer.

The recent changes to Reporter allow run-time support for converting dynamic Report procedures into static procedures. Also, the introduction of the RWConv utility offers users their first exposure to the 4GL API. These changes allow users to write code that utilizes the API in a supported fashion.

Make Defining Break Columns Optional in Reporter

OpenROAD 4.0 Reporter provides three options to make page configuration of a report and they are form report presentation, tabular report presentation and page layout template report presentation. The tabular report presentation allows columnar-style reports, which are suitable, for example, for sales and inventory reports. For the tabular report presentation, Reporter requires a break column be defined so that when its value changed, a page break will be made and page footer if there is one will be printed. A lot of customers, especially new Reporter users, have complained about this requirement. They want to make this requirement optional so that they can print a table content and put as many rows as possible in a page.

OpenROAD 4.1 has removed the restriction that prevents user from continuing if no break column is defined and has added logic to handle page breaks based on a page length parameter when no break column is defined for a tabular presentation report.

New OpenROAD Workbench Startup Options

OpenROAD Workbench has now been enhanced to allow new startup options. When the OpenROAD Workbench is started, it will now accept a command line flag to skip the splash screen.

For example:

```
w4gldev runimage workbnch.img -/appflags nosplash
```

A new command line flag has been provided that allows a user to specify a connection profile on the command line.

This is as follows:

```
w4gldev runimage workbnch.img -/appflags profile=profilename nosplash
```

When specified on the command line, the profile name may not contain embedded white space.

Chapter

3

OpenROAD 4.1 Demo Overview

OpenROAD 4.1 is best understood by the examination of the demos. The demos have been made an integral part of the OpenROAD 4.1 product. They have been designed to be simple and easily understood. They will be the starting point that many will use to understand the advanced capabilities of OpenROAD 4.1

A lot of work has been put into the creation of these demos. The source code for all demos has been provided and you are free to examine or reuse any of it in your applications.

Active Server Page Demo

This demo illustrates how to access OpenROAD from an Active Server Page (ASP). Access to OpenROAD from the ASP is accomplished via the COM-based OpenROAD Application Server.

This is a very simple demo that illustrates how the OpenROAD Application Server can be used to expose business logic to a client. This demo consists of a single OpenROAD Application called SISBL. This application exposes a number of 4GL Service Call procedures.

This application supports two basic demos:

- SISBL ASP access to the demo database
- TMQuery access to the demo database

There is a description of the demos. The source code is provided for all components. The ASP and 4GL source code has been also provided in a hyperlinked format to allow the flow of the requests to be more easily understood.

This demo also contains an application that can be used to load the sample database data. This data load utility has been tested against Ingres, Enterprise Access, and EDBC data sources successfully.

SISUI Demo

This demo illustrates how to access OpenROAD from a thin client OpenROAD Application. This client uses COM/DCOM to communicate with an OpenROAD Application Server to run queries against the database.

This demo accesses the demo database using an OpenROAD Application Server. It uses the same OpenROAD Application Server as the SISBL and TMQuery demos.

Meeting Point Tutorial Demo

This demo employs an integrated help system to provide commentary on four production standard applications:

1. MP 1: Two Tier Fat Client: A traditional fat client application, where graphical presentation, business logic, and database access are combined;
2. MP 2: Three Tier Fat Client: Conversion to stateless 4GL procedures within the Application Server. These procedures perform the database access, and are called from a separate client application;
3. MP 3: Fat Server: Introduces some design techniques to help make the server application as fat and reusable as possible;
4. MP 4: Web Client: Reuses the fat server application, this time driving it with a Web Server and ASP.

The ASP Thin Client application also demonstrates the use of XML with the OpenROAD Application Server. The responses from the OpenROAD Application Server are returned as an XML document. This XML document is sent to the browser. The browser applies the style sheet to generate the user interface.

Object Factory Demo

This demo illustrates an object oriented design technique that can be used within new Application Server systems. The technique can also be used to convert existing fat client systems into a partitioned application. It keeps disruption of client code to a minimum and produces a fat reusable server application.

Chapter 4

Application Server Support for OpenROAD 4.1

The OpenROAD Application Server is a collection of objects and services that supports n-tiered applications written in the OpenROAD 4GL. The Application Server facilities allow OpenROAD 4GL business logic to be shared by a very wide variety of client programming environments and languages, including web server scripting languages.

This chapter outlines the following Application Server features found in OpenROAD 4.1:

- Utilizing COM
- 4GL Remote Procedure Calls
- Automation Types
- Fixed Signature, Dynamic Data
- Structured Data
- The Remote Server Object
- The Parameter Data Object
- The 4GL REMOTESERVER System Class
- Private Server, Shared Server
- Application Server Object Library (ASOLib)

Utilizing COM

The OpenROAD Application Server consists of a set of COM (Microsoft's "Component Object Model") objects exposing COM Automation-compatible interfaces. This helps maximize the number of client environments that can take advantage of the OpenROAD Application Server facilities, and it leverages the extensive support COM provides for local and remote server activation, marshaling of parameters, and configurable security controls.

4GL Remote Procedure Calls

The basic conceptual model is that of a remote procedure call, where the called procedure is an OpenROAD 4GL procedure contained in some specified OpenROAD application image. OpenROAD 4GL procedures can now be called remotely, and called by clients that are entirely outside the context of OpenROAD. Named parameters can be passed by value or by reference. Parameters passed by reference may be modified by the 4GL procedure, and those changes will be mapped back to the caller's copy of the parameters.

Automation Types

To maximize the number of different client languages that can utilize the OpenROAD Application Server, all server facilities are accessed via COM Automation-compatible interfaces. This means that parameters passed to the remote 4GL procedures must be expressed within the set of datatypes supported by COM Automation.

This common denominator provides a rich collection of types, but does not include all the object types (such as System Classes) that an OpenROAD procedure might accept as parameters in a purely OpenROAD context. When an OpenROAD client makes a remote call to an OpenROAD server procedure, the parameter types that can be passed are limited to the set of types that can be converted to and from COM Automation types.

The primary goal of the OpenROAD Application Server is to allow many different client languages to share the same OpenROAD 4GL business logic. It does not (at this time) provide transparent partitioning of OpenROAD applications. Remote 4GL procedure calls are limited to the common denominator of Automation parameter types, and therefore cannot duplicate all the functionality of purely local OpenROAD 4GL procedure calls.

Fixed Signature, Dynamic Data

Instead of constructing and deploying separate proxy/stub code for customized COM signatures for each OpenROAD 4GL procedure, the Application Server uses a generalized method signature, and the data passed through that signature is dynamically mapped to the actual signature of the called 4GL procedure. This is somewhat analogous to the way parameters are passed through the IDispatch Automation interface.

This generalized signature provides an easy-to-deploy baseline facility, which is the only interface directly supported by the Application Server. There is nothing to prevent developers from building their own customized COM objects and method signatures to wrap the underlying interface, but the Application Server facilities do not yet provide any direct assistance for that.

OpenROAD 4GL procedures take named parameters. Parameters are passed through the COM interface via pairs of arrays: an array of parameter names and a corresponding array of parameter values. Constructing these paired arrays can be tedious (or impossible) in languages that have limited support for Automation array types. Therefore, additional objects (such as the Parameter DataObject, and the OpenROAD 4GL REMOTESERVER system class) have been provided to encapsulate the actual array structures and make it easier to construct and access the parameter data.

Structured Data

OpenROAD supports structured data in the form of userclasses and userclass arrays. A userclass is analogous to a row of data with a well-defined set of column types, and a userclass array is analogous to multiple rows of data with the same column types. OpenROAD userclasses and userclass arrays can be nested to any number of levels, allowing the construction of very complex hierarchical data structures.

Fortunately, COM Automation supports two-dimensional arrays, and allows those arrays to be nested to any number of levels. Therefore, OpenROAD structured data can be expressed as nested two-dimensional arrays, where each row represents one userclass instance or one row of a userclass array. The parameter name array can be similarly nested, to create a descriptor that defines not only the names of the parameters at the top level, but also the mappings to names of userclass attributes at all the nested levels. This allows very complex structured data to be passed as parameters to remote 4GL procedure calls.

Creating and manipulating the nested two-dimensional arrays that represent complex structured parameters is not easy, but it is not necessary to manipulate those structures directly. The Parameter Data Object (and OpenROAD 4GL REMOTESERVER system class) encapsulate those array structures and provide a more intuitive syntax for creating and accessing complex structured parameters.

The Remote Server Object

The OpenROAD Remote Server COM object provides the methods that allow a client to connect to a desired OpenROAD application image and to call 4GL procedures in that application image. It is registered under the COM PROGID "OpenROAD.RemoteServer". Methods provided by this COM object include:

- **Initiate** - to connect to an appropriate server process that hosts the specified OpenROAD application image.
- **CallProc** - to call a specified 4GL procedure within that Initiated application, passing named parameters by value or by reference, via the Parameter Data Object.

The Parameter Data Object

The OpenROAD Parameter Data COM object provides an easy way to construct complex nested parameter structures. It is also important for client languages that do not support direct manipulation of Automation types. It is registered under the COM PROGID "OpenROAD.ParameterData". Methods provided by this COM object include:

- **DeclareAttribute** - to declare a named parameter.
- **SetAttribute** - to set the value of a named parameter.
- **GetAttribute** - to retrieve the value of a named parameter that was passed by reference and modified by the called 4GL procedure.

The 4GL REMOTESERVER System Class

Within OpenROAD 4GL code, a REMOTESERVER system class wraps the OpenROAD Remote Server COM object. The Parameter Data Object is not used in this context because the REMOTESERVER system class provides a more direct way to specify parameters for the remote 4GL procedures. Methods provided by the OpenROAD REMOTESERVER system class include:

- **Initiate** - to connect to an appropriate server process that hosts the specified OpenROAD application image.
- **Call4GL** - to call a specified 4GL procedure within that Initiated application, passing named parameters by value or by reference. Parameters are passed using normal 4GL call syntax.

Private Server, Shared Server

The OpenROAD application image that executes the remote 4GL procedure calls can run in either a private server or a shared server.

A private server is a separate server process dedicated to serving one and only one client. This configuration is easy to understand and it avoids the complications that can arise when interleaving concurrent requests from multiple clients in a shared server. But the performance cost is high, because a new server process must be launched each time a client calls the Remote Server Initiate method. Further, it does not scale well to large numbers of clients because each client must have a separate, dedicated server process on the server machine.

A shared server allows any number of clients to connect to the same server process. A shared server allows the Remote Server Initiate request to be very fast, since most of the time a shared server process for the desired application is already active and does not need to be launched or initialized. A shared server also scales better for large numbers of clients, since many clients can share the resources of one server process.

Using a shared server requires special discipline on the part of the application developer because remote 4GL procedure calls from different clients can be interleaved in any order. Remote calls are serialized, so that the 4GL need not manage concurrent threads within the scope of one remote call, but the application must be designed to allow switching client context between one call and the next. This means that care must be taken to associate the appropriate application state with the appropriate client.

One simple technique for managing client state in a shared-server environment is to have each client hold a structure containing its own state variables, and to pass that structure as an extra by-reference parameter on all remote 4GL calls. The server application can then use that structure to restore the current client's application state at the beginning of the call and then save any changes in that client state before returning.

The OpenROAD Application Server can be configured to have a single shared server process for each application, or multiple server processes for a single application. The multiple-server configuration provides automatic load balancing across the multiple server processes, and allows larger numbers of clients to be served efficiently (especially on multiprocessor machines). However, the multiple-server configuration requires even more discipline in the management of application state from one call to the next, because each time a client makes a remote call it may be routed to a different server process.

An optional parameter on the Remote Server Initiate method specifies whether the client wants to use a private or a shared server. The default is to use a shared server.

Visual ASA

A new tool to support the Application Server has been added: Visual Application Server Administrator. This is a Graphical User Interface tool designed to allow administrators to manage and monitor systems running under the OpenROAD Application Server. It works in conjunction with ASOLib (see below).

There are three distinct functions provided by Visual ASA:

- Name Server Maintenance;
- Application Monitoring and Session Analysis;
- Housekeeping Console.

Visual ASA provides an intuitive Explorer style interface for Name Server management. There is one Name Server per AppServer installation. Multiple applications can be registered with one or more Name Servers to provide connection load balancing across processes within a given machine, and across processes on different installations.

Each installation's Server Pooler (SPO) and Application Object (ASO) configuration parameters are maintained using Visual ASA. Details of each application registered with a Name Server can also be maintained and interrogated, and each running application can be monitored.

Application monitoring uses the same interface to drill down into a particular application instance. The application instance can be started and stopped, and individual user sessions can be monitored and traced. Each application's callable interface can also be explored dynamically using the Visual ASA MetaData interface.

Visual ASA also acts as a production system Housekeeping Console, invoking garbage collection and expired session timeout within each application, freeing up memory used by stale objects where appropriate.

A limited functionality browser-based version of Visual ASA is also provided.

Application Server Object Library (ASOLib)

ASOLib is a new 4GL library built into the OpenROAD core. It contains components that can assist with design, development, and deployment of applications for the Application Server.

These components can be used to create persistent Application Server session context. This enables pseudo-conversational dialogue between client and server applications and allows the creation of persistent userclass objects associated with a client session.

This approach can help to achieve fat-server/thin-client systems, where the majority of the business logic, however complex, is contained within the server application.

Chapter

5

Reporter Enhancements

Reporter is an OpenROAD application that allows the design of reports through a graphical interface. The reports are stored in the Reporter catalogues as collections of Reporter-defined userclasses and data extracted from OpenROAD system classes representing the fields on the report.

When a report is run within Reporter the graphical definition is converted to a dynamic 4GL procedure which runs an SQL query and calls API functions to prepare an output file which can then be printed with the *repprint.exe* utility. Since Reporter reports are not applications or components of applications there is no direct way to make an "image" of a Reporter report, or an application containing a report, as can be done with OpenROAD applications developed with Workbench. Prior to the introduction of *enhanced runtime support,* reports had to be run within Reporter and then depended upon the Reporter system catalogues to provide the report definition. This imposed limitations on portability, requiring that when Reports are moved between sites they must be exported from the design database and imported into the runtime database. Customization to the new site may also have been necessary. The bulk loading of reports was only possible by reloading the Reporter system catalogues. Other portability issues are discussed below.

Enhanced Runtime Support

Enhanced runtime support is provided in the following ways:

1. **Reporter will run in a runtime environment** (runimage) allowing viewing of graphical report definitions and running of any reports in the reporter catalogues.
   - This requires access to a database with installed reports
   - Some bugs in this area have been fixed
2. **Improved error reporting**. When a report is run, any errors that occur are written to the trace window. Some, optionally, can be displayed in popup windows. When run without popups (default) the report is non-interactive, thus allowing it to be run in "batch" mode.

In the Setup/Cleanup frame you can add error checking for the SQLcode that you provide. Curprocedure.dbsession.errornumber can be used to detect errors and the Reporter-defined variable "errtxt" can be used to provide the error description:

```
inquire_sql(errtxt = errortext);
```

Reports now return:

```
-1: error
 0: no rows
>0: rowcount or OK
```

3. **More flexible support for runtime tables.** Previous versions of Reporter required that tables to be included in a query must exist at design time. It was possible that, at runtime, the actual tables used may be of different structure as long as the names remained the same and the columns being retrieved were compatible with the design specification.

   Reporter now supports the use of global temporary tables and variable table names, which may be passed as parameters to the report. This is achieved through the use of *example* tables. An example table must exist in the design environment while the report is being specified but need not be kept once report definition is complete. It is required to contain only runtime compatible column definitions that are required by the report. A table that takes part in a query can be one of three example types:

   - **Self –** original Reporter behavior
   - **Temporary –** a global temporary table that may already exist in the session that calls the report (in which case the same database session must be used by the report), or may be created during the setup phase of the report.
   - **Parameter –** the tablename is a report variable that must be initialized at runtime. The table name passed in may be that of a temporary table. It may have a default value.

4. **Reporter reports may be converted to static 4GL procedures,** which can be run from any OpenROAD application independent of Reporter or its catalogues

   If the original report uses image trim the procedure will require additional support:

   - If image trim is stored in the design database, at runtime an "Image Server" frame is necessary to provide access to the images unless a copy of the Reporter design database image catalogues is accessible.
   - If image trim is stored on disk at design time, a variable can be set to point to a directory containing copies of the images on the deployment machine

Changes to Reporter

Changes were made to the following features of Reporter:

- Main menu
- Query editor
- Variable list frame
- Variable properties frame
- Print dialog
- Reporter procedure tool

Main Menu

The following two items were added to the main menu:

Menu Item	Description
File/Compile	Like File/Print, this button will generate the dynamic 4GL procedure for a report without running it. To save or run this procedure, see below.
Tools/Procedure List	This button starts the new Reporter Procedure Tool.

Query Editor

Example is a new option added to the Edit menu of the Query Editor. This option is active when a table has been selected in the graphical table display. An example type may be one of three choices

- **Self**: (Default) The table is an example of a runtime table of the same name. This may include a schema.
- **Temporary**: the table is an example of a global temporary table. Reporter will automatically refer to the table as *session.*<*tablename*>. These tables may be created at runtime by the caller in the same session as the report will be run or may be created as part of Report/Setup using the OpenROAD syntax for *declare global temporary table.*

  To drop a temporary table explicitly use *drop* '*session.*<*tablename*>'.
- **Parameter**: When a table is specified as a parameter you will be prompted to create a variable via the *Variable List* frame. Variables that are associated with tables are always of type varchar and are report parameters.

Notes:

- A temporary table may be a parameter. In this case define the example type as Parameter and pass the table name as *session.<tablename>*.
- Parameter tables may not have schema attached within the Query Editor. If required, pass the schema as part of the tablename.
- In the graphical display area of the query editor, tables that are examples of Temporary tables, are prefixed with *(T)*, and parameter examples are *(P)*.
- Quotes (") are required around the table name for the drop statement.
- When the example type is parameter this option is available to view or edit the associated parameter variable.

Variable List Frame

The "Create" Button is labeled "Create Field" or "Create Param" depending upon the context. It may not be always be visible.

The "Cancel" button is now labeled "Close" to reflect the fact that actions initiated in this frame cannot be undone directly; variables are created and edited by calling the Variable Properties frame (Buttons N and Magnifying Glass). Variables are deleted (trash can) permanently after confirmation if they are not in use.

When the variable list frame is called as a result of the user choosing to create a variable field or table parameter, any changes made to variables are kept whether or not the use selects "Create..." or "Close". Selecting "Create..." associates the selected variable with a field or table parameter; selecting "Close" simply aborts the association but does not affect any changes made to variables.

Variable Properties Frame

Is Table Name is a new indicator in the Expression section of the Variable Properties frame. You set this indicator when the variable is a table name parameter. This field is view only. The Query Editor controls the use of a variable as a table name parameter.

When a variable is a parameter, the entryfield Prompt allows the variable name to be replaced by an explanatory phrase in the Print Dialog Frame.

When a variable is a table name parameter, its datatype is always Varchar and IsParameter is always TRUE.

Print Dialog

The dynamically created dialog frame associated with a report has an additional toggle, Popup Errors. The default is FALSE. This toggle controls whether errors, mainly SQL, are displayed in a popup window. All errors are logged in the trace window in either case.

Report parameters now have an optional prompt or explanation string. Instead of simply listing parameters by name it is possible to display a brief explanation of what information is required.

Reporter Procedure Tool

A dynamic procedure for a report is regenerated whenever a report is Printed or Compiled from the main Reporter window. These dynamic procedures can now be reexecuted (avoiding the retranslation which occurs with Print) during the current session or saved to export files. Exported report procedures may be imported to any OpenROAD 4.1 application (which must include repcomp.img and repopen.img). Some modifications to the dynamic procedures have made to remove assumptions that they are being run within the Reporter environment.

Each report procedure has an associated print dialogue frame, which is normally seen when a report is run within Reporter. This frame optionally may be exported along with the procedure. This frame will provide default parameters for the procedure and will allow the user to set the target database, database flags, temporary directory, and the name of the intermediate file. This frame is not required to run the report and the developer may call the procedure via *callproc*, just like any other 4GL procedure. The call interface to a report is defined later in this chapter.

Reports Using Image Trim

A major part of these changes is related to reports that use Image Trim as part of their output. Note that *Image Trim* is a static part of the report definition used to "beautify" the report and is not user bitmap data retrieved through *Image Fields* from the user's data tables. 4GL procedures cannot contain initialized image trim objects, so the image trim used by a report procedure must be stored elsewhere.

During report definition these images are stored in the database or on disk. This makes porting of dynamic procedures difficult since the location of the required images is encoded within the report. If the images were originally stored in the database, the Reporter catalogues would need to exist and the images would need to be stored with the same unique keys and names. The only way this could safely be accomplished is to load the Reporter catalogues from the original design database.

For images stored on disk the same path to the image files would have to exist on the target machine. This requires bitmap image files to be deployed in addition to the report.

These mechanisms are still available but, in addition, more portable mechanisms are now provided for handling both types of image storage:

Database-resident images

Database-resident images may be copied into an Image Server Frame, which is imported into the target application or an included application of the target application.

Before a report is run, an Image Server variable must be initialized with the Image Server Frame containing Image Trim required by the report. A global Image Server Variable (G_ImgSvr) of type *u_imgsvr* is provided in repopen.img (which must be included in any application that runs report procedures). An application developer may declare additional image server variables but a report procedure always expects to find its images in G_ImgSvr. An Image Server Frame may contain images for any number of reports and mimics the original database-resident mechanism for image storage.

When running a static Reporter-generated procedure from an application, an Image Server is used by default for any image trim originally stored in the design database.

Notes:

- When an Image Server is used it is assumed that all database-resident images are available from the Image Server.
- A report run from the Reporter File/Print menu item will maintain its previous behavior; it will NOT use the Image Server since, in this mode, the report and its images must already exist in the database.

The Procedure Tool provides the means to construct and update an Image Server Frame.

Disk-resident images

Disk-resident images may be installed in a single directory, which is definable at runtime, or via the environment.

The directory containing report image trim bitmaps may be named by the environment variable II_REPIMAGE_DIR or the report parameter PM_IMGDIR.

Notes:

- If an image directory is specified by either method, ALL "disk-resident" images are assumed to be located in the named directory.
- If PM_ImgDir is set it will override any setting of II_REPIMAGE_DIR.

- If both PM_ImgDir nor II_REPIMAGE_DIR are not set the original paths will be used for disk-based images.
- To force use of original paths even when an environment variable is set use:

  ```
  PM_Dir = '<embed>'
  ```

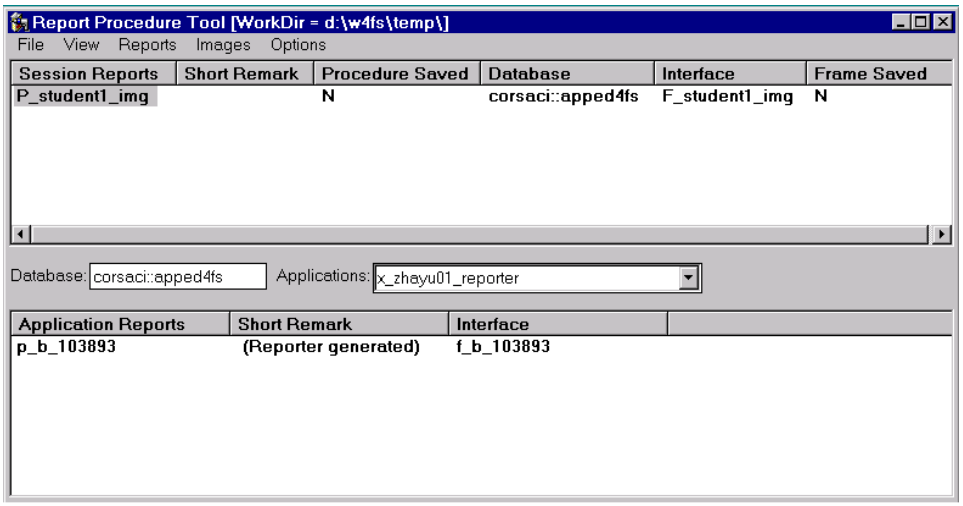
- Reporter File/Print always uses the original paths.

Report Procedure Tool

This section contains a detailed description of the Report Procedure Tool.

The Report Procedure Tool display consists of the menu bar and the following three regions:

- Dynamic Procedure List
- Database/Application Information
- Report List



Dynamic Procedure List

This view-only list displays the dynamic procedures created during the current Reporter session. The fields are:

Field Name	Description
Session Reports	Dynamic procedure names
Short Remark	Comment for report

Field Name	Description
Procedure Saved	Saved flag: whether the procedure has been exported
Database	Default database for report
Interface	Associated print dialog frame
Frame Saved	Saved flag: whether frame has been exported

Database/Application Information

The database/application information area has the following fields:

Field Name	Description
Database	Entry field allows the naming of any accessible database to which a connection may be established
Applications	Option Field listing OpenROAD applications within selected database

Report List

There is no way to distinguish Reporter procedures from other procedures. To facilitate the identification of Reporter-generated procedures contained in OpenROAD applications, the first part of the short remark is an identifier tag "<...>" that is used by the Procedure Tool.

Note: If you edit Reporter-generated procedures or frames be sure not to change the identifier tag if you wish the Procedure Tool to recognize the Reporter-generated objects. Failure to recognize a component prevents its being displayed in the Application Reports List, but does not otherwise interfere with functionality.

The fields in this view-only list are:

Field Name	Description
Application Reports	A list of reports in the currently selected application
Short Remark	With tag removed
Interface	Associated print dialog frame

Menu Bar

File

The following items are found on the File menu:

Menu Item	Description
Save Session Report	Export selected dynamic procedure and specified associated components (Reports/Options/Include) to a file. This operation always writes out the export file(s), even if not changed.
Save Image Server	Export a new or updated Image Server Frame to a file.
Import Dyn Proc	Temporarily load a report procedure from an export file.
Exit	Exits Procedure Tool. Will prompt for specified unsaved components.

View

Note: All menu items marked with an asterisk (*) are definable as user preferences.

The following items are found on the View menu:

Menu Item	Description
Application Info*	Show Database/Application and Application Report areas
Refresh Applications	Reload application list for selected database
Refresh Reports	Reload report list for selected application
Image Server	Display the current Image Server Frame
Batch File	Display the current application report import script for viewing only

Reports

Note: All menu items marked with an asterisk (*) are definable as user preferences.

The following items are found on the Reports menu:

Menu Item	Description
Run	Runs the currently selected dynamic procedure using the Procedure Tool's settings for locating images.
Convert RW Reports	Runs the Report-Writer Conversion Tool RWConv to convert Report-Writer reports into 4GL procedures that can be run using the Reporter API.
Install To Application/ Write Install Script	Import selected report components to selected application or create a batch file for execution at a later time. These two items are alternate views of the same menu button and are controlled by Reports/Options/Immediate. The user is prompted to save if the component has not been exported during the current session.
Batch Import	Execute an existing application report import script.

Menu Item	Description
Options ->	
Replace Existing*	The default is FALSE. If TRUE use '–m –nreplace' when importing procedures to target application and do not prompt if clashes detected. If FALSE and a clash is detected you will have the option to overwrite. (Clashes cannot always be detected.)
Immediate*	The default is FALSE. When TRUE exported reports are imported into target application immediately by 'w4gldev backupapp in ...' When FALSE a report import script is created and can be run from *Reports/Batch Import*, or at any time from the Windows Run command or from a CMD window.
Add Selected/Add All*	Select which dynamic procedures to install to target application.
Include ->	Applies to both export and import of selected dynamic procedure
Procedure*	(Default TRUE) - dynamic procedure
Dialog Frame*	(Default FALSE) - associated print dialog frame
Image Server*	(Default FALSE) - current Image Server
Trace ->*	Standard OpenROAD trace flags to be used on import of procedures to applications

Images

Note: All menu items marked with an asterisk (*) are definable as user preferences.

The following items are found on the Images menu:

Menu Item	Description
Image Server ->	
Load	Import an image server frame from disk and initialize the Image Server global variable
Create New	Create a new Image Server Frame from a built-in template
Add Images	Add images to the dynamic image server frame from the currently available dynamic procedures
Add Image Options ->	
Replace Existing Image*	If an image of the same id already exists in the image server, replace it. (Not yet implemented.)
Add Selected/ Add All*	Add images for the currently selected procedure or for all listed dynamic procedures
Options ->	
Check for Images*	During install of reports to target applications
Use Image Server*	(Default TRUE) - Set to FALSE to run dynamic procedure with database-resident images
Use Image Dir*	(Default TRUE) - Set to FALSE to run dynamic procedures with original paths for images

Options

Note: All menu items marked with an asterisk (*) are definable as user preferences.

The following items are found on the Options menu:

Menu Item	Description
Auto ->	
Refresh Apps*	Reload application list when database is changed
Refresh Reports*	Reload report list when application is changed
Load ImgSvr*	Load image server on startup of frame
Files ->	
Replace Existing*	Overwrite existing files without prompting
Use Default Names*	Derive names from component names (recommended)
Batch File*	Set/change report import script name
Image Directory*	Set/change bitmap image directory
Image Server*	Set/change Image Server Frame name
Working Directory*	Set/change directory for report procedures –shown in title bar

Tutorial

This is a brief tutorial to familiarize you with the new features of the Report Procedure Tool.

Before You Begin

1. Start Reporter and load a report from the database or import one from a file.
2. To test most of the new features you should use a report with at least two different image trims and have at least one of each kind: database-resident and disk-resident.
3. Copy disk-resident images to a directory that is different from the one defined in the report. All runtime disk-based images should reside in the same directory.

4. Compile or Print the report in Reporter. Once run or compiled the report document may be closed.
5. From the Tools menu, Select the menu item Procedure List.

Using the Procedure Tool

When the Procedure Tool starts, depending upon the settings, it may:

- load the application list for the current database
- load the report list for the current application
- load the default Image Server Frame from disk

If the Procedure Tool is run more than once in the same session the global Image Server may already be initialized. If auto load is set for the Image Server you will be given a chance to retain the in-cache version of the Frame and not re-load from disk.

When the Procedure Tool Frame appears, all Report procedures created in the current session by successfully compiling or printing reports will be listed as *Session Reports.* The reports do not have to be open. The title of the frame will include the working directory.

File Names

When report components are saved to disk the use of files is controlled by two options (*Options/Files*):

- **Replace Existing:** If TRUE, overwrite without prompting if file already exists, otherwise prompt for confirmation.
- **Use Default Names**: If TRUE, derive names from component names, otherwise display a filepopup.

Saving Report Procedures

Any report procedures or dialogue frames which have already been exported during this session will have "Y" in the **Procedure Saved** and/or **Frame Saved** column. To Export a report or its associated components (frame and image server) choose *File/Save Session Report.* The current procedure and/or its components (see *Reports/Options/Include*) will be exported to files as described above.

Running Report Procedures

A dynamic procedure may be run directly from here. The default mode for running the dynamic procedure is "runtime," using an image server and/or directory if image-trim support is required. To run a procedure as if from Print in the main window, set *Images/Options/Use Image Server* and *Images/Options/Use Image Directory* to FALSE.

To re-run a report that you are designing, if the report has not changed since the last run, or you wish to run the previously printed version before regenerating a new version from a changed design, it is quicker to use *Reports/Run* than the Main Window's *File/Print.*

Importing Reports to Target Applications

Note: In order for a target application to use report procedures, it must include repcomp.img and repopen.img

1. Select the target database, if necessary, by entering a valid 'node_name::database_name' into the *Database* entry field.
2. If the *Application* list is not loaded or is out of date, click *View/Refresh Applications*. If the Applications option field displays "Load Applications" then selecting this option will also load the application list.
3. Select the target applications from the *Applications* option field
4. If the Application Report List is not loaded or is out of date, click *View/Refresh Reports, or if* the Application Reports List displays "Load Reports" click on this row to load reports.
5. Click *Reports/Import to Application* or *Reports/Write Install Script* - only one of these options is available at a given time and is controlled by *Reports/Options/Immediate.*

   If the selected report or any of its specified components *(Reports/Options/Include)* has not been saved during the current session you will be prompted to:

   - Continue: Use a corresponding component on disk.
   - Cancel Install: Abort attempt to import the procedure.
   - Save Now: Export the required components and continue. If some requested component cannot be saved you will be allowed to cancel the install.

   Otherwise you will be prompted to confirm the components to be imported.

If *Reports/Options/Immediate* is TRUE the exported report components will be imported into the target application, otherwise an import script will be written and can be run via *Reports/Batch Install.*

Note: Reports/Options/Replace Existing

- If TRUE will apply the '-m –nreplace' flag to the import without prompting for confirmation.
- If FALSE and name conflicts are detected you will be given the option to Skip, Cancel Install, or Replace. Name conflicts can only be detected if the current application report list is up to date.

Working With Image Trim

As described above, reports that use image trim in their definitions may require additional support when imported to target applications. If this support is not provided the report can still be run:

- *with image-trim* - if database or disk resident images are available on the target machine/database and stored as on the design machine/database.
- *without image-trim* – if original images are not available.

Image Directories

Disk-resident image trim must exist in the original directory from which it was defined or in a single directory anywhere on the local machine if the environment variable II_REPIMAGE_DIR is set to point to that directory or the parameter PM_IMGDir is passed to the report or its dialogue frame. For a given report ALL disk-resident image trim must be stored in the same way.

In the procedure tool a report can be run in either mode determined by the setting of *Images/Options/Use Image Dir.* The image directory can be set by *Options/Files/Image Directory* and this will override II_REPIMAGE_DIR if that is also set.

Image Servers

Image trim that was database-resident at design time may exist in the Reporter images catalogues that MUST be a copy of the design database Reporter images catalogues, or the image bitmaps may be copied into an Image Server Frame.

An Image Server is an OpenROAD frame managed by a Reporter-defined userclass. The Image Server Frame becomes part of the target application for report procedures and may be constructed by the **Procedure Tool**. An appropriate Imager Server frame must be *created* or *loaded from disk.* Options/Files/Image Server may set a default name and it is expected to reside in the working directory.

Images/Image Server/Load – load an image server frame from disk and initialize the Global Image Server object

Images/Image Server/Create – build a new image server frame

Note: In either case if an Image Server is already initialized you will be prompted to keep or replace.

Images/Image Server/Add Images: - determine which database-resident images are required by the specified report(s) and add them to the image server frame. Each image has a unique ID that corresponds to the ID in the design database. Multiple documents can share the same image.

Note: A Single Image Server Frame can support multiple reports but all must come from the same design database.

Images/Image Server/Add Image Options:

- **Replace Existing**: (currently, always FALSE) – attempts to add an image trim that is already present will be ignored.

  Note: Each report can add an *association* for an existing image to the server frame – these associations are held in a viewable tablefield (*View/Image Server*)

- **Add Selected/Add All:** add images from current procedure or all procedures listed as Session Reports

File/Save Image Server: Export the image server to disk. Only available if server has been modified

Call Interface to Dialog Frames and Reporter Procedures

Calling via Dialog Frame

```
callframe <Dialogname> (PM_RunMode = runmode,
            PM_PopErr = smallint not null,
            PM_DirName = dirname,
            PM_Img_Db = imgdb,
            PM_ImgDir = imgdir,
            PM_database = datadb);
```

Optional Parameters

PM_RunMode: default CS_RM_IS (Image Server) – only meaningful if report uses image trim

PM_PopErr: (default FALSE) – when true errors are displayed in popup windows

PM_DirName: directory for temporary output files. The dialog will try to pick a reasonable directory if none given

PM_Database: (default = curframe.dbsession.database (database name)) - required only if data resides in a database other than the current dbsession.

PM_Img_Db: required only if:

- report uses image trim and
- PM_RunMode = CS_RM_DI (database images) and
- images are stored in database other than PM_Database (when PM_database is supplied) or images are not stored in current database

PM_ImgDir: required only if report uses disk-resident images and images are NOT located in original paths

Calling A Report Procedure

```
callproc <procname>(PM_Outputdir = tmpdir,
PM_OutputFile = left(repname,8),
PM_runmode = runmode,
PM_PopErr = poperr,
PM_Imgdir = imgdir );
```

Required Parameters

PM_Outputdir - directory for temporary output files

PM_OutputFile – temporary file name

Optional Parameters

PM_Runmode: default CS_RM_IS

This is only meaningful if report uses image trim that is not stored locally on disk valid runtime values:

- CS_RM_IS – use image server
- CS_RM_DI – use database-resident image trim

PM_PopErr: default FALSE

When TRUE, most runtime errors are displayed as a popup

PM_ImgDir : required only if report uses disk-resident images and images are not located in original paths

valid runtime values:

'dirname' - a valid directory containing all disk-based image trim

'' - use environment variable II_REPIMAGE_DIR, if set, otherwise use original paths

'<embed>' – use original paths – ignore any setting of II_REPIMAGE_DIR

Chapter 6

Upgrading from OpenROAD 3.5 to 4.1

Upgrading an OpenROAD application from release 3.5 to 4.1 should be a rather straightforward process as long as a few rules are adhered to. The purpose of this chapter is to describe these rules and in what context they apply.

Rule 1

OpenROAD 3.5 will not be able to access any application object that has been saved and/or created by OpenROAD 4.1. OpenROAD 4.1 cannot run OpenROAD 3.5 image files.

This rule means that to run an OpenROAD 3.5 application under OpenROAD 4.1, it needs to be converted to OpenROAD 4.1. Once an application has been converted to OpenROAD 4.1, it can no longer be accessed, developed, or run from within OpenROAD 3.5. The application developer who is responsible for converting an OpenROAD 3.5 application to 4.1 should export the OpenROAD 3.5 application from within an OpenROAD 3.5 environment as a form of backing up the application prior to OpenROAD 4.1 conversion.

Rule 2

Within a userclass hierarchy, all inherited attributes and methods must have identical properties to those of the parent superclass. This rule means that for a userclass derived from another userclass, each of the derived userclass's inherited attributes and methods must have the same datatype, nullability, and encapsulation as the derived userclass's superclass. From within the OpenROAD 3.5 application editor, the application developer will need to verify that this rule is satisfied BEFORE the application is ever accessed by the OpenROAD 4.1 workbench. The OpenROAD 4.1 workbench disallows completely the ability to change properties of inherited attributes and methods.

Rule 3

Prior to accessing any component of an application created by OpenROAD 3.5 with OpenROAD 4.1, the 3.5 application needs to be completely recompiled by OpenROAD 4.1. OpenROAD 4.1 can access a 3.5 application that already exists in a database. One may also use OpenROAD 3.5 to export the application, and use OpenROAD 4.1 to import the application into another database. At this point all applications and included applications need to be completely recompiled by the OpenROAD 4.1 compiler.

Chapter

7

Virtual Key Values

Virtual Key Values

The KeyDown and ChildKeyDown events return values that correspond to the following table of values. Each virtual key that can be returned is listed here along with its corresponding HEX and decimal value.

Virtual Key			Virtual Key			Virtual Key		
VK_BACK	0x08	8	VK_TAB	x09	9	VK_RETURN	0x0D	13
VK_SHIFT	0x10	16	VK_CONTROL	0x11	17	VK_PAUSE	0x13	19
VK_CAPITAL	0x14	20	VK_ESCAPE	0x1B	27	VK_SPACE	0x20	32
VK_PRIOR	0x21	33	VK_NEXT	0x22	34	VK_END	0x23	35
VK_HOME	0x24	36	VK_LEFT	0x25	37	VK_UP	0x26	38
VK_RIGHT	0x27	39	VK_DOWN	0x28	40	VK_SELECT	0x29	41
VK_PRINT	0x2A	42	VK_INSERT	0x2D	45	VK_DELETE	0x2E	46
VK_HELP	0x2F	47	VK_0	0x30	48	VK_1	0x31	49
VK_2	0x32	50	VK_3	0x33	51	VK_4	0x34	52
VK_5	0x35	53	VK_6	0x36	54	VK_7	0x37	55
VK_8	0x38	56	VK_9	0x39	57	VK_A	0x41	65
VK_B	0x42	66	VK_C	0x43	67	VK_D	0x44	68
VK_E	0x45	69	VK_F	0x46	70	VK_G	0x47	71
VK_H	0x48	72	VK_I	0x49	73	VK_J	0x4A	74
VK_K	0x4B	75	VK_L	0x4C	76	VK_M	0x4D	77
VK_N	0x4E	78	VK_O	0x4F	79	VK_P	0x50	80
VK_Q	0x51	81	VK_R	0x52	82	VK_S	0x53	83

Virtual Key			Virtual Key			Virtual Key		
VK_T	0x54	84	VK_U	0x55	85	VK_V	0x56	86
VK_W	0x57	87	VK_X	0x58	88	VK_Y	0x59	89
VK_Z	0x5A	90	VK_NUMPAD0	0x60	96	VK_NUMPAD1	0x61	97
VK_NUMPAD2	0x62	98	VK_NUMPAD3	0x63	99	VK_NUMPAD4	0x64	100
VK_NUMPAD5	0x65	101	VK_NUMPAD6	0x66	102	VK_NUMPAD7	0x67	103
VK_NUMPAD8	0x68	104	VK_NUMPAD9	0x69	105	VK_MULTIPLY	0x6A	106
VK_ADD	0x6B	107	VK_SEPARATOR	0x6C	108	VK_SUBTRACT	0x6D	109
VK_DECIMAL	0x6E	110	VK_DIVIDE	0x6F	111	VK_F1	0x70	112
VK_F2	0x71	113	VK_F3	0x72	114	VK_F4	0x73	115
VK_F5	0x74	116	VK_F6	0x75	117	VK_F7	0x76	118
VK_F8	0x77	119	VK_F9	0x78	120	VK_F10	0x79	121
VK_F11	0x7A	122	VK_F12	0x7B	123	VK_F13	0x7C	124
VK_F14	0x7D	125	VK_F15	0x7E	126	VK_F16	0x7F	127
VK_F17	0x80	128	VK_F18	0x81	129	VK_F19	0x82	130
VK_F20	0x83	131	VK_F21	0x84	132	VK_F22	0x85	133
VK_F23	0x86	134	VK_F24	0x87	135	VK_NUMLOCK	0x90	144
VK_SCROLL	0x91	145	VK_LSHIFT	0xA0	160	VK_RSHIFT	0xA1	161
VK_LCONTROL	0xA2	162	VK_RCONTROL	0xA3	163	VK_LMENU	0xA4	164
VK_RMENU	0xA5	165						