Contents

Omnis Studio External Components

Creating your own External Components

Omnis Software Ltd

May 2023

About This Manual

This manual describes how you can create your own external components to integrate into Omnis Studio. You can download sample source code from the Omnis website to help you do this.

For more information about Omnis external components, and to download the latest source files, please go to:

https://www.omnis.net/developers/resources/download/tools/buildyourown.jsp

This manual introduces key development topics and expands to form a reference guide for each of the main APIs provided by the Omnis component library.

Upgrading to the Latest Version

If you are upgrading to the latest version of Omnis Studio (e.g. version 11), and you have created your own external components for a previous version, then these components will need to be recompiled for the new version of Omnis Studio using the latest external component source files, which can be downloaded using the link above.

Copyright info

Disclaimer

This document is provided as a free resource to developers wishing to write custom components for use with Omnis Studio. We hope you find the enclosed material useful and we have endeavoured to ensure its accuracy at time of print. Your attention is drawn to the terms of your Omnis Developer Partner Program contract (ODPP) which excludes development support for customised Omnis external components. This material supplements other external component documentation published by Omnis Software and is provided as a self-help resource.

Use and development of the supplied source code and associated utilities is carried out at the discretion of the external developer and the developer assumes responsibility for any consequential damage and/or loss of data which may result.

IN NO EVENT WILL OMNIS SOFTWARE BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF OMNIS SOFTWARE HAS BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Chapter 1—Omnis External Components

Introduction

Omnis external components are plug-in modules that extend the range of visual and non-visual objects available in the design and runtime environments in Omnis, as well extending the Omnis programming language. There are many different external components supplied with Omnis, but you can create your own using your own software development tools and the information in this manual.

Once built and installed into Omnis, external components behave in exactly the same way as standard built-in Omnis components. You can change the properties of an external component in design mode using the Property Manager. Likewise, at runtime you can manipulate an external component using methods and the notation, and examine its runtime properties in the Notation Inspector.

External components can also contain functions or methods and events, which you can call or intercept using Omnis methods. You can build all of these features into your own external components.

The type and range of external components include:

- **Window objects (including background objects) and Report objects**
  Both window and report external components appear in their respective group in the Omnis Component Store and can be used in your libraries in exactly the same way as built-in GUI objects.
- **Static Functions**
  Static functions are components that contain functions, that appear in the Omnis Catalog under the 'Functions' group. These functions can be invoked from calculations in your Omnis code.
- **Omnis objects**
  Omnis objects or so-called 'non-visual' components are objects that can contain methods and properties, which can be used in the Omnis language or called to perform some specific function. External objects can be sub-classed, just like normal Omnis objects, to form new objects. The SQL DAMs are examples of non-visual components.

This manual describes how you can build external components, for adding to Window and Report classes, or for using in your code, but if you are creating components for the *JavaScript Client* (to add to Remote forms) you need to consult the following manual: JavaScript Component SDK

Creating your own External Components

Using the libraries supplied, you can create Omnis external components that run under all platforms supported in Omnis. All of the samples supplied have independent source code. The Omnis resource compilers for Linux and macOS (Xcode) are supplied. These compile simple Windows style .RC files, and support image types .BMP, allowing the entire component to be portable.

Components in Omnis

When you start Omnis, you have to tell it to load your new component. You can load an external component via the #EXTCOMPLIBS system table. You can access this via the Studio Browser, or open a Window class in design mode and right-click on the Component Store, and select the **External Components** option.

If the components you create are OK, they should appear in the #EXTCOMPLIBS system table. If you cannot find your component in the external component list, check the Omnis Trace Log window. Omnis will always write any errors to the trace log during startup.

In the External Component dialog, use one of the radio button options to load the component, either 'Starting Omnis' or 'Opening <libname>'. When you return to the design window, your component will be added to the relevant group in the Component Store (as defined in the component), e.g. the Other group (or you can use the Search box to find the component). You should be able to drag the control on to a window class and your component is created.

Loading Components

On all platforms, external components are loaded from the relevant sub-directory (xcomps, jscomps and logcomps folders) of both the Omnis data folder and the Omnis program / Application folder.

If there is a component (.dll, .u_... or .so, depending on platform) with the same case-sensitive name in both relevant sub-directories of the data and program folders, the component in the data folder is loaded.

Upgrading to the Latest Version

If you are upgrading to the latest version of Omnis Studio (e.g. version 11), and you have created your own external components for a previous version, then these components will need to be recompiled for the new version of Omnis Studio using the latest external component source files, which can be downloaded here:

https://www.omnis.net/developers/resources/download/tools/buildyourown.jsp

Windows and Child Windows

Omnis supports two window types. Top level windows and child windows. In the Omnis IDE, you can create top level windows as window classes, and design the contents of the window by adding controls such as buttons and lists. All window controls such as button and list controls are child windows. A child window is a window that sits inside another parent window. Child windows can also contain other window controls, thus the parent-child relationship can be nested at several levels. For example, a scrollbox window field is a child control within the window class, but it can have other child controls placed within it, thus making it a parent.

An external component operates inside a child window and performs some kind of operation within the child window. The component can do virtually anything from draw a graph, scroll a message, or pick up a click within it and send a message back to Omnis. To do this, the window receives and processes messages. A message informs the child window of all events that affect it, such as the user clicking on it with the mouse. Later, when you create a component, you need to tell Omnis the name of a procedure that Omnis can call with your message. This procedure is often referred to as the WNDPROC (short for Window Procedure) or message handler. There are many messages defined by the component library that your procedure is sent, some you will want to deal with, others you can ignore; you will see how to deal with these messages.

Data types Defined by the Component Library

To help you write platform-independent components, you should use the data types declared by the component library. All APIs in the library use the following data types.

C-type	Omnis type	Description
unsigned char orunsigned long*	qchar	standard unsigned char value*qchar is defined as 4 bytes for Unicode ta
char orunsigned short	qoschar	platform API-dependent Unicode character. 2 bytes for Win32 & Mac OS Linux targets & non-Unicode targets.
unsigned char	qbyte	assumed to hold 0-255
unsigned char	qbool	assumed to hold qtrue or qfalse
short	qshort	standard short value
unsigned short	qushort	standard unsigned short value
long	qlong	standard long value
unsigned long	qulong	standard unsigned long value
platform dependent	qreal	used for real arithmetic
short	qret	return type from some API calls
enum	qnil	can be used to assign to some objects to clear them
unsigned char	qint1	1 byte unsigned integer (as stored on disk)
short	qint2	2 byte integer (as stored on disk)
unsigned short	qword2	2 byte unsigned integer (as stored on disk)
long	qint4	4 byte integer (as stored on disk)
unsigned long	qword4	4 byte unsigned integer (as stored on disk)
long	rstrno	uses when calling RESxxx functions
short	attnum	property numbers
qbool	qfalse = 0	false boolean value
qbool	qtrue = 1	true boolean value
qret	e_ok = 0	no error occurred
qret	e_negative = 1	error occurred

As well as using the data types, you should try to use the component API as much as possible to ensure platform independent code. In the long run, it may mean you have to recompile for another platform, rather than having to port lots of code.

Types of visual components

Omnis supports different types of external component which you can add to window and report classes. When Omnis starts up, the component specifies what kind of component it is, and what class type it should appear in.

- **cObjType_Basic**
  a generic window class component.

- **cObjType_Picture**
  a derived Omnis picture component for window classes.
- **cObjType_List**
  a derived Omnis list component for window classes.
- **cObjType_DropList**
  a derived Omnis droplist component for window classes.
- **cObjType_IconArray**
  a derived Omnis icon array component for window classes.
- **cObjType_PriOutput**
  a custom report output device
- **cRepObjType_Basic**
  a generic report class component.
- **cRepObjType_Picture**
  a derived Omnis picture component for report classes.

Components can be both window and report objects. For example, you may want to create a picture-handling component, that works in both window and report mode, therefore its returns type should be:

```
cObjType_Picture | cRepObjType_Picture
```

Basic Components

Basic components are generic controls that receive all messages via the WNDPROC. You must code all actions that you want to happen inside your control.

See the examples provided.

Picture Components

Picture components are objects derived from the internal Omnis picture field. Omnis calls your WNDPROC with standard messages, but you also receive some specific messages only for derived picture controls. For example, Omnis calls you to inquire how big your image is, so it can handle the scrolling and call you to paint.

See the PCX example.

List Components

List components are objects derived from the internal Omnis list field. Omnis calls your WNDPROC with standard messages, but you also receive some specific messages only for derived list controls. For example, when Omnis paints your list, you are called to draw individual lines, possibly in a selected state.

See the PICLIST example.

Droplist Components

Droplist components are objects derived from the internal Omnis droplist field. Omnis calls your WNDPROC with standard messages, but you also receive some specific messages only for derived droplist controls. For example, when Omnis paints your droplist contents, you will be called to draw individual lines.

Icon Array Components

Icon array components are objects derived from the internal Omnis icon array field. Omnis calls your WNDPROC with standard messages, but you also receive some specific messages only for derived icon array controls. For example, when Omnis paints your icon array, you will be called to draw individual icons and icon labels.

See the ICNARRAY example.

Report Components

Report components should be treated as if they were window components. When printing is required, you are called with specific report printing messages.

See the PCX example.

Background Components

Background components are objects that behave like internal Omnis background objects. For example, background objects never have the focus or receive events. They are always drawn as part of the background of the window they belong to. One of the sample background components supplied is an object that allows a bitmap to be tiled over an area.

Background components do not have their own **cObjType_XXX** type, and need to be defined as a **cObjType_Basic** type component. A flag needs to be set on ECM_CONNECT to indicate the component should be treated as a background component.

However, it is important to note that you cannot have both background and other visible components in the same library.

See the TILE or WASH example.

Types of non-visual components

Omnis supports various types of non-visual components. In this context, 'non-visual' means a component that does not have a visual interface but one that provides some functionality, such as functions or methods, that can be used in the Omnis programming language. Most non-visual components do not need to be placed on a window or report for their functions or methods to be called. Non-visual and visual components may co-exist in the same library.

Picture Format Conversion

Picture format conversion are libraries which provide functionality to convert from the specified format to a native O/S picture and visa-versa.

See PCX example.

Static Functions

Static functions behave just like Omnis functions and appear in the catalog just like in-built Omnis functions.

See the FILEOPS example.

Object Components

Object components appear in Omnis as objects and can therefore be utilized by adding an 'Object' variable (with the appropriate sub-type).

Just like Omnis object classes, external object components may be sub-classed to form new objects.

See the FILEOPS example.

DAMs

Writing custom Data Access Modules for Omnis Studio is discussed in the "Omnis Studio DAM API" chapter. This chapter discusses the additional damlib library needed to build these specialised non-visual components as well as datatypes, structures, classes and general techniques involved.

See the GENERIC DAM example.

Writing Thread-Safe Components

Where several instances of your component may be in use at the same time, you will need to design your code with thread isolation in mind. Use of object-oriented techniques provides the basis for thread-safety as this gives each object instance its own memory and variables.

Where shared memory or commonality exists between multiple instances, the C/C++ programming language facilitates thread management, semaphores and mutual-exclusive execution (MUTEXs) which can be used to control access to the shared resources. Any such commonality should be identified at the design stage.

You can also enhance the thread-safety of your component by passing the EXT_FLAG_SESSION flag to Omnis when processing the ECM_CONNECT message. (See Structures, Messages and Functions for more details).

Component SDK Requirements

- **Microsoft Windows**
  Microsoft Visual Studio 2019 (v142 build tools).

- **macOS**
  Xcode 14.x (requires macOS Monterey 12.5+ or macOS Ventura).
- **Linux**
  CMake 3.15.5+.
  GNU C and C++ standard libraries (from GCC version 7 or later).
  Clang 8.0.0+.

Component SDK Source Files

There is one SDK source tree for every supported platform. Each source tree contains example external components, ranging from the very simple, GENERIC (lets you create a basic shell component), to the more complex controls, such as CALENDAR. The source code for the components is generally 100% cross platform and has been duplicated in the various source trees. A few components have some code which is not shared by all platforms. Such platform dependent source will usually be found in source files with names that start with an 'x'. The source trees have scripts or projects that can be used to build the components.

When creating external components, try to keep to the tree structure, that is, if you want to create a new component called 'Simple', create a directory called Simple inside the source tree. Keeping to the structure will help when porting to other platforms.

Getting Started with Generic

One of the many samples supplied to help you create Omnis external components is **generic**. There are four versions of this control in the source tree that explain how to write Omnis components and give you a useful starting framework for building your own components. Before you begin to write some code, you need to setup your development environment.

Component library

Inside the SDK tree the Studio component library and header files used for building a component are located in the following folders.

Windows:

libs - contains Debug and Release versions of the component library (complib.lib) for linking against the multi-threaded DLL C runtime library (dllrt) or the statically linked multi-threaded C runtime library (mt).

The Studio xcomp projects use the multi-threaded DLL C runtime and locate a copy of the DLLRT complib in either the Debug_x86/x64 or Release_x86/x64 folder.

include - contains the component library header files to be used with the component library.

Use either the x86/x64 version of the Windows SDK when targeting 32/64 bit versions of Windows.

macOS:

complib/complib_debug - contain the Release/Debug component library framework (complib.framework).

Linux:

complib - contains the Release version of the component library (libhomnis.a) and the component library headers.

complib_debug - contains the Debug version of the component library.

Building the Generic Component

Windows 32 bit:

Select the project file at:

`\Windows-SDK-11-x86\xcomp\generic\windows\generic.vcxproj`

and open with Visual Studio.

Choose either the Debug or Release Solution Configuration and build that solution.

The DLL generated will be placed in one of:

`Windows-SDK-11-x86\build\Release_x86\xcomp\generic.dll`

`Windows-SDK-11-x86\build\Debug_x86\xcomp\generic.dll`

Windows 64 bit:

Select the project file at:

`\Windows-SDK-11-x64\xcomp\generic\windows\generic.vcxproj`

and open with Visual Studio.

Choose either the Debug or Release Solution Configuration and build that solution.

The DLL generated will be placed in one of:

`Windows-SDK-11-x64\build\Release_x64\xcomp\generic.dll`

`Windows-SDK-11-x64\build\Debug_x64\xcomp\generic.dll`

macOS:

Building Studio resources (.RC files) for a component requires that the Studio resource compiler is added to the Xcode tree.

Copy tools/omnisrc64.app into:

`/Applications/Xcode.app/Contents/Developer/Tools`

Select the project file at:

`/macOS_SDK_11/xcomp/generic/mac/generic.xcodeproj`

and open with Xcode.

For a Deployment/Release build choose Product->Build For->Profiling (Shift-Cmnd-I) and for a Development/Debug build choose Product->Build For->Testing (Shift-Cmnd-U).

The resulting xcomp bundle will be placed at:

`/OmnisSDKBuild/Release/xcomp/generic.u\_xcomp` and `/OmnisSDKBuild/Debug/xcomp/generic.u\_xcomp` (relative to the root of the SDK location).

On macOS, a Release build will generate a Universal binary which targets both Intel and M series Silicon architectures and a Debug build will target only the platform it was built on.

Therefore, when creating a new component ensure that if linking with third party libraries those libraries are provided as Universal versions.

Linux:

Uses the CMake tool to automate the building of components. More information can be found at https://cmake.org.

All components (including generic.so) are built from the terminal line using the build_all.sh bash shell script at:

`/linux-headless-sdk-11/master/linux/build\_all/build\_all.sh`

Usage:

```
build_all.sh [-o|--outdir <dir>] [-d|--debug]
-o | --outdir <dir>: The root output directory for the build.
-d | --debug: Debug
--clean: Deletes the output directory before building.
```

To build a set of Release components change the current path to the folder containing the script and use:

```
./build_all.sh
```

This will place the resulting libraries at:

`/linux-headless-sdk-11/output/Release/headless/Server/xcomp/`

The generic component will build into:

`/linux-headless-sdk-11/output/Release/headless/Server/xcomp/generic.so`

Adding the -d switch will produce a set of debug libraries at:

`/linux-headless-sdk-11/output/Debug/headless/Server/xcomp/`

The default build and output path are created at the root of the SDK tree. To change this, specify a different path using the -o switch.

Testing the Generic Component

To use the component, place a copy in the XCOMP folder of the Studio tree.

To add the component to the Component Store right click the Project Libraries node in the Studio Browser and select Show Comps.lbs.

The Component Library node will be shown. Select External Components from the class pane.

This will show the External components window from which a component in the tree can be selected.

Expand the External Components node and select the Generic Library.

From the Pre-Load Status options select Starting Omnis. This will load the component for use from the Component Store when Studio is started.

Now when the Component Store is used it will show the Generic Control in the External Components group and this can be dragged and dropped onto a window.

Any errors raised when loading the component will be reported in the Omnis trace log.

Creating a new Component

The generic example can be used as a template for creating a new component.

New components should use the same tree structure and project settings as generic with project source files added to the source folder.

This will ensure that the component SDK headers and libraries can be located and that a build is output to the expected location.

On Windows and macOS ensure the Visual Studio and Xcode project settings are based on the settings used for a sample component to ensure they build correctly.

On Linux to use a new component (myxcomp) with the CMake build tool modify the following files (assuming this used generic as a template).

`/linux-headless-sdk-11/xcomp/myxcomp/linux/myxcomp/CMakeLists.txt`

Alter the name of the project on line 2 from generic to myxcomp and update the set(RCFILE) and set(SOURCES) commands on the lines which follow to use the correct resource file and source files.

`/linux-headless-sdk-11/xcomp/linux/all_xcomps/CMakeLists.txt`

Add a line to the end of the file which will call the build macro for the new component.

```
add_xcomp_project("myxcomp")
```

Components which are not required can have their add_xcomp_project entry removed or commented out from this file.

Use the build_all.sh script to build the component via CMake.

Debug Symbols

By default, Omnis makes use of the Google Crashpad crash-reporting system to log the state of the program when a crash occurs.

To debug crashes in a component requires that Release versions of components contain debug symbols.

Therefore, the Release versions of the SDK sample components will use debug symbols and generate an associated symbol file.

The debug symbol file for the component binary can then be used with a Crashpad mini dump (.dmp) to provide information about a crash, e.g. stack trace and variable state.

Windows:

A Release build uses a PDB file as its symbol file and places this in:

`\Windows-SDK-11-x64\intbuild\debugsymbols_x64\xcomp\<component_name>.pdb`

Studio Crashpad mini dump files (.dmp) are placed in the Studio log at:

`AppData\Local\Omnis Software\OS 11\logs\crashes\reports\reports\`

The following provides an overview of how to use Visual Studio to open the dmp file, load the DLL and its PDB file to provide information about a crash.

https://learn.microsoft.com/en-us/visualstudio/debugger/using-dump-files?view=vs-2019

macOS:

A Deployment/Release build uses a dSYM file as its symbol file and places this in:

`/OmnisSDKBuild/DebugSymbols/xcomp/<component\_name>.u\_xcomp.dSYM\` (relative to the root of the SDK location.)

Studio Crashpad mini dump files (.dmp) are placed in the Studio log at:

`/Application\ Support/Omnis/Omnis\ Studio\ 11/logs/crashes/reports/pending/`

Linux:

A Release build uses a .debug file as its symbol file and places this in,

`/output/debugsymbols/headless/Release/xcomp/<component\_name>.so.debug` (relative to the output location specified when building).

Studio Crashpad mini dump files (.dmp) are placed in the Studio log at,

`/logs/crashes/reports/pending/`

Debugging on macOS/Linux

On both macOS and Linux the lldb command line debugger tool should be used with the dmp file and symbol file to provide information about a crash.

https://lldb.llvm.org

Typically, lldb starts with the path to the main executable which generated the crash and the dmp file.

macOS:

```
lldb /Applications/Omnis\ Studio\ 11.app --core <path_to_dmp_file>
```

Linux:

```
lldb /usr/local/omnis_software/omnis_studio_headless_app_server_11/homnis --core <path_to_dmp_file>
```

The external library module can be added to the debugger by using the target modules command with the location of the library binary and its symbol file.

https://lldb.llvm.org/use/symbolication.html#

For example:

macOS:

```
target modules add /Applications/Omnis\ Studio\ 11.app/Contents/MacOS/xcomp/myxcomp.u_xcomp/Contents/MacOS/myx
```

Linux:

```
target modules add /usr/local/omnis_software/omnis_studio_headless_app_server_11/xcomp/myxcomp.so '/home/user/
```

Generic source code in more depth

The generic sources provide a general template for component code.

Generic.cpp

The main body of the component implementation is described here in more detail.

The includes:

```
#include <extcomp.he>
#include <extfval.he>
#include <hwnd.he>
#include <gdi.he>
#include "generic.he"
```

extcomp.he, extfval.he, hwnd.he *and* ***gdi.he*** are external component library header files. ***extcomp.he*** declares various external component specific APIs; ***extfval.he*** declares the data storage class; ***hwnd.he*** declares the child window API calls; ***gdi.he*** declares the graphical API calls; and ***generic.he*** defines the component class.

The component entry point and message procedure is as follows.

```
extern "C" qlong OmnisWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
```

OmnisWNDPROC is a #define that the Omnis component library has setup. This defines some calling conventions that vary from platform to platform. For now this is all you really need to know. Next is the name, GenericWndProc. This is the message procedure Omnis calls with your child window messages. This procedure has the following parameters:

HWND	hwnd	This is a handle to the child window the message is for
LPARAM	Msg	This is the message
WPARAM	wParam	This is extra information for the message
LPARAM	lParam	This is extra information for the message
EXTcompInfo*	eci	This is a pointer to some information about your component

When Omnis calls the message procedure, it sends along the HWND in the **hwnd** parameter. This is the child window the message was for. Complex components may have many child windows all using the same message procedure. Using the HWND helps the component do the right thing for the right child window.

Msg is the message. There are many messages that can be sent to your procedure, such as WM_PAINT or WM_LBUTTONDOWN.

wParam is some extra information for the message. Sometimes messages need to pass other information, such as the cursor position when the WM_LBUTTONDOWN was generated.

lParam, like wParam, is used for extra message information.

eci is a pointer to a structure holding information about your component. It is used with various API calls. See later.

Next is the first and *most important* line of the message procedure.

```
ECOsetupCallbacks(hwnd, eci);
```

Most of the API calls call Omnis for some information, or to do some processing. This line enables the call to Omnis to work. If this line is missing, your component will crash.

Next there is a switch statement testing the message parameter:

ECM_GETCOMPLIBINFO. This is the first message the message procedure handles. Omnis is calling your message procedure trying to find out how many controls your component supports, and the name of your library.

```
return ECOreturnCompInfo( gInstLib, eci, LIB_RES_NAME, OBJECT_COUNT );
```

Here you return the result of a function call *ECOreturnCompInfo*. This function is described later, but usually takes a string resource number which holds the name of your component library, and takes the number of controls your component contains.

ECM_GETCOMPID. Omnis now knows how many controls you are intending to support in your component library due to the result of the last message. It now wants to know what ID each control within the component library should have. The id can be any number you decide to associate with the control. In the future when Omnis wants something to happen to a control, it uses the id you return here. Omnis calls you with this message *for 1 to n* times, where n is the number of controls your library supports. The calling count is passed in ***wParam***.

```
if ( wParam==1 )
  return ECOreturnCompID( gInstLib, eci, OBJECT_ID1, cObjType_Basic );
return 0L;
```

Since generic supports one control (**OBJECT_COUNT=1,** this is defined in your header file), you wait for a call where wParam is 1. On this message, you return the result of ***ECOreturnCompID***. This API specifies the controls id, and the type of control you want it to be. See ***Types Of Component*** later in this document. Here you indicate the control has an id of **OBJECT_ID1** and is a **cObjType_Basic** basic component.

ECM_GETCOMPICON. Now Omnis knows how many controls your library has and the id for each control, it asks for the icon to use in the Omnis Component Store.

```
if ( eci->mCompId==OBJECT_ID1 )
  return ECOreturnIcon( gInstLib, eci, GENERIC_ICON );
return qfalse;
```

Here, you are checking a member of the **eci** parameter, **mCompId.** This is set to an id you returned from the last message (**OBJECT_ID1)**. The ***ECOreturnIcon*** API is described later, but generally it extracts a **.bmp** (bitmap) from the resource file so you can return it to Omnis.

With regards to setting up your component so that Omnis knows it is there, these messages are generally all you need. The next set of messages are used when you place your component on a window or report class. When that happens, Omnis calls your message procedure with many more messages. Here are the important ones.

ECM_OBJCONSTRUCT. Omnis is calling the message procedure as it is just about to create an instance of your object. This can happen when you drag a component out of the Component Store on to a design window or report class, or a window class is being opened in runtime mode, or a report is being printed. This is the code you need to execute:

```
tqfGenericObject* object = new tqfGenericObject( hwnd );
ECOinsertObject( eci, hwnd, (void*)object );
return qtrue;
```

The first line creates a new object called ***tqfGenericObject***, which is defined below. This class performs all of the operations for your control. Next it is calling ***ECOinsertObject***. This API adds a pointer to the ***tqfGenericObject*** just created into a chain of objects. The pointer and the ***hwnd*** being passed are stored in the chain. The external component library maintains this chain, so later when a message arrives in your message procedure, you can ask for the object (***tqfGenericObject***) based on the child window the message was for, and get the correct object to process the message. This is necessary as multiple instances of your component can be created.

Finally you return qtrue. This informs Omnis you have processed the message. Not all messages expect qtrue to indicate the message was handled. The return value from all messages can be found in this manual.

ECM_OBJDESTRUCT is the next message. Omnis is calling the message procedure as it is just about to delete an instance of your object. This can happen when you close a window class containing your component, or a report has finished printing your component:

```
tqfGenericObject* object = tqfGenericObject*)ECOremoveObject(eci,hwnd);
if ( NULL!=object )
{
  delete object;
}
return qtrue;
```

The first line here is calling ***ECOremoveObject*** to find an object in the chain of objects based on the passed **hwnd.** If an object is found, it is removed from the list and a pointer to the object is returned. If the pointer is valid, you delete it, freeing all memory previously allocated. Again, you return qtrue to inform Omnis you have processed the message.

Finally:

```
return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
```

This is another *very* important line. Remember that many messages are sent to the message procedure, some important, others not so important. This is where the not so important messages should go. ***WNDdefWindowProc*** is an API to which all messages not handled should be passed. This allows Omnis to do the default operation for messages you do not want to handle.

On previous messages, ***ECM_OBJCONSTRUCT*** and ***ECM_OBJDESTRUCT*** referred to the ***tqfGenericObject*** class. This class contains the code which makes the component actually do something. You can add to this later.

Generic.he

This defines the tqfGenericObject class declared in generic.cpp.

Generic.rc

This is the resource file. It is laid out like a Windows resource file. Under macOS, the Studio resource compiler (which is part of the SDK) must be used to support the basic set of Windows resource keywords. If you keep the resource files simple, they will be cross-platform.

The resource file first includes a bitmap (generic.bmp). You can either create a small Window **.BMP** file (16x16 preferably), or use a copy of the generic.bmp file.

String 31000 is **very important**, as this is the name of the message procedure that Omnis tries to call. If you do not have this string, the name of your message procedure should be 'OmnisEXTCOMPONENT'. If the message procedure is not called this, and you do not have a string defining the name to call, Omnis will not call you.

Windows .DEF / Generic.def

Under Windows, an export file is needed for the compiler. This file describes what functions can be called from outside the component. As described above, Omnis needs to call your message procedure with messages for your child windows. For it to do that, the message procedures must be *exported*.

Extending Generic

To extend the functionality of the first example the generic2 sample component implements adding a property to the component which will appear in the Property Manager. You can modify component properties in design mode and runtime using the Property Manager.

The definition of the tqfGenericObject in generic2.he adds a new member called **mMyColor** to the class. Its type is **qcol**. This is a type defined in **gdi.he** and represents a color value (RGB).

A new member function **attributeSupport** is also added. This function is used when Omnis is doing something with your properties.

The paint method in generic2.cpp uses some API calls from both ***hwnd.he*** and ***gdi.he.*** When this method is called as a result of a message, it fills your component with the color that is stored in the new color member **mMyColor.** You should read the *HWND Reference* and *GDI Reference* chapters for an explanation of the APIs used, but generally, the code gets the size of your child window (left, top, width, height), and gets a solid brush. It sets the color of the solid brush to the color in the new color member and then fills the child window with that color.

Consider the following messages in the GenericWndProc entry procedure in generic2.cpp:

WM_PAINT message informs use the **hwnd** needs painting.

ECM_GETPROPNAME is sent by Omnis to ask for the component's property table.

ECM_PROPERTYCANASSIGN is sent by Omnis to see if a property can have values assigned.

ECM_SETPROPERTY is sent by Omnis to get the value of a property.

ECM_GETPROPERTY is sent by Omnis to set the value of a property.

When you get a **WM_PAINT** message, you find the object in the chain of object instances from the hwnd coming into the message procedure. If you find the object, you call the ::paint() member function of the object.

When you get a **ECM_GETPROPNAME** message, you call another ECO API to build a property table and return it to Omnis. This API is described later, see *Component Properties*.

```
const cMyColorProp = 1;
ECOproperty MyProperties[] =
{
  cMyColorProp, 4000, fftInteger, EXTD_FLAG_PWINDCOL, 0, 0, 0
};
```

This table defines your properties. The layout of the table is described in the ECOproperty section of the Structures, Messages and Functions chapter, but generally it describes the property id, the resource name of the property, its data type, and the type of data as shown in the Property Manager. The property table, when returned to Omnis using the code shown below, controls how your properties are handled.

```
return ECOreturnProperties( gInstLib, eci, &MyProperties[0], 1 );
```

The attributeSupport method is called when Omnis needs to do something with your properties. The messages used in this method are covered in more detail in the ECM_PROPERTYCANASSIGN, ECM_GETPROPERTY and ECM_GETPROPERTY sections of the Structures, Messages and Functions chapter, but generally it lets you handle your color property or any future properties you decide to add. For this example, when a color property is assigned, you alter the member in the **tqfGenericObject** class with the new color

value being sent from Omnis, force your child window to be repainted, resulting in the new color being drawn on screen. When the Property Manager needs to know what the color is, you send it the value back, and you also tell the Property Manager if it is allowed to assign color to your object.

The generic2.rc resource file also has an entry for the description of the $mycolor property.

After building the generic2 component close Omnis if it is still running, and move the component into your XCOMP folder. Restart Omnis. In the Omnis IDE, when you open a window class and click on your component control, a Custom tab is displayed in the Property Manager. Select it and you should see your color property. Now try assigning some values and it should change the color of your component.

You have covered the very basics of building your own external component. The source contains further generic samples that build on from the basic one adding more properties, events, and component methods.

If you are ready for more of a challenge, the source has many other controls that demonstrate much more of the external components interface. All of the samples supplied are completely cross-platform.

Bear in mind Omnis is a cross-platform development tool, and the external component interface has been designed with this in mind. If you want your controls to run on all platforms supported in Omnis, try to use the Omnis API as much as possible. There is very little it cannot do, and if you *only* use the API, your code should remain completely portable, all you need to do is recompile.

General Hints

Here are some general points you should remember when writing Omnis components:

- Keep to the External Component API; this helps you port your controls to other platforms
- Initialize all members used to handle properties. When the control is first created, the initial values, as displayed in the Property Manager, are the values you initialize your members to.
- Read the 'Memory Issues' for the EXTfldval and EXTqlist classes later defined.
- Do NOT nest painting. See ***WNDstartDraw*** and ***WNDendDraw*** in the HWND documentation.
- For large amounts of data, such as picture components, you can use the ***MEMincAddr*** and ***MEMdecAddr*** function to handle large images.
- If you have any problems with your component when you are within the Omnis IDE, such as the DLL not appearing in the #EXTCOMP dialog, check the Omnis trace log. Any problems encountered in Omnis with respect to your components are reported to the trace log.
- String resource 31020 should not be used as it is reserved for Web Client version number checking.
- If you declare a date property (fftDate), depending on the date subtype used with the EXTfldval::setDate() API, the Property Manager uses either #FT or #FDT to format the property value.

```
EXTfldval fval;
fval.setDate( myDateValue, dpFtime );
```

This example stores a date value in an EXTfldval object. The Property Manager would use #FT to format value because the date subtype used was dpFtime.

macOS and XCode Resource Files

Please note that Xcode and its underlying build scripts may encounter problems where file or folder names contain spaces. For this reason it is best to use underscores in place of spaces. Alternatively, it may be possible to work around the issue by adding double quotes around various build attributes, for example:

- The two arguments to the *cp* command that follows the *omnisrc* command in the rule for compiling rc files.
- The header search path for the project headers.
- The framework search path.

Linux Compilation Issues

Ensure that your Linux system has the necessary GCC C/C++ libraries (version 7 or higher) installed in addition to the Clang compiler (version 8 or higher) and CMake build tool (version 3.15.5 or higher).

Check the installed version of GCC using:

```
gcc --version
```

This can be installed or updated using the package manager provided by the system, e.g. for Ubuntu:

```
sudo apt update
sudo apt install build-essential
```

To check the installed version of Clang use:

```
clang --version
```

If this needs to be installed or updated use the package manager provided by the system, e.g. for Ubuntu:

```
sudo apt install clang-8 --install-suggests
```

Check the version of CMake installed using:

```
cmake --version
```

Use the package manager provided by the system to update or install, e.g. for Ubuntu:

```
sudo apt install cmake
```

Creating Non-Visual Components

Non-visual components are component libraries which contain either Omnis static functions and/or Omnis external class objects.

Just like in high-level languages such as C++ static functions are useful when processing single non-related tasks. However when functions are related, it is sometimes useful to build a collection of related functions into a class object.

Static Functions

Static functions are functions which can be used in the Omnis script language in calculations. Component library static functions appear in the 'Functions' category in the Catalog window.

Adding static functions to your component library requires the following steps:

- Add the flag EXT_FLAG_NVOBJECTS to the set of flags returned by ECM_CONNECT message. Without adding this flag, Omnis will not request the list of static functions from your library.
- Return a list via ECOreturnMethods in response to a ECM_GETSTATICOBJECT message. This message is sent to the component library when Omnis requires a list of static functions.
- Respond to ECM_METHODCALL. However, as these are static functions, the HWND parameter will be NULL. As will wParam and lParam.
- An example of static functions in use would be (excerpts from FILEOPS):

```
ECOmethodEvent fileStaticFuncs[ cSMethod_Count ] =
{ // Unique external ID Resource Number Flags
  cSMethodId_CreateDir , 8000, 0, 0, 0, 0, 0,
  ….
  cSMethodId_Rename , 8014, 0, 0, 0, 0, 0
};
extern "C" qlong OmnisWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_CONNECT:
    { // Component library contains NV objects & should always be loaded
      return EXT_FLAG_LOADED|EXT_FLAG_ALWAYS_USABLE|EXT_FLAG_NVOBJECTS;
    }
    case ECM_GETSTATICOBJECT:
    { // Omnis is requesting a list of our static functions
      return ECOreturnMethods( gInstLib, eci, &fileStaticFuncs[0], cSMethod_Count );
    }
    case ECM_METHODCALL:
    { // Omnis requires a static method to be called
      switch ( ECOgetId(pEci) )
      {
        case cSMethodId_CreateDir:... Processing
         …
        case cSMethodId_Rename: … Processing
      }
      return 1L;
    }
  }
  return DefWindowProc( hwnd,Msg,wParam,lParam,eci);
}
```

See also ECM_CONNECT, ECM_GETSTATICOBJECT, ECM_METHODCALL, ECOreturnMethods

Class Objects

Class objects are, as in the Omnis language, objects which group together data and functions into a single entity.

Adding class objects requires the following steps :-

- Add the flag EXT_FLAG_NVOBJECTS to the set of flags returned by ECM_CONNECT message. Without adding this flag, Omnis will not request of list of objects from your library.
- Respond to the ECM_GETOBJECT message and return a list of objects via ECOreturnObjects. It is important to note that your ECOobject structure should contain unique ids. During subsequent calls the EXTCompInfo mCompId member will contain this id to inform you of the type of object.
- Respond to ECM_OBJCONSTRUCT, ECM_OBJDESTRUCT and ECM_OBJECT_COPY to ensure that your objects are created, destructed and copied.
- Respond to ECM_GETMETHODNAME and ECM_GETPROPNAME to return any methods and properties that your object may have.
- Respond to ECM_PROPERTYCANASSIGN, ECM_SETPROPERTY, ECM_GETPROPERTY in normal manual to manage your objects' properties.
- Finally, respond to ECM_METHODCALL to inform any of your objects' methods.

It is important to note that during all of the above messages (except ECM_GETMETHODNAME, ECM_GETPROPNAME and ECM_OBJECT_COPY) lParam will contain a unique reference to your object. You should use ECOfindNVObject to retrieve your objects' data.

It is also important to note how you require your objects to be managed. For example the FILEOPS example uses a container to hold the actual object. The object is only released/freed when a reference count gets to zero. This allows several Omnis object variables to point to the same FILEOPS object (similar to COM).

An example of use may be (excerpts from FILEOPS):

```
ECOobject fileObjects[ cObject_Count ] =
{ // Unique external ID Resource Number Flags
  cObject_FileOps, 2000, 0, 0
};
extern "C" qlong OmnisWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_CONNECT:
    { // Component library contains NV objects & should always be loaded
      return EXT_FLAG_LOADED | EXT_FLAG_ALWAYS_USABLE | EXT_FLAG_NVOBJECTS;
    }
    case ECM_GETOBJECT:
    { // Omnis is requesting a list of our object class names
      return ECOreturnObjects(gInstLib,eci, &fileObjects[0], cObject_Count);
    }
    case ECM_OBJCONSTRUCT:
    { // Omnis is requesting the construct of a new object.
     if ( eci->mCompId==cObject_FileOps )
      {
       tqfFileOpsContainer* object = (tqfFileOpsContainer*) ECOfindNVObject(eci->mOmnisInstance, lParam );
       if ( !object )
        {
          tqfFileOpsContainer* obj = new tqfFileOpsContainer((qobjinst)lParam);
          ECOinsertNVObject(eci->mOmnisInstance,lParam,(void*)obj);
        }
        return qtrue;
      }
      return qfalse;
    }
    case ECM_OBJDESTRUCT:
    { // Omnis is requesting the destruction of your object
      if (eci->mCompId==cObject_FileOps && wParam==ECM_WPARAM_OBJINFO)
      {
        void* object=ECOremoveNVObject(eci->mOmnisInstance,lParam );
        if ( object )
        {
          tqfFileOpsContainer* fileOps = (tqfFileOpsContainer*)object;
          delete fileOps;
        }
      }
      return qtrue;
    }
    case ECM_OBJECT_COPY:
    { // Omnis requires a new object to be created from an existing one
      objCopyInfo* copyInfo = (objCopyInfo*)lParam;
      tqfFileOpsContainer* srcobj = (tqfFileOpsContainer*) ECOfindNVObject(eci->mOmnisInstance,copyInfo->mSour
      if ( srcobj )
      {
        tqfFileOpsContainer* destObj = qfFileOpsContainer*)ECOfindNVObject(eci->mOmnisInstance, copyInfo->mDes
        if ( !destObj )
        {
          destObj = new tqfFileOpsContainer( (qobjinst)copyInfo->mDestinationObject,srcobj);
          ECOinsertNVObject( eci->mOmnisInstance, copyInfo->mDestinationObject, (void*)destObj );
```

```
      }
      else
        destObj->setObject( (qobjinst)copyInfo->mDestinationObject,srcobj );
    }
    break;
  }
  case ECM_GETMETHODNAME:
  { // Omnis is requesting a list of our objects' methods
    if ( eci->mCompId==cObject_FileOps )
      return ECOreturnMethods( gInstLib, eci, &fileObjFuncs[0], cIMethod_Count );
    break;
  }
  case ECM_GETPROPNAME:
  { // Omnis is requesting a list of our objects' properties
    // but we don't have any so simply return
   break;
  }
  case ECM_PROPERTYCANASSIGN:
  case ECM_SETPROPERTY:
  case ECM_GETPROPERTY:
  {   // Omnis requires property management
      // but we don't have any so simply return
      break;
  }
  case ECM_METHODCALL:
  { // Omnis requires a method to be invoked
    if ( eci->mCompId==cObject_FileOps )
    {
      void* object = (void*)ECOfindNVObject( eci->mOmnisInstance, lParam );
      tqfFileOpsContainer* fileOps = (tqfFileOpsContainer*)object;
      if ( fileOps->mObject )
        return fileOps->mObject->methodCall( Msg, wParam, lParam, eci );
      }
      break;
    }
  }
  return DefWindowProc( hwnd,Msg,wParam,lParam,eci);
}
```

Object Base Classes

OmnisObject is a base class that can be used for representing an external object that is being implemented. *OmnisObjectContainer* is a container for OmnisObject objects which could then be cast back to the external object class when needed.

These base classes provide a starting point and a container for multiple external component classes – you could offer more object classes from one external component project (e.g. workers and non-visual objects) and keep your classes in a OmnisObjectContainer.

Furthermore, these track the number of references internally and delete themselves when references reach 0, in line with other Omnis external components.

Control Handlers

This section describes how to develop a control handler component. Control handlers are essentially components which handle other components, such as an ActiveX.

To create a control handler, you should follow these steps. Note that the CONTROLLIB and CONTROL classes are used for illustration purposes and do not exist in the Omnis component environment.

- Add EXT_FLAG_CTRLHANDLER to the component flags.
- Process ECM_GETHANDLERICON to inform Omnis of the HBITMAP to use for the components' group in the Component Store.

- Support for ECM_GETCOMPLIBINFO must be restructured. The component must provide the information for all the control libraries that it supports. The control library names that the component supports must include a file path as a prefix. An example of this would be:

```
if ( !pEci->mCompLibId )
{
  // Omnis is inquiring on the handler.
  ECOreturnCompInfo(gInstLib,pEci,CTRL_RES_NAME,0);
  // Id of first library
  pEci->mCompLibId = 1;
  return qtrue;
}
else
{
  // Omnis is inquiring on a control libraries
  CONTROLLIB* prevLib = getLib( pEci->mCompLibId );
  // Find library from id
  CONTROLLIB* nextLib=0;
  if ( prevLib )
    nextLib=prevLib->mNextLibrary;
  if ( nextLib )
  {
    // you have another library for Omnis
    EXTfldval exfldval;
    // Format of returned name <FilePath>+'\0'+<Library Name>
    str255 ctrlInfo = nextLib->mLibraryPath;
    // Space for NULL
    ctrlInfo[0]++;
    // Terminate Cstring
    ctrlInfo[ ctrlInfo.Length() ] = '\0';
    // Add control library name
    ctrlInfo.concat( nextLib->mLibraryName );
    exfldval.setChar( ctrlInfo );
    ECOaddParam(pEci,&exfldval);
    // Set Unique id of this library. The id may change between sessions.
    pEci->mCompLibId = nextLib->mLibraryId;
    // Return number of controls within library
     return nextLib->mControlCount;
  }
  // No more libraries
  return qfalse;
}
```

- Support for ECM_GETCOMPID message must return the unique identifier for the control. It should be noted that like the controls' library (mCompLibId), it is not necessary to maintain the same unique identifier between Omnis sessions. Example of ECM_GETCOMPID:

```
CONTROLLIB* curLib = getLib( pEci->mCompLibId );
if ( curLib )
{
  EXTfldval exfldval;
  exfldval.setChar( curLib->getControlName(wParam) );
  ECOaddParam(eci,&exfldval);
  pEci->mCompId = curLib->getControlId( wParam );
}
return cObjType_Basic;
```

- Support for ECM_GETCOMPICON must be restructured to return the HBITMAP for the control. The code for this, may be:

```
// wParam is true if the library is to be loaded. This enables fastest
// load time as it avoids loading the bitmap for every library that
// the handler supports.
if ( wParam )
{
  CONTROL* control = getControl( pEci->mCompLibId, pEci->mCompId );
  if (control)
  {
    EXTfldval exfldval;
    exfldval.setLong( (qlong) control ->getHBitMap() );
    ECOaddParam(eci,&exfldval);
    // Bitmap returned
    return qtrue;
  }
}
// No bitmap returned
return qfalse;
```

- Support for ECM_GETCONSTNAME must be restructured. Obviously control constants are not in the handlers' resources, so the constant list returned to Omnis must be manually built. An example of this would be:

```
CONTROLLIB* curLib = getLib( pEci->mCompLibId );
if ( curLib )
{
  EXTfldval extfldval;
  EXTqlist list; list.clear( listVlen );
  for ( qshort constCount=1; constCount<=curLib->mConstantCount; constCount++ )
  {
    EXTfldval cval; qlong line = list.insline();
    // Constant ID
    list.getColValRef( line , 1, cval, qtrue );
    cval.setLong( curLib->getConstId(constCount) );
    // Constant String
    list.getColValRef( line , 2, cval, qtrue );
    cval.setChar( curLib->getConstantName(constCount) );
  }
  extfldval.setList( list, qtrue);
  ECOaddParam(pEci,&extfldval);
  return qtrue;
}
// No constants
return qfalse;
```

- Support for ECM_GETPROPNAME, ECM_GETEVENTNAME, ECM_GETMETHODNAME must be restructured to return the properties for a control. An example of this would be:

```
CONTROL* cntrl = getControl( pEci->mCompLibId, pEci->mCompId);
if (cntrl)
{
  EXTfldval extfldval;
  EXTqlist list; list.clear( listVlen );
  for ( qshort num=1; num <= cntrl->getCount(); num ++ )
  {
    EXTfldval cval;
    qlong line = list.insertLine();
    // External id
    list.getColValRef( line , 1, cval, qtrue );
    cval.setLong(cntrl->getId(num));
    // Name
```

```
    list.getColValRef( line, 2, cval, qtrue );
    cval.setChar(cntrl ->getName(num) );
    // fft type of property/return type
    list.getColValRef( line , 3, cval, qtrue );
    cval.setLong(cntrl->getType(num) );
    // EXTD_ flags
    list.getColValRef( line, 4, cval, qtrue );
    cval.setLong(cntrl ->getFlags(num) );
    if ( ECM_GETPROPNAME==message )
    {
      // For properties you need to set the constant range
      // Const Start (zero if none)
      list.getColValRef( line , 6, cval, qtrue );
      cval.setLong(cntrl->getConstStart(num));
      // Const End (zero if none)
      list.getColValRef( line , 7, cval, qtrue );
      cval.setLong(cntrl->getConstEnd(num));
    }
    else
    {
      // For functions & events you need to add a
      // list containing parameters
      EXTqlist paramlist;
      paramlist.clear( listVlen );
      for ( qshort m=1; m<=cntrl->getParamCount(); m++ )
      {
        qlong paramline = paramlist.insertLine();
        // Parameter name
        paramlist.getColValRef( paramline, 1, cval, qtrue);
        cval.setChar(cntrl->getParamName(m));
        // fft Data type
        paramlist.getColValRef( paramline, 2, cval, qtrue);
        cval.setLong(cntrl->getParamType(m));
        // EXTD_ flags
        paramlist.getColValRef( paramline, 3, cval, qtrue);
        cval.setLong( cntrl->getParamFlags(m) );
      }
      list.getColValRef( line , 6, cval, qtrue );
      cval.setlist( paramlist, qtrue );
    }
  }
  extfldval.setList( list, qtrue);
  ECOaddParam(pEci,&extfldval);
  return qtrue;
}
// No properties
return qfalse;
```

- Finally, on receipt of the ECM_OBJCONSTRUCT, the control handler needs to construct the appropriate control. To enable this, the members mCompId and mCompLibId in the EXTCompInfo structure will contain the unique identifiers as declared during ECM_GETCOMPLIBINFO and ECM_GETCOMPID messages.

See also ECM_CONNECT, ECM_GETHANDLERICON, ECM_GETCOMPLIBINFO, ECM_GETCOMPID, ECM_GETCOMPICON, ECM_GETCONSTNAME, ECM_GETPROPNAME, ECM_GETEVENTNAME, ECM_GETMETHODNAME.

Background Components

When creating a background external component, you need to be aware of the differences between real components, and of the extra messages you may need to respond to.

A background component is created in a different way within Omnis during runtime and design mode. When you are designing a background component in design mode, the component will be given a child window (HWND) to draw within. During design mode, Omnis maintains this child window. During runtime, no child window is created. Omnis will call your object to paint, in an existing window at a certain location. Given this runtime/design mode difference, you should not use any HWND API that requires a window, such as WNDsetCapture() as you may not have a valid child window.

The following messages describe the differences or meaning when received by a background component.

ECM_OBJCONSTRUCT

ECM_OBJCONSTRUCT is sent to all component types. For background components you can test the wParam parameter and the ECM_WFLAG_NOHWND flag to tell if you are being created during design or runtime. For example :

```
case ECM_OBJCONSTRUCT:
{
  tqfTile* object = new tqfTile( hwnd );
  object->mIsRealHWND = !(wParam & ECM_WFLAG_NOHWND);
  ECOinsertObject( eci, hwnd, (void*)object, wParam );
  return qtrue;
}
```

The above example creates a new background component object, and stores a flag in the class so the control knows if it has a real child window or not.

ECM_CONNECT

The ECM_CONNECT needs to be handled for background external components. When Omnis calls your component with this message, the following code should be used. If the code is omitted, Omnis will create the control as a first class foreground object.

```
case ECM_CONNECT:
{
  return EXT_FLAG_LOADED | EXT_FLAG_BCOMPONENTS;
}
```

ECM_PRINT

ECM_PRINT is a very important message. Normally with standard components you pick up the WM_PAINT message so you can paint your control. During runtime, as you do not have a child window, you will never receive a WM_PAINT message. During design mode you do have a child window, so in theory you could get a WM_PAINT message, but you will not. To help background components keep a simple interface, Omnis sends only ECM_PRINT to your component during runtime and design mode to indicate that it needs to be painted. A WNDpaintStruct is passed in the **lParam** parameter which holds the area that needs painting and a HDC to paint within.

```
case ECM_PRINT:
{
  tqfTile* object = (tqfTile*)ECOfindObject( eci->mOmnisInstance, hwnd, wParam );
  WNDpaintStruct* ps = (WNDpaintStruct*)lParam;
  if ( object ) object->paint( ps->hdc, &ps->rcPaint );
    return qtrue;
}
```

The above example shows how to paint you background object in runtime or design mode.

Web Client Components

Writing Web Client components is almost identical to writing standard window components. In fact, you can build both from the same source. There are however some small differences.

- You will need to link against a different set of libraries. Use the example project files as a guide.
- The final DLL/shared library name must match the component library name as specified by your resources. The library name resource ID is returned by ECOreturnCompInfo as a response to the ECM_GETCOMPLIBINFO. As a rule of thumb, all our web client component names start with "FORM", i.e. FORMTREE, FORMTIME, etc.
- Web client components must respond to the ECM_GETCOMPSTOREGROUP message and return the group name with ECOreturnCStoreGrpName ECM_GETVERSION (see Chapter 2—Components Reference). The group name must be "WEB Components" for web controls and "WEB Background Objects" for web background objects.
- Web client components must be data bound in order to manipulate Omnis data. There is no other way of telling the client that data has changed and needs to be sent to the server for the next event. There is a message ECM_HASPRIMARYDATACHANGED (see Chapter 2—Components Reference) which the component needs to implement. You use it to tell the web client if the primary data has been changed by the user. If the component only displays data, it can have non-data bound properties which take instance variable names, but the component will not know when the data has changed.
- Web client components need to implement the following additional messages which deal with focusing and mouse clicks.
  ECM_CANFOCUS
  ECM_ CANCLICK
  (see Chapter 2—Components Reference)
- In addition, web client components must implement a proper versioning system. There is a new ECOreturnVersion function which must be used as a response to ECM_GETVERSION (see Chapter 2—Components Reference).
- ECOsendEvent function will always return true when called from web client components. In order to receive a result, the component must implement the ECM_EVENTRESULT message (see Chapter 2—Components Reference).
- Some functions or classes require additional parameters when used from web client components. These are
  EXTBMPref::EXTBMPref
  EXTCURref::EXTCURref
  (see Chapter 3—EXTBMPref/EXTCURref Class Reference)
- The EXTfile class and related functions are currently not supported.

The new generic stationary in the MACIDE folder (Mac only) includes basic code needed for writing web client components.

Chapter 2—Structures, Messages & Functions

This section describes control, resource allocation and general functions provided by the component library. Where object classes provide additional functions related to their operation, these are documented at the end of the relevant section.

Structures

ECOmethodEvent (for methods)

This is the structure defining information about a component method. The address to a table of method items should be used with the **ECOreturnMethodsEvents** API.

See some of the samples for an example.

```
struct ECOmethodEvent
{
  qlong      mId;
  qlong      mNameResID;
  qlong      mReturnDataType;
```

```
  qlong      mParameterCount;
  ECOparam*  mParameters;
  qlong      mFlags;
  qlong      mExFlags;
};
```

- **mId** - The unique identifier, within the method table, for the method. All external methods must have a positive number and must not be zero. All negative numbers are assumed to be Omnis internal methods, presently only Omnis internal methods ECF_CUSTOM & ECF_ABOUT are supported (neither have any parameters).
- **mNameResID** - Resource id which contains the method name. Method names should, ideally, be unique to avoid ambiguity in Omnis notation. If there is a clash between Omnis and the component method names, you may use a prefix of '::' to reference the external method. For example, Calculate #1 as $cobj.$::clashMethod.
- **mReturnDataType** - Returned data type of type fftxxx. Specify 0 for no returned data (e.g. void) and fftNone for an unspecified data type.
- **mParameterCount** - Number of parameters for the method. Specify zero for no parameters.
- **mParameters** - Pointer to an array of parameters. Specify NULL if there are no parameters.
- **mFlags** - Method flags of type EXTD_FLAG_xxxx.
- **mExFlags** - Use zero. Extended flags for future enhancement.

Once a table of methods has been returned, you should be ready to receive the ECM_METHODCALL message.

If the method is to support parameters, you need to supply information describing the parameters' properties. This is the parameter structure.

```
struct ECOparam
{
  qlong mNameResID;
  qlong mDataType;
  qlong mFlags;
  qlong mExFlags;
};
```

- **mNameResID** - Resource id which contains the parameters' name.
- **mDataType** - fftxxx data type of the parameter. Use fftNone for an unspecified data type.
- **mFlags** - Parameter flags of type EXTD_FLAG_xxxx. Examples are EXTD_FLAG_PARAMOPT and EXTD_FLAG_PARAMALTER.
- **mExFlags** - Must be zero. Extended flags for future enhancement.

Example of a method table

```
// The parameters
ECOparam CALENDARparams[2] =
{
  // string 7000 for param name, fftInteger type
  7000, fftInteger, 0, 0,
  // string 7001 for param name, fftInteger type
  7001, fftInteger, 0, 0
};
// The method table
ECOmethodEvent CALENDARmethods[3] =
{
  cCalendarMethodSetDayIcon, 6000, 0, 2, &CALENDARparams[0], 0, 0,
  cCalendarMethodClearDayIcons, 6001, fftInteger, 1, &CALENDARparams[0], 0, 0, cCalendarMethodGetDayIcon, 6002
};
```

```
// method cCalendarFuncSetDayIcon uses string 6000 for its name,
// no return type, 2 parameters, the address to a parameter.
// The last two items are method flags, see member description above.
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETMETHODNAME:
    {
      // you want to support method, so send OMNIS the method table.
      return ECOreturnMethods( gInstLib, eci, &CALENDARmethods[0], 3 );
    }
    case ECM_METHODCALL:
    {
      // OMNIS code is calling your component method
      qlong methodID = ECOgetId(eci);
      switch(methodID)
      {
        case cCalendarMethodSetDayIcon: …….
        case cCalendarMethodClearDayIcons: …….
        case cCalendarMethodGetDayIcon:
        {
          // this method supports parameters
          // so get information for parameter 1
          EXTParamInfo* param = COfindParamNum( eci, 1 );
          // create an EXTfldval from the information data
          EXTfldval passedParam( (qlong)param->mData );
          qlong valuePassed = passedParam.getLong();
          …….
          break;
        }
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnMethodsEvents, ECM_METHODCALL, EXTD_FLAG_PARAMOPT, EXTD_FLAG_PARAMALTER

ECOmethodEvent (for events)

This is the structure defining information about a component's events. The address to a table of events information should be used with the **ECOreturnMethodsEvents** API call.

```
struct ECOmethodEvent
{
  qlong mId;
  qlong mNameResID;
  qlong mReturnDataType;
  qlong mParameterCount;
  ECOparam* mParameters;
  qlong mFlags;
  qlong mExFlags;
};
```

- **mId** - The unique identifier, within the event table, for the event. All external events must have a positive number and must not be zero. All negative numbers are assumed to be Omnis internal events. For a list of supported internal events, look for ECE_ in EXTDEFS.HE.
- **mNameResID** - Resource id which contains the event name. Event names must be unique and must not clash with Omnis internal events. The string 'ev' is used automatically as a prefix for any event.
- **mReturnDataType** - Returned data type of type fftxxx. Specify 0 for no returned data (e.g. void) and fftNone for an unspecified data type.
- **mParameterCount** - Number of parameters for the event. Specify zero for no parameters.
- **mParameters** - Pointer to an array of parameters. Specify NULL if there are no parameters.
- **mFlags** - Event flags of type EXTD_FLAG_xxxx.
- **mExFlags** - Use zero. Extended flags for future enhancement.

Once a table of event information has been returned, you can use the **ECOsendEvent** API.

If your events support parameters, you need to supply information describing the parameters. See the *Component Methods* section for a description of the **ECOparam** structure, or see some of the example components.

Example of an events table

```
// The event parameters
ECOparam SLIDERnewPos[1] =
{
  // resource 6000 for its name and type fftInteger
  6000, fftInteger, 0, 0
};
// The event table
ECOmethodEvent SLIDERevents[3] =
{
  cSliderEvStartSlider, 5000, 0, 0, 0, 0, 0,
  cSliderEvEndSlider, 5001, 0, 0, 0, 0, 0,
  cSliderEvNewSliderPos, 5002, 0, 1, &SLIDERnewPos[0], 0, 0
};
// function cSliderEvNewSliderPos uses string 5002 for its name, no return
// type, 1 parameters, the address to a parameter table. The last two items
// are event flags, see memeber description above.
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETEVENTNAME:
    {
      // you want to support events, so send OMNIS the event table.
      return ECOreturnEvents(gInstLib,eci,&SLIDERevents[0],3);
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
// Events can be sent using the ECOsendEvent API.
// e.g. ECOsendEvent( mHWnd, cSliderEvStartSlider, 0, 0 );
// or with an event parameter
// EXTfldval evParam;
// evParam.setLong( 10 );
// ECOsendEvent( mHWnd, cSliderEvNewSliderPos, &evParam, 1 );
```

ECOproperty

This is the structure of a single property. The address to a table of properties should be used with the **ECOreturnProperties** API when Omnis calls your component with a **ECM_GETPROPNAME** message.

See some of the samples for an example.

```
struct ECOproperty
{
  qlong mPropID;
  qlong mNameResID;
  qlong mDataType;
  qlong mFlags;
  qlong mExFlags;
  qlong mEnumStart;
  qlong mEnumEnd;
};
```

- **mPropID** - Property Identifier. External properties ids must be positive and unique within the property table. These id's link the Omnis data with the associated property and therefore must not change.
- **mNameResID** - Resource id for the property name. Property names should, ideally, be unique to avoid ambiguity in Omnis notation. If there is a clash between Omnis and the component property, you may use a prefix of '::' to reference the external property, e.g. Calculate #1 as $cobj.$::clashProperty.
- **mDataType** - fftxxx data type.
- **mFlags** - EXTD_FLAG_xxx.
- **mExFlags** - Extended flags for future enhancements.
- **mEnumStart** - Constant id enumeration start (0 if not required).
- **mEnumEnd** - Constant id enumeration end (0 if not required).

Example Property Table

```
ECOproperty OMNISICNproperties[4] =
{
  cOmnisIcnBackColor, 4000, fftInteger, EXTD_FLAG_PWINDCOL, 0, 0, 0,
  cOmnisIcnIsTransparent, 4001, fftBoolean, 0, 0, 0, 0,
  cOmnisIcnIconId, 4002, fftInteger, EXTD_FLAG_PWINDICON, 0, 0, 0,
  cOmnisIcnScale, 4003, fftBoolean, 0, 0, 0, 0
};
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPROPNAME:
    {
      // you want to support properties, so send OMNIS
      // the property table.
      return ECOreturnProperties( gInstLib, eci, &OMNISICNproperties[0], 4 );
    }
    case ECM_PROPERTYCANASSIGN:
    {
      // OMNIS wants to know if you allows assignment to a property
      qlong propID = ECOgetId(eci);
      // you should return 1L if the
      // propID ( e.g. cOmnisIcnBackColor ) can be assigned.
      return 0L;
```

```
    }
    case ECM_SETPROPERTY:
    {
      // OMNIS is informing you to set a property value.
      qlong propID = ECOgetId(eci);
      // get the parameter information
      EXTParamInfo* param = ECOfindParamNum( eci, 1 );
      // create a EXTfldval object containing the new value
      EXTfldval newValue( (qlong)param->mData );
      // assign property 'propID' the value stored in 'newValue'
      // always return 1L if you handled the assignment.
      return 1L;
    }
    case ECM_GETPROPERTY:
    {
      // OMNIS wants to know a property value
      qlong propID = ECOgetId(eci);
      // prepare a EXTfldval for return
      EXTfldval returnVal;
      // you must return the value for 'propID', the value 10
      // is returned for this example
      returnVal.setLong( 10 );
      // send the return value back to OMNIS
      ECOaddParam(eci,& returnVal);
      // always return 1L if you handled the call.
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

Once a table of properties has been returned, you should be ready to receive the **ECM_PROPERTYCANASSIGN**, **ECM_SETPROPERTY** and **ECM_GETPROPERTY** messages.

The property flags are used to describe information about a property in the property table that must be returned to Omnis if you intend to support properties.

See also ECOreturnProperties, ECM_GETPROPNAME, ECM_PROPERTYCANASSIGN, ECM_SETPROPERTY, ECM_GETPROPERTY.

EXTclipType

The following enum values are used with the function ECOclipboardHasFormat().

```
enum EXTclipType
{
  eExtClipText = 0,
  eExtClipPicture = 1
};
```

- **eExtClipText** – use this enum when testing the clipboard for text data.
- **eExtClipPicture** - use this enum when testing the clipboard for picture data.

See also ECOclipboardHasFormat

EXTCompInfo

```
struct EXTCompInfo
{
```

```
  qlong        mCompLibId;
  qlong        mCompId;
  void*        mGdata;
  EXTHANDLE    mOmnisInstance;
  EXTParamInfo*  mParamFirst;
  void*        mPrivate;
  EXTADDR      mECOCallBack;
  EXTADDR      mGDICallBack;
  EXTADDR      mHWNDCallBack;
  EXTADDR      mFVALCallBack;
  EXTADDR      mQLISTCallBack;
  EXTADDR      mBMPCallBack;
  EXTADDR      mCRBCallBack;
  EXTADDR      mPRICallBack;
  EXTADDR      mQFILECallBack;
  locptype*    mLocLocp;
  locptype*    mInstLocp;
  EXTADDR      mDAMCallBack;
};
```

- **mCompLibId** - Contains the unique identifier for the component library. This value should only be used by control handlers.
- **mCompId** - Unique identifier for the control.
- **mGdata** - Pointer which is maintained by the external component.
- **mOmnisInstance** - Instance of Omnis.
- **mParamFirst** - Pointer to the first parameter.
- **mPrivate** - Private pointer used by Omnis. The component must not alter this member.
- **mECOCallBack, mGDICallBack, mHWNDCallBack, mFVALCallBack, mQLISTCallBack, mBMPCallBack, mCRBCallBack, mPRICallBack, mQFILECallBack, mDAMCallBack(v3.1)** - Data for Omnis call-back functions. The component must not alter these members.
- **mLocLocp** - The context of the calling Omnis method. When the external component is not called from an Omnis method, it is identical to mInstLocp.
- **mInstLocp** - The context of the Omnis class instance, which contains the object instance. When the external component is not called from a class instance, it points at the library or root.

EXTParamInfo

This structure contains all the parameter information required for many ECM_xxxx messages and functions.

```
struct EXTParamInfo
{
  long        mId;
  long        mInfo;
  void*       mData;
  long        mParent;
  unsigned char mNum;
  char        mFlags;
  EXTParamInfo* mNext;
  void*       mItem;
  void*       mVpt;
};
```

- **mId** – Parameter id. Depends on the context in which the EXTParamInfo structure is used. For example, during property messages this will contain the unique property identifier (mPropID).

- **mInfo** – Not currently used.
- **mData** – Pointer to data.
- **mParent** – Not currently used.
- **mNum** – Specifies the parameter number. A value of zero indicates that it is a return parameter.
- **mFlags** – Flags of type EXTC_FLAG_xxxx for the parameter.
- **EXTC_FLAG_EXTDEL** – Indicates that the parameter should be > deleted by the component. The component must not manually set > this flag.
- **EXTC_FLAG_PARAMCHANGED** – Indicates that the parameter has > been changed. The component must not manually set this flag.
- **EXTC_FLAG_HASITEM (v3.1) -** Indicates that the EXTParamInfo > contains valid mItem and mVpt fields. These fields are required by > some of the new callbacks in v3.1. If building components for 3.1 > you should return this flag during connect.
- **mNext** – Pointer to the next EXTParamInfo structure (may be NULL).
- **mItem (v3.1)** – Contains pointer to an Omnis item reference. Required by some new callbacks in v3.1.
- **mVpt (v3.1)** – Contains pointer to an Omnis parameter info structure. Required by some new callbacks in v3.1.

EXTParamTypeInfo (v3.1)

Returns information about the Omnis data field.

```
struct EXTParamInfo
{
  qshort     mType;
  qshort     mSubType;
  qlong      mLength;
  str255     mName
};
```

- **mType** – The Omnis data type.
- **mSubType** – The Omnis data sub type.
- **mLength** – The maximum length in bytes or characters of the data field. Zero means unlimited (10,000,000).
- **mName** – The Omnis data field name.

See also ECOgetParamInfo

EXTSerialise (v3.1)

Structure used by the IS_SERIALISED control message.

```
struct EXTserialise
{
  str255    mProductCode;
  str255    mFunctionCode;
  str255    mSerial;
  str255    mNotes;
};
```

- **mProductCode** – Product code supplied by component. Must be 4 alpha/numeric characters.
- **mFunctionCode** – Functionality code returned by Omnis. These consist of 4 alpha/numeric characters describing the enabled functionality.

- **mSerial** – Complete serial number. Returned by Omnis.
- **mNotes** – Notes as entered with the serial number by the user. Returned by Omnis.

See also ECOisSerialised, IS_SERIALISED

Flags

EXTD_OBJFLAG_xxx

These defines are applied to the ECOobject structure which is used to return a list of objects that the component supports.

EXTD_OBJFLAG_SINGLE_NOTIFY

Indicates that this is a worker object using pushWorkerCallback and can only have a single outstanding notification.

EXTD_OBJFLAG_WORKER

Indicates that this is a worker object.

EXTD_EFLAG_xxx

These defines are used in the mExFlags member of the ECOproperty structure.

EXTD_EFLAG_REPFONT

Indicates that Omnis should use report fonts for this property.

EXTD_EFLAG_MVDBUTTON

Indicates that this is a button on an MVDesigner window/form.

EXTD_EFLAG_LISTDATEFORMATCUSTOM

anumPropList for this property lists $jsdateformatcustom

EXTD_EFLAG_LISTNUMBERFORMAT

anumPropList for this property lists $jsnumberformat

EXTD_EFLAG_ISDATANAME

This property needs to be treated like $dataname

EXTD_EFLAG_NOEXPORT

This property is not to be exported by Omnis X e.g. because it is read-only

EXTD_EFLAG_REPORT_MEASURE

This property is a report measurement: the dps are stored in enumStart.

EXTD_EFLAG_EXT_PROPERTIES_CRB

Set this for standard Omnis properties (defined in anums.he) which are handled by the component rather than using the Omnis core to handle the property.

EXTD_FLAG_xxx

These defines are used in the mFlags member of the ECOproperty structure.

EXTD_FLAG_BUTTON

Indicates that Omnis should provide a button on the Property Manager.

EXTD_FLAG_EDITRONLY

Indicates that Omnis stops editing of the property on the Property Manager.

EXTD_FLAG_ENUM

Indicates that the property is an ENUM. For this type of property, Omnis sends the component the ECM_GETPROPERTYENUMS message.

See also ECM_GETPROPERTYENUMS

EXTD_FLAG_EXTCONSTANT

Indicates the property is an external constant value.

EXTD_FLAG_FAR_SRCH

Indicates that the property will be searched on during find and replace.

EXTD_FLAG_FONTPROP

Indicates that the property is a font.

EXTD_FLAG_HIDDEN

Indicates that the property is hidden, that is, the property does not appear in the Property Manager at all.

EXTD_FLAG_INTCONSTANT

Indicates the property is an internal constant value. For example the following property entry (extract from Calendar) indicates that the property is a internal (i.e. Omnis) constant between constant ids, pre3DStyleF □ pre3DStyleL (See **DMCONST.HE** for the entire Omnis constant range).

```
cCalendar_HeadingMode,4002,fftInteger,EXTD_FLAG_INTCONSTANT,0,pre3DStyleF,pre3DStyleL
```

EXTD_FLAG_PARAMALTER

Indicates that the parameter can be altered during a function call.

See also ECOsetParameterChanged

EXTD_FLAG_PARAMOPT

Indicates that the function parameter (and every parameter after) is optional.

EXTD_FLAG_PRIMEDATA

Indicates the property is a data field. Each object may have only **one** primary data field and appears as the $dataname property in Omnis.

See also ECM_SETPRIMARYDATA, ECM_GETPRIMARYDATA, ECM_GETPRIMARYDATALEN, ECM_CMPPRIMARYDATA, ECM_PRIMARYDATACHAN

EXTD_FLAG_PROPACT

Indicates that the property appears on the Action tab.

EXTD_FLAG_PROPAPP

Indicates that the property appears on the Appearance tab.

EXTD_FLAG_PROPCOLS

Indicates that the property appears on the Columns tab.

EXTD_FLAG_PROPCUSTOM

Indicates that the property appears on the Custom tab (default).

EXTD_FLAG_PROPDATA

Indicates that the property appears on the Data tab.

EXTD_FLAG_PROPGENERAL

Indicates that the property appears on the General tab.

EXTD_FLAG_PROPGRP1

Mask for Property Manager tab.

EXTD_FLAG_PROPJAVA

Indicates that the property appears on the Java tab.

EXTD_FLAG_PROPPREFS

Indicates that the property appears on the Preferences tab.

EXTD_FLAG_PROPPANE

Indicates that the property appears on the Pane tab.

EXTD_FLAG_PROPSECTIONS

Indicates that the property appears on the sSections tab.

EXTD_FLAG_PROPTEXT

Indicates that the property appears on the Text tab.

EXTD_FLAG_PWINDCOL

Indicates that the popup color window should be provided.

EXTD_FLAG_PWINDCOL256

Indicates that the popup 256 color window should be provided. Useful for interfacing with non-Omnis components such as Active-X or Java Beans.

EXTD_FLAG_PWINDCURSOR (v3.1)

Indicates that the popup cursor window should be provided.

EXTD_FLAG_PWINDFSTYLE

Indicates that the popup font style window should be provided.

EXTD_FLAG_PWINDICON

Indicates that the popup icon window should be provided.

EXTD_FLAG_PWINDLSTYLE

Indicates that the popup line style window should be provided.

EXTD_FLAG_PWINDMLINE

EXTD_FLAG_PWINDPAT

Indicates that the popup pattern window should be provided.

EXTD_FLAG_PWINDSET

Indicates that the popup checkbox selection window should be provided.

EXTD_FLAG_PWINDTYPE

Mask for the popup window types.

EXTD_FLAG_RUNTIMEONLY

Indicates that the property is runtime only, that is, the property appears in the Property Manager during design mode if the Show runtime properties option is switched on.

EXTD_FLAG_SECTIONS

Indicates that the property appears on the sections tab.

EXTD_FLAG_SINGLESEL

Indicates that the property appears in the Property Manager when only one object is selected.

EXTD_FLAG_STATEONLY

Indicates that Omnis displays [Empty] or [Not Empty] in the Property Manager.

EXTD_FLAG_SUPPRESS

Indicates that the standard anum (see anums.he) property should be suppressed in the Property Manager.

General Messages

This section describes some of the messages you receive via your WNDPROC. For additional messages see the HWND and GDI message section.

ECM_ADDTOPRINTJOB

The ECM_ADDTOPRINTJOB message is send to a report object when the object is to add itself to the print job. This message will only be sent if you returned 1L as a response to the message ECM_CANADDTOPRINTJOB.

- **lParam** - points to the printInfo structure. This structure contains a pointer to the print job mJob of type PRIjob, and a pointer to the object information mObj of type PRIobjectStruct. See print manager documentation for more information about PRIobjectStruct and adding objects to a print job.

Returns:

If the component has added objects to the print job, return 1L. Otherwise return 0L.

```
case ECM_ADDTOPRINTJOB:
{
  tqfRepObj *obj = (tqfRepObj*)ECOfindObject( eci, hwnd );
  if ( obj )
  {
    printInfo *info = (printInfo*)lParam;
    info->mObj->mType = PRI_OBJ_TEXT;
    info->mObj->mAddEllipsis = qtrue;
    qprierr err = PRIaddObject( info->mJob, info->mObj );
    return err == PRI_ERR_NONE ? 1L : 0L;
  }
  return 0L;
}
```

ECM_BOBJ_EXERASE

The ECM_BOBJ_EXERASE message is sent to the **background** components to inquire on whether the background objects' frame region should be excluded from the erase background region.

Returns:

The component should return true if the components' frame region should be excluded, false otherwise.

ECM_CANADDTOPRINTJOB

External report objects can have full control over what is added to a print job when the object is about to be printed. In order to take advantage of this feature, you must implement this message and return 1L. You will then receive a ECM_ADDTOPRINTJOB message which allows you to add one or more objects supported by the print manager. See print manager documentation for more information about adding objects to a print job.

Returns:

Return 1L if you wish to control what is added to a print job, otherwise return 0L.

ECM_CANCLICK (Web Client 1.0)

The ECM_ CANCLICK message is sent, when the web client needs to know if the component can receive mouse messages.

- **wParam** – is 1 if the component is enabled, otherwise it is 0.

Returns:

Return 1L if the component can receive mouse messages, otherwise return 0.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_ CANCLICK:
    {
      // the component can receive mouse messages if it is enabled
      return wParam;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CANFOCUS (Web Client 1.0)

The ECM_CANFOCUS message is sent, when the web client needs to know if the component can receive the input focus.

- **wParam** – is 1 if the component is enabled, otherwise it is 0.

Returns:

Return 1L if the component can receive the input focus, otherwise return 0.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_ CANFOCUS:
    {
      // the component can receive the focus if it is enabled
      return wParam;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CANSHOWSYSTEMFOCUS (V3.2)

This message is send to the component when Omnis needs to know if the systems focus border is to be drawn around the component (Macintosh only).

Returns:

Return 1L if a focus border is to be drawn, otherwise return 0.

ECM_CMPPRIMARYDATA

The ECM_CMPPRIMARYDATA message is sent to the component to compare its objects' data with the data provided in parameter one.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CMPPRIMARYDATA:
    {
      EXTParamInfo* param = ECOfindParamNum(eci,1);
      if ( param && param->mData )
      {
        EXTfldval newValue( (qlong)param->mData );
        if ( newValue.compare( myComponentData )==0 )
          return DATA_CMPDATA_DIFFER;
      }
      return  DATA_CMPDATA_SAME;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_PRIMEDATA

ECM_COMPONENTCMD

The ECM_COMPONENTCMD message is sent to the component in response to the $cmd notation method being executed.

The $cmd method provides functionality to Omnis scripting language which might otherwise be inaccessible.

For example, the javabean component provides functionality to enumerate beans. However, this functionality is normally only available via a dialog. $cmd also provides this functionality without the use of the dialog.

Once invoked, all parameters are passed to the component.

An example of use may be :-

```
OMNIS script code :-
  Do $components.MyLibrary.$cmd(1)
External C++ Library code :-
  case ECM_COMPONENTCMD:
  {
    EXTParamInfo* param = ECOfindParamNum( pEci, 1 );
    if ( !param ) return rtnVal; // Method called with too few parameters
    EXTParamInfo* ecp = eci.findParam((qbyte)n);
    EXTfldval fval( (qfldval)ecp->mData );
    If ( fval.getLong()==1 )
      // Do processing …
    break;
  }
```

ECM_CONNECT

The ECM_CONNECT message is sent to the component after an Omnis instance has loaded the component.

Returns:

The component should return one or more of the following flags: -

- **EXT_FLAG_LOADED** - Component has been loaded successfully. The component must return this flag otherwise Omnis assumes the component failed to load.
- **EXT_FLAG_USABLE** Note: FOR INTERNAL USE ONLY. *A component* must not* return this flag. ***
- **EXT_FLAG_ALWAYS_USABLE** - Component is always available regardless of its load status. This flag enables components to be usable in Omnis **without** having to load it via the external component dialog. For example, Omnis OLE & Graph components both set this flag.
- **EXT_FLAG_REMAINLOADED** - Component remains loaded even after its usage has returned to zero. This flag provides the best component performance and may be used if the component connection process is too slow.
- **EXT_FLAG_HIDDEN (v3.3)**– Component will not be visible in the object notation tree displayed when creating variables of type 'Object'.
- **EXT_FLAG_DAM (v5.0)**- The external component is a DAM; must be set in addition to EXT_FLAG_SESSION for DAMs only.
- **EXT_FLAG_CTRLHANDLER** - Component is a control handler . Please refer to the section '**Control Handlers**' for more information.
- **EXT_FLAG_EVENTHANDLER** – Component in an event handler. Treatment of this flag is the same as EXT_FLAG_CTRLHANDLER.
- **EXT_FLAG_SESSION (v3.1)** – Component is a SQL session object. (Omnis Studio version 3.0 onwards). This flag is also used to elicit thread-safe behavior when writing multi-threaded components.
- **EXT_FLAG_OWNROOTNODE (v4.1)** – Specifies that the component should be assigned its own root node in the object notation tree displayed when creating variables of type 'Object'.
- **EXT_FLAG_BCOMPONENTS** - Component library contains only background components.
- **EXT_FLAG_NVOBJECTS** – Component library contains non-visual objects (either static functions or Omnis objects).
- **EXT_FLAG_PRI_OUTPUT** - Component library contains output devices.
- **EXTC_FLAG_HASITEM (v3.1) -** Indicates that the EXTParamInfo contains valid mItem and mVpt fields. These fields are required by some of the new callbacks in v3.1. If building components for 3.1 you should return this flag during connect.

Note: Most components do not need to catch this message. The default returned value in WNDdefWindowProc is EXT_FLAG_LOADED.

ECM_CONSTPREFIX

The ECM_CONSTPREFIX message is sent when Omnis requires the prefix string for all components' constants.

If the component requires a constant prefix, it should add a parameter containing the string.

Returns:

Return true if the constant prefix has been returned.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CONSTPREFIX:
    {
      EXTfldval prefixName;
      str15 prefixStr;
      prefixStr[0] = RESloadString( gInstLib, resourceID,&prefixStr[0], 15 );
```

```
      prefixName.setChar(prefixStr);
      ECOaddParam(eci,&prefixName);
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CONVFROMHPIXMAP (Studio 2.1)

The ECM_CONVFROMHPIXMAP message is sent to a picture format component when Omnis requires an HPIXMAP to be converted into raw binary picture data (as stored on disk).

Parameters:

- **lParam** – HPIXMAP required to convert.

Returns:

Return qtrue if the component has successfully converted the HPIXMAP to binary data, qfalse otherwise.

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo* ec
{
  case ECM_CONVFROMHPIXMAP:
  {
    qbool rtnVal = qfalse;
    HPIXMAP thePixMap = (HPIXMAP)lParam;
    qHandle binaryPCX;
    if ( PixmapToPCX(thePixMap ,binaryPCX) )
    {
      EXTfldval fval;
      fval.setHandle(fftBinary,binaryPCX,qfalse);
      ECOaddParam(eci,&fval);
      rtnVal = qtrue;
    }
    binaryPCX.setNull();
    return rtnVal;
  }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CONVHEADER (Studio 2.1)

The ECM_CONVHEADER message is sent to a picture format component when Omnis requires a picture formats' header to be added or removed.

- **wParam** – True if a header should be added, false if it should be removed.
- **Parameter 1** – Picture data.

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo* ec
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CONVHEADER:
    {
      EXTParamInfo* param = ECOfindParamNum(eci,1);
```

```
      if ( param )
      {
        EXTfldval fldval( (qfldval)param->mData );
        srcHan = fldval.getHandle (qfalse);
        qHandle destHan;
        if ( wParam )
        { // Add tqgpict header (& any other component header)
          addPCXheader(srcHan,destHan);
        }
        else
        { // Remove tqgpict header (& any other component header)
          removePCXheader(srcHan,destHan);
        }
        EXTfldval fval;
        val.setHandle(fftBinary,destHan,qfalse);
        ECOaddParam(eci,&fval);
        return qtrue;
      }
      return qfalse;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CONVTOHPIXMAP (Studio 2.1)

The ECM_CONVTOHPIXMAP message is sent to a picture format component when Omnis requires an raw picture data to a HPIXMAP. It is important to note that the data supplied may, or may not, include any headers.

Parameters:

- **Parameter 1** – Picture data.

Returns:
Return an HPIXMAP handle, NULL otherwise.

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg,
WPARAM wParam, LPARAM lParam, EXTCompInfo* eci )
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CONVTOHPIXMAP:
    {
      EXTParamInfo* param = ECOfindParamNum(eci,1);
      EXTfldval fldval( (qfldval)param->mData );
      qHandle theData = fldval.getHandle (qtrue);
      HPIXMAP thePixmap = PCXtoPixMap( theData );
      return (qlong) thePixmap;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_CUSTOMTABNAME

The ECM_CUSTOMTABNAME message is sent to the component when Omnis requires the name of the custom tab in the Property Manager.

The component should add a parameter containing the custom tab character name.

A component should call ECOsetCustomTabName to provide the necessary information.

Returns:
Return true if a custom tab name has been supplied.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CUSTOMTABNAME:
    {
      // use resource 8000 for the name of the tab in the Property Manager
      ECOsetCustomTabName( gInstLib, eci, 8000 );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOsetCustomTabName

ECM_DEBUGGING

The ECM_DEBUGGING message is sent to the component:

- just after a component library has been loaded (immediately after ECM_CONNECT).
- when sys(4000) to enable debugging has been called.
- when sys(4001) to disable debugging has been called.

Components may utilize this message to provide debugging statements in the trace log, and so on.

The debugging flag is maintained between Omnis sessions.

Parameters:

- **wParam** - True if debugging is enabled, false otherwise.

Returns:
Any returned value is ignored.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_DEBUGGING:
    {
      qbool debuggingOn = (qbool)wParam;
      if ( debuggingOn )
      {
        // If debugging is on, the component may wish to provide
        // verbose information to the developer via various
        // methods (e.g. trace log, and so on)
      }
      break;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_DISCONNECT

The ECM_DISCONNECT message is sent to the component before an Omnis instance unloads the component. It should always be passed to the WNDdefWindowProc.

Returns:

Any returned value is ignored.

Note: Most components do not need to catch this message.

ECM_EVENTRESULT (Web Client 1.0)

The ECM_EVENTRESULT message is sent to a Web Client external component, when the result of a custom event is returned from the server. Because events are executed on the server, the result returned from ECOsendEvent is meaningless and will always return qtrue in the Web Client environment. The true result will be sent as the ECM_EVENTRESULT message once the server returns control to the client.

Parameters:

- **wParam** - the event code which was specified when ECOsendEvent was called.
- **lParam** - the result 0 or 1.

Returns:

Return 1L.

See also ECOsendEvent

ECM_FMT_CANASSIGN

The ECM_FMT_CANASSIGN message is sent to the component when Omnis needs to know if a property can be written to. This message is used for format notation and even if the component does not respond to the message, it is assumed that the property can be written to.

Returns:

Return FMT_CANASSIGN if the property can be written to, return FMT_NOCANASSIGN otherwise.

See also ECM_PROPERTYCANASSIGN,Component Properties section.

ECM_FMT_GETPROPERTY

The ECM_FMT_GETPROPERTY message is sent to the component when Omnis needs to know the value of a property.. This message is used for format notation and even if the component doesn't respond to the message the property will be retrieved from the format.

Parameter one contains the current property value.

Returns:

Return FMT_VALID if the property was successfully retrieved, FMT_INVALID otherwise.

See also ECM_GETPROPERTY, Component Properties section.

ECM_FMT_SETPROPERTY

The ECM_FMT_SETPROPERTY message is sent to the component when Omnis needs to set the value of a property.. This message is used for format notation and even if the component doesn't respond to the message the property will be modified in the format.

Parameter one contains the new property value.

Returns:

Return FMT_VALID if the property was successfully modified, FMT_INVALID otherwise.

See also ECM_SETPROPERTY,Component Properties section.

ECM_GETCOMPICON

The ECM_GETCOMPICON message is sent to the component when Omnis requires the HBITMAP for the component icon. A component should add a long parameter containing the HBITMAP or may call ECOreturnIcon to provide the information. Please note that the HBITMAP returned belongs to Omnis and is deleted by Omnis when the component is of no further use.

Parameters:

- **wParam** - wParam is true if the library is available to the user.

Returns:

Return true if the bitmap has been returned, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETCOMPICON:
    {
      return ECOreturnIcon( gInstLib, eci, iconResID );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETCOMPID

The ECM_GETCOMPID message is sent when Omnis requires the object name, type and unique identifier.

The component should add a parameter which contains the character name of the object, it should also set the EXTCompInfo member mCompId to a unique identifier for that object. The mCompId is used by the component to determine to which type of object messages are referring.

Parameters:

- **wParam** - Contains a sequential number (starting from 1) which indicates the object which is being inquired upon.

Returns:

The component should return the object type cRepObjType_xxxx and/or cObjType_xxxx or FALSE if there are no more objects in the component.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETCOMPID:
    {
      // returns a single component of id ‘compID’ and
      // of type ‘cObjType_Basic’
      return ECOreturnCompID( gInstLib, eci, compID, cObjType_Basic );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnCompID

ECM_GETCOMPLIBINFO

The ECM_GETCOMPLIBINFO message is sent when Omnis requires the components' library name and the number of objects it supports.

Returns:

The component should add a parameter containing the character name of the component library and should also return the number of objects supported. A component may use the function ECOreturnCompInfo to provide the necessary information.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETCOMPLIBINFO:
    {
      // returns the name of the component library ( resource id )
      // and the number of components this library supports.
      return ECOreturnCompInfo( gInstLib, eci, LIB_RES_NAME, COMPONENT_COUNT );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnCompInfo

ECM_GETCOMPSTOREGROUP (Studio 2.1)

The ECM_GETCOMPSTOREGROUP message is sent to the component library when Omnis requires the name of the component store group.

The component should add a parameter containing the component store group name (maximum 31 characters), if required.

A component should call ECOreturnCStoreGrpName to provide the necessary information.

Returns:

Return true if a component store group name has been supplied.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETCOMPSTOREGROUP:
    {
      // use resource 8000 for the name of component store group
      ECOreturnCStoreGrpName( gInstLib, eci, 8000 );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnCStoreGrpName

ECM_GETCOMPSTOREICON (Studio 2.1)

The ECM_GETCOMPSTOREICON message is sent to the component when Omnis requires the bitmap of the component store group. This message will only be sent if the component library returned a component store group name (see ECM_GETCOMPSTOREGROUP).

The component should add a parameter containing the bitmap.

A component should call ECOreturnIcon to provide the necessary information.

Returns:

Return true if a bitmap has been supplied.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETCOMPSTOREICON:
    {
      // use resource 8000 for the component store groups' bitmap
      ECOreturnIcon ( gInstLib, eci, 8000 );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECM_GETCOMPSTOREGROUP, ECOreturnIcon

ECM_GETCONSTNAME

The ECM_GETCONSTNAME message is sent to the component when Omnis requires a list of the constants that the component library supports.

Constant resource strings are in the format of: -

- **Name** - The name of the constant as it appears in Omnis methods. Constant names may contain a group name (name prefixed by group name followed by a tilde ['~'] mark) which informs Omnis that the component constants should be sub-grouped.
- **Numeric value** - The numeric value of the constant.
- **Character value** - The character value of the constant.
- **Description** - The description of the constant.

A component should call ECOreturnConstants to provide the event information.

Returns:

Return true if the event list has been returned.

See also ECOreturnConstants

ECM_GETEVENTMETHOD

The ECM_GETEVENTMETHOD message is sent to the component when Omnis requires the list of method lines for the objects' event. This message is only sent during design mode when a new object has been created.

The component should add a single column list parameter or call function ECOreturnEventMethod.

Returns:

Return true if a method list has been provided, false otherwise.

```
example strings:
8000, "on evMyEvent"
8000, "; This event is sent for xxx reason"
8001, ""
8002, ""
8003, "on evMyEvent2"
8004, "; This event is sent for yyy reason"
// a break in the run is needed ( 8005 is missing )
8010, ""
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETEVENTMETHOD:
    {
      // this uses strings 8000 onward, until a gap in the run
      return ECOreturnEventMethod(gInstLib, eci, 8000 );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnEventMethod

ECM_GETEVENTNAME

The ECM_GETEVENTNAME message is sent to the component when Omnis requires a list of the events that the object supports.

A component should call ECOreturnFuncsEvents to provide the event information.

Returns:

Return true if the event list has been returned.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETEVENTNAME:
    {
      return ECOreturnFuncsEvents( gInstLib, eci, &eventTable[0], evtTableCnt );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnFuncsEvents

ECM_GETHANDLERICON

The ECM_GETHANDLERICON message is sent to the component when Omnis requires the HBITMAP for the control handler icon. A component should return the HBITMAP for the bitmap. Note that the HBITMAP returned belongs to Omnis and is deleted by Omnis when the control handler is of no further use.

Returns:

Return the HBITMAP of the handlers' icon.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETHANDLERICON:
    {
      // Provide OMNIS with a bitmap for the Component Store group.
      HBITMAP compStoreIcon = RESloadBitMap( gInstLib, COMP_STORE_GROUP_ID );
      return (qlong)compStoreIcon;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETMETHODNAME

The ECM_GETMETHODNAME message is sent to the component when Omnis requires a list of the methods that the object supports.

A component should call ECOreturnMethodsEvents to provide the method information.

Returns:

Return true if the function list has been returned.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETMETHODNAME:
    {
      return ECOreturnMethodsEvents(gInstLib, eci, &funcTable[0], funcTableCnt );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnMethodsEvents

ECM_GETOBJECT

The ECM_GETOBJECT message is sent to a library which supports non-visual objects.

A component should call ECOreturnObjects to provide the object information.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETOBJECT:
    {
      return ECOreturnObjects(gInstLib,eci,&objTable[0],objTableCnt);
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnObjects, EXT_FLAG_NVOBJECTS, Non-Visual Components

ECM_GETOBJECTRECT

The ECM_GETOBJECTRECT message is sent to the component to retrieve the initial dimensions of the object during design mode when the object is created via the Component Store drag and drop or by double-clicking.

Parameters:

- **lParam** - Pointer to qrect structure which should be populated with the initial dimensions of the object.

Returns:

Return qtrue if the object rectangle has been set, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETOBJECTRECT:
    {
      qrect* initialRect = (qrect*)lParam;
      // sets the controls initial size to 100, 100
      GDIsetRect( initialRect, 0, 0, 100, 100 );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETPICTFILEDESC (Studio 2.1)

The ECM_GETPICTFILEDESC message is sent to a picture format component when Omnis requires a string for the “Paste from file” file dialog.

The string returned must be a valid file filter string.

Returns:

Return qtrue if the component has returned a string, qfalse otherwise.

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg,
WPARAM wParam, LPARAM lParam, EXTCompInfo* eci )
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPICTFILEDESC:
    { // Return a string containing the picture file filter
      str15 name("PCX Files (*.pcx)|*.pcx|");
      EXTfldval fval; fval.setChar(name);
      ECOaddParam(eci,&fval);
      return qtrue;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETPICTFORMAT (Studio 2.1)

The ECM_GETPICTFORMAT message is sent to the component during the initial loading of the component. A component which supports picture conversion, for example PCX, should return a string containing the name of the format e.g. “JPEG” or “PCX” etc..

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg,
WPARAM wParam, LPARAM lParam, EXTCompInfo* eci )
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPICTFORMAT:
    { // Return a string ("PCX") containing the picture format
      str15 name("PCX");
      EXTfldval fval; fval.setChar(name);
      ECOaddParam(eci,&fval);
      return qtrue;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETPICTUREDIM

The ECM_GETPICTUREDIM message is sent to the component to retrieve the dimensions of the object which has been defined as cObjType_Picture.

Parameters:

lParam - Pointer to a qrect structure. The component should modify the members accordingly.

Returns:

Return true if the component has populated the structure, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPICTUREDIM:
    {
      qrect* pictDim = (qrect*)lParam;
      // returns the bounds of the picture you are currently displaying
      GDIsetRect( pictDim, 0, 0, mWidth, mHeight );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_GETPRIMARYDATA

The ECM_GETPRIMARYDATA message is sent to the component to obtain the data for an object.

If the component is handling the data for an object, it should return this in parameter one.

Returns:

Return true if the data has been supplied, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPRIMARYDATA:
```

```
    {
      EXTfldval exfldval;
      EXTParamInfo* newparam = ECOaddParam(eci,&exfldval);
      exfldval.setBinary(fftPicture,mPCXData,mPCXDataLen);
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_PRIMEDATA

ECM_GETPRIMARYDATALEN

The ECM_GETPRIMARYDATALEN message is sent to the component when Omnis requires the object's data length.

Returns:

The component should return the objects data length.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPRIMARYDATALEN:
    {
      return myDataLength;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_PRIMEDATA

ECM_GETPROPERTY

The ECM_GETPROPERTY message is sent to the component when Omnis requires the data for a property.

The component should add a return parameter which contains the property data.

Returns:

Return true if successful, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPROPERTY:
    {
      // propID is the id of the property defined in your proptable
      qlong propID = ECOgetId(eci);
      // Get the value of your property.
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also Component Properties section.

ECM_GETPROPERTYENUMS

The ECM_GETPROPERTYENUMS message is sent to the component when Omnis requires the enum list for a property (previously defined with EXTD_FLAG_ENUM).

The component should return a list containing the line data and, optionally, the marks which identify each line. After an item has been selected from the list, Omnis sends the component an ECM_SETPROPERTY message with the line data or the line mark (if a line mark was provided).

Returns:

Return true if enum list has been provided, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPROPERTYENUMS:
    {
      EXTqlist enumList;
      enumList.clear(listScol);
      for ( qshort i = 1; i<=5; i++ )
      {
        str255 enumName;
        enumName[0] = RESloadString(gInstLib, i, &enumName[1], 255 );
        enumList.insline( 0, &enumName, i );
      }
      EXTfldval returnVal;
      returnVal.setList( &enumList, qtrue );
      ECOaddParam( eci, &returnVal );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_ENUM

ECM_GETPROPNAME

The ECM_GETPROPNAME message is sent to the component when Omnis requires a list of the properties that the object handles.

A component should call ECOreturnProperties to provide the property list.

Returns:

Return true if the property list has been returned.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETPROPNAME:
    {
      return ECOreturnProperties( gInstLib, eci, &propTable[0], propTableCnt );
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnProperties

ECM_GETSTATICOBJECT

The ECM_GETSTATICOBJECT message is sent to a library which supports non-visual objects.

A component should call ECOreturnMethods to provide the static object information.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_GETSTATICOBJECT:
    {
      return ECOreturnMethods(gInstLib,eci, &objStaticTable[0], objStaticTableCnt);
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOreturnMethods, EXT_FLAG_NVOBJECTS, Non-Visual Components

ECM_GETVERSION

The ECM_GETVERSION message is sent when Omnis requires the version number of the component.

A component should call ECOreturnVersion to provide the version number. If the component fails to respond to this message then Omnis will assume a version number of 1.0.

For web client components, the version number of the component must be implemented as a string in the string resources of the component. The web client plug-in reads this string for the purpose of the automated download mechanism. See ECOreturnVersion for more details.

Returns:

Return the return value from ECOreturnVersion

See also GDIreadVersion, ECOreturnVersion

ECM_HASPRIMARYDATACHANGED (Web Client V1.0)

The ECM_HASPRIMARYDATACHANGED message is sent to web client components to determine if the components primary data has changed since the last ECM_SETPRIMARYDATA or ECM_GETPRIMARYDATA. When writing data bound web client controls, the control is responsible for maintaining its own modified state. This is so the web client only returns data for fields to the server, which have been changed by the user. Return one of the following:

- **ECMRET_NOTIMPLEMENTED** - default return value.
- **ECMRET_NOTCHANGED** - return this if the data has NOT been changed by the user since the last ECM_GETPRIMARYDATA or ECM_SETPRIMARYDATA. This should be the default return value for read only controls.
- **ECMRET_CHANGED** - return this if the data has been changed by the user since the last ECM_GETPRIMARYDATA or ECM_SETPRIMARYDATA.
- **ECMRET_CURROWCHANGED** - return this if the primary data is a single selection list and the current row has changed since the last ECM_GETPRIMARYDATA or ECM_SETPRIMARYDATA.
- **ECMRET_ROWSELECTCHANGED (v3.1) -** return this if the primary data is a multiple selection list and the current row and list selection state has changed since the last ECM_GETPRIMARYDATA or ECM_SETPRIMARYDATA
- **ECMRET_CURROWSELECTCHANGED (v3.1) -** return this if the primary data is a multiple selection list and the current row and list selection state of the current row only has changed since the last ECM_GETPRIMARYDATA or ECM_SETPRIMARYDATA

See also ECM_SETPRIMARYDATA, ECM_GETPRIMARYDATA

ECM_ICONDRAWENTRY

The ECM_ICONDRAWENTRY message is sent to inform the component to draw an icon for an object which has been defined as cObjType_IconArray.

Parameters:

lParam - Pointer to EXTIconArrayInfo structure (see Below).

Returns:

Return true if the icon was drawn, false otherwise (which results in Omnis drawing the icon).

```
struct EXTIconArrayInfo
{
  HDC       mHdc;
  qlong     mLine;
  qrect     mEntryRect;
  qrect     mDrawRect;
  qbool     mDrawFocus;
  qbool     mSelected;
  qbool     mDragging;
  qbool     mSmallIcons;
  EXTqlist* mListPtr;
};
```

- **mHdc** - Device context into which the icon should be drawn.
- **mLine** - The line number.
- **mEntryRect** - The rectangle of the icon array entry/cell.
- **mDrawRect** - The rectangle of the text or icon (dependant on whether the message is ECM_ICONDRAWENTRY or ECM_TEXTDRAWENTRY).
- **mDrawFocus** - True if the icon array entry/cell currently has the input focus.
- **mSelected** - True if the entry/cell is selected.
- **mDragging** - True if the entry is currently being dragged.
- **mSmallIcons** - True if the small icons are to be drawn (as opposed to large icons).
- **mListPtr** - List data pointer. This member contains the list variable pointer as defined in the property member data name.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_ICONDRAWENTRY:
    {
      EXTIconArrayInfo* arrayInfo = (EXTIconArrayInfo*)lParam;
      // Draw icon using info supplied in arrayInfo
      return 1L;
    }
    case ECM_TEXTDRAWENTRY:
    {
      EXTIconArrayInfo* arrayInfo = (EXTIconArrayInfo*)lParam;
      // Draw text using info supplied in arrayInfo
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also cObjType_IconArray, ECM_TEXTDRAWENTRY

ECM_INBUILT_OVERRIDE

The ECM_INBUILT_OVERRIDE message is sent from Omnis for certain **built in** properties which are normally handled by Omnis. Built in properties consist of anumFont, anumFontsize, anumTextColor, anumFontStyle, anumAlign, anumVScroll, anumHScroll, anumHScrolltips, anumVScrolltips, anumHorzscroll, anumVertscroll, anumEffect, anumHelpid, anumContextmenu, and anumFldStyle.

A component return 1L if it wants to manually maintain the built in property.

ECM_INSTALLLIBRARY

The ECM_INSTALLLIBRARY message is sent to a control handler when a request has been made to install another library via the #EXTCOMPS dialog>>Install button.

Returns:

Return true if message is processed, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_INSTALLLIBRARY:
    {
      // Control handler may wish to create a modal window to enable
      // controls to be installed/uninstalled etc…
      doInstallComponent();
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_ISCONVFORMAT (Studio 2.1)

The ECM_ISCONVFORMAT message is sent to a picture format component when Omnis is attempting to establish, from binary data, the picture format. This will be sent because the Omnis script function **pictformat** has been invoked.

It is important to note that the data supplied may, or may not, include any headers.

Parameters:

- **Parameter 1** – Picture data.

Returns:

Return qtrue if the picture data is in a format that the component supports, false otherwise.

```
extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg,
WPARAM wParam, LPARAM lParam, EXTCompInfo* eci )
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_ISCONVFORMAT:
    {
      EXTParamInfo* param = ECOfindParamNum(eci,1);
      if ( param )
      {
        EXTfldval fldval( (qfldval)param->mData );
        qHandle srcHan = fldval.getHandle(qfalse);
```

```
        if ( PCXObject::isPCXdata(srcHan) )
          return qtrue;
      }
      return qfalse;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_LISTDRAWLINE

The ECM_LISTDRAWLINE message is sent to inform the component to draw a list line for a object which has been defined as cObjType_List or cObjType_DropList.

Parameters:

- **lParam** - Pointer to EXTListLineInfo structure (see Below).

Returns:

Return true if the list line was drawn, false otherwise (which results in Omnis drawing the line).

```
struct EXTListLineInfo
{
HDC mHdc;
qrect mLineRect;
qlong mLine;
qbool mSelected;
EXTqlist* mListPtr;
qbool mDrawFocusRect;
};
mHdc - Device context into which the line should be drawn.
mLineRect - The rectangle of the line.
mLine - The line number.
mSelected - True if the line is selected.
mListPtr - List data pointer. This member contains the list variable pointer as defined in the property member
mDrawFocusRect - True if the focus rectangle should be drawn.
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_LISTDRAWLINE:
    {
      EXTListLineInfo* lineInfo = (EXTListLineInfo *)lParam;
      // paint line using info supplied in lineInfo
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also cObjType_List, cObjType_DropList

ECM_MEMORYDELETION

The ECM_MEMORYDELETION message is sent to inform the component library it needs to free previously allocated memory. This message should always be passed on to WNDdefWindowProc.

Note: Components do not need to catch this message, just pass it to the WNDdefWindowProc.

See also ECOmemoryDeletion

ECM_METHODCALL

The ECM_METHODCALL message is sent to inform the component that an objects' method has been invoked. All parameters for the method have been added to the EXTCompInfo structure. A component should add any return parameter.

Returns:

Return true if method has been invoked, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_METHODCALL:
    {
      // OMNIS code is calling your component method
      qlong methodID = ECOgetId(eci);
      switch(methodID)
      {
        case cMyMethod1: …….
        case cMyMethod2: …….
        case cMyMethod3: …….
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also Component Methods Section

ECM_NEWMETHODFLAGS

The ECM_NEWMETHODFLAGS message is sent to the component in response to the component sending a WM_CONTROL message (wParam = RESET_METHOD_FLAGS) to the objects HWND.

It enables controls such as Graphs to update the Property Manager depending on the context.

Returns:

The component should return the new EXTD_FLAG_xxx flags for the method.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_NEWMETHODFLAGS:
    {
      qlong newMethodFlags = 0;
      Cobj* object = (Cobj*)ECOfindObject( eci->mOmnisInstance, hwnd );
      if ( object )
      {
        qlong methodId = (qlong)lParam;
        newMethodFlags = object->getMethodFlags(methodId);
      }
      return newMethodFlags;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also RESET_METHOD_FLAGS

ECM_NEWPROPERTYFLAGS

The ECM_NEWPROPERTYFLAGS message is sent to the component in response to the component sending a WM_CONTROL message (wParam = RESET_PROPERTY_FLAGS) to the objects HWND.

Enables controls such as Graphs to update the Property Manager depending on the context.

Returns:

The component should return the new EXTD_FLAG_xxx flags for the property.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_NEWPROPERTYFLAGS:
    {
      qlong newPropertyFlags = 0;
      Cobj* object = (Cobj*)ECOfindObject( eci->mOmnisInstance, hwnd );
      if ( object )
      {
        qlong propId = (qlong)lParam;
        newPropertyFlags = object->getPropertyFlags( propId );
      }
      return newPropertyFlags;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
See al
```

See also RESET_PROPERTY_FLAGS

ECM_OBJCONSTRUCT

The ECM_OBJCONSTRUCT message is sent to instruct the component to construct an instance of the object.

Parameters:

- **hWnd** - The HWND of the object which is being constructed.
- **wParam** – For visual components wParam is either ECM_WPARAM_WINDOWOBJ or ECM_WPARAM_REPORTOBJ depending on the type of object to construct. For non-visual components wParam is either :-
  ECM_WPARAM_OBJMSG to indicate that the message is due to $construct.
  Or ECM_WPARAM_OBJINFO to indicate that the message is due to a new object being created.
  wParam may also contain the flag ECM_WFLAG_NOHWND for background objects.
- **lParam** – A pointer to the calling Omnis object instance. This qobjinst* can be stored an subsequently passed to functions such as ECOdoMethod() that require an object instance pointer.

Returns:

The component should return qtrue if it processes the message.

Note: It is good practice to use the ECO Object chain. New objects can be added to the chain with ECOinsertObject, and removed using ECOremoveObject. All supplied examples use this chain.

Example 1

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_OBJCONSTRUCT:
    {
      // create a new object - Cobj is an example class name
      Cobj* object = new Cobj( hwnd );
      // and add it to the ECO object chain
      ECOinsertObject( eci, hwnd, (void*)object );
      // if your component library supports multiple controls,
      // you can use eci->mCompId to determine what sort of control to create.
      return 1L;
    }
  }…
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

Example 2

```
extern "C" LRESULT OMNISWNDPROC MySqlObjProc(OMNISHWND hwnd, UINT Msg,WPARAM wParam,LPARAM lParam,EXTCompInfo*
{
  switch (Msg)
  {
    case ECM_OBJCONSTRUCT:
    {
      tqfDAMObjCont* object = (tqfDAMObjCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
      if ( !object )
      {
        tqfMySqlDAMObj* damObj = new tqfMySqlDAMObj(eci);
        tqfDAMObjCont* obj = new tqfDAMObjCont((qobjinst)lParam, damObj);
        ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
        return qtrue;
      }
    }…
  }
  return DAMdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_OBJDESTRUCT

The ECM_OBJDESTRUCT message is sent to instruct the component to destruct an instance of the object.

Parameters:

- **hWnd** - The HWND of the object which is to be destructed.
- **wParam** –For non-visual components wParam is either :-
  ECM_WPARAM_OBJMSG to indicate that the message is due to $destruct.
  Or ECM_WPARAM_OBJINFO to indicate that the message is due to a new object being destroyed.

Returns:

Any returned value is ignored.

Note: It is good practice to use the ECO Object chain. New objects can be added to the chain with ECOinsertObject, and removed using ECOremoveObject. All supplied examples use this chain.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_OBJDESTRUCT:
    {
      // retrieve and remove your object from the ECO object chain.
      Cobj* object = (Cobj*)ECOremoveObject( eci, hwnd );
      // and delete it.
      if ( object ) delete object;
        return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_OBJECTDATABLOCK

The ECM_OBJECTDATABLOCK message is sent to the component when Omnis is setting or getting the properties for the object. Most components ignore this message as property assignment/retrieval is provided automatically in Omnis, and in this case the component must return false.

However, some control types (ActiveX for example) require objects to be initialized using a data block. In this case, if wParam = ECM_WPARAM_BLOCKLOAD, the first parameter contains the property data for the object otherwise the component should add a parameter which contains the property data for the object.

Parameters:

- **wParam** - Contains either ECM_WPARAM_BLOCKSAVE or ECM_WPARAM_BLOCKLOAD.

Returns:

Return true if successful (i.e. the object supports data block property assignment), false otherwise.

ECM_OBJECT_COPY

The ECM_OBJECT_COPY message is sent to the component when a non-visual object assignment is required.

Parameters:

- **lParam** – lParam contains a pointer to a objCopyInfo structure which contains the copy information.

Returns: Any return value is ignored.

See also EXT_FLAG_NVOBJECTS, Non-Visual Components

ECM_OBJECT_REBUILD

The ECM_OBJECT_REBUILD message is sent to the component to inquire whether a rebuild of a non-visual objects' properties and/or methods is required.

Returns:

Return true if the object requires a rebuild.

See also EXT_FLAG_NVOBJECTS, Non-Visual Components

ECM_OBJINITIALIZE

The ECM_OBJINITIALIZE message is sent twice during the construction of an object. Once, just before any properties have been set, and once after.

Parameters:

- **wParam** - wParam contains false before the object is initialized (i.e. properties set), true after the object has been initialized.

Returns:

Any returned value is ignored.

Note: Components do not need to catch this message, just pass it on the WNDdefWindowProc.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_OBJINITIALIZE:
    {
      // You may need to load other DLL's once only.
      // after, you always need to pass this message
      // on to WNDdefWindowProc
      return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_PAINTCONTENTS

The ECM_PAINTCONTENTS message is sent to inform the component to draw the droplist contents window for a object which has been defined as cObjType_DropList.

Parameters:

- **lParam** - Pointer to EXTListLineInfo structure (see ECM_LISTDRAWLINE).

ECM_PRIMARYDATACHANGE

The ECM_PRIMARYDATACHANGE message is sent to inform the component that its objects data has changed. Most components ignore this message, but more specialized components may need to complete additional data processing after the data has changed.

Returns:

Any return value is ignored.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_PRIMARYDATACHANGE:
    {
      Cobj* object = (Cobj*)ECOfindObject( eci->mOmnisInstance, hwnd );
      if ( object )
      {
        // … Additional processing …
```

```
        object->inval();
      }
      break;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_PRIMEDATA

ECM_PRINT

The ECM_PRINT message is sent by Omnis to inform the component to print the object. You will also receive ECM_PRINT messages for background components when they need to be painted. Background objects do not receive WM_PAINT messages.

Parameters:

- **wParam** - **Picture object type**: wParam contains ECM_WPARAM_PICTNOSCALE bit set if no scaling if required.
- **lParam** - lParam contains a pointer to a WNDpaintStruct structure which contains the printer HDC and the object print rectangle.

Parameter 1 - contains any primary data (as during ECM_SETPRIMARYDATA message).

Returns:

Any return value is ignored.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_PRINT:
    {
      PCXObject* object = (PCXObject*)ECOfindObject( eci->mOmnisInstance, hwnd );
      if ( object )
      {
        EXTParamInfo* param = ECOfindParamNum(eci,1);
        if ( param && param->mData )
        {
          // Set objects' data from param variable.
          object->setPrimaryData( eci, param );
        }
        WNDpaintStruct* paintInfo = (WNDpaintStruct*)lParam;
        // you can paint your object using
        //
        // paintInfo->hdc
        //
        // using the bounds
        //
        // paintInfo ->rcPaint;
        object->print( paintInfo );
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECM_SETPRIMARYDATA

ECM_PRINTMAPPING

The ECM_PRINTMAPPING message is sent to the component to inquire on any print mapping required.

Print mapping enables Omnis to suitably scale the object. See CALENDAR and PCX for examples.

Returns:

The component should return true if print mapping is required, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_PRINTMAPPING:
    {
      return 1L;
      // returns 1L for print mapping - scales object
      // dependent on print DPI
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_PROPERTYCALCTYPE

The ECM_PROPERTYCALCTYPE message is sent to the component when Omnis needs to know the calculation type for calculation properties. If a property is not a calculation, do not implement this message.

Returns:

Return ctySquare if the property is of type square bracket calculation (the actual calculations are embedded in text using square brackets. Return ctyCalculation if it is a standard calculation, i.e. field name or functions.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_ PROPERTYCALCTYPE:
    {
      // return the property calculation type
      EXTfldval calcType;
      calcType.setLong( ctySquare );
      ECOaddParam( eci, &calcType );
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECM_PROPERTYCANASSIGN

The ECM_PROPERTYCANASSIGN message is sent to the component when Omnis needs to know if a property can be written to or not.

Returns:

Return true if the property can be written to, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_PROPERTYCANASSIGN:
    {
      // propID is the id of the property defined in your proptable
      qlong propID = ECOgetId(eci);
      // you should return 1L if the property 'propID' is
      // assignable, and 0L if the property is read-only
      return 0L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also Component Properties section.

ECM_SETPRIMARYDATA

The ECM_SETPRIMARYDATA message is sent by Omnis to inform the component to set the data for the object. The first parameter contains the new data for the object.

Returns:

Return true if the component handles the data, false otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, wParam, LPARAM lParam, EXTCompInfo* eci )
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_SETPRIMARYDATA:
    {
      EXTParamInfo* param = ECOfindParamNum(eci,1);
      if ( param && param->mData )
      {
        EXTfldval newValue( (qlong)param->mData );
        // new value stored in EXTfldval 'newValue'
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also EXTD_FLAG_PRIMEDATA

ECM_SETPROPERTY

The ECM_SETPROPERTY message is sent to the component when Omnis requires a property to change.

Parameter one contains the new data for the property.

Parameters:

- **wParam** - wParam is set to ECM_WPARAM_PROPBUTTON if the Property Manager popup button was pressed to set the property. For example, a **file name** property may wish to use a file open dialog if the popup button was pressed. Please note that if wParam is ECM_WPARAM_PROPBUTTON, parameter one does **not** contain any data.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_SETPROPERTY:
    {
      // propID is the id of the property defined in your proptable
      qlong propID = ECOgetId(eci);
      // set the new value of your property.
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also Component Properties section.

ECM_SQLOBJECT_COPY (v3.1)

wParam is 0 (add to NV Chain), 1 (remove from NV Chain)

This message can be used to prevent Omnis from creating unnecessary copies of external objects. Once implemented you can simply create a single object instance and increment or decrement the usage count, depending on the value of wParam.

Parameters:

- **wParam** – if 0 increment usage count, if 1 decrement usage count.

ECM_TEXTDRAWENTRY

The ECM_TEXTDRAWENTRY message is sent to inform the component to draw the text for an object which has been defined as cObjType_IconArray.

Parameters:

- **lParam** - Pointer to EXTIconArrayInfo structure (see Below).

```
struct EXTIconArrayInfo
{
  HDC        mHdc;
  qlong      mLine;
  qrect      mEntryRect;
  qrect      mDrawRect;
  qbool      mDrawFocus;
  qbool      mSelected;
  qbool      mDragging;
  qbool      mSmallIcons;
  EXTqlist*   mListPtr;
};
```

- **mHdc** - Device context into which the text entry should be drawn.
- **mLine** - The line number.
- **mEntryRect** - The rectangle of the icon array entry/cell.
- **mDrawRect** - The rectangle of the text or icon (dependant on whether the message is ECM_ICONDRAWENTRY or ECM_TEXTDRAWENTRY).

- **mDrawFocus** - True if the icon array entry/cell currently has the input focus.
- **mSelected** - True if the entry/cell is selected.
- **mDragging** - True if the entry is currently being dragged.
- **mSmallIcons** - True if the small icons are to be drawn (as opposed to large icons).
- **mListPtr** - List data pointer. This member contains the list variable pointer as defined in the property member data name.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_TEXTDRAWENTRY:
    {
      EXTIconArrayInfo* arrayInfo = (EXTIconArrayInfo*)lParam;
      // Draw text using info supplied in arrayInfo
      return 1L;
    }
    case ECM_ICONDRAWENTRY:
    {
      EXTIconArrayInfo* arrayInfo = (EXTIconArrayInfo*)lParam;
      // Draw icon using info supplied in arrayInfo
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also cObjType_IconArray, ECM_ICONDRAWENTRY

WM_CONTROL Messages

WM_CONTROL is a group of messages which may be sent to the HWND to instruct Omnis objects to perform specialized actions. Some of the messages described are implemented as functions in Omnis, but are included here for completeness.

DESKTOP_MENU_ENABLED

Instructs Omnis to set the enabled state of the desktop switch. This is useful if the component supports functionality similar to OLE in-place activation (as Omnis OLE does), whereby, during in-place activation the desktop switch menu should be disabled to avoid the user changing the desktop mode.

Please note that the menu enabled state can be changed on the development version of Omnis only, the runtime version (which doesn't have the menu) ignores this message.

- **lParam** - qtrue if menu should be enabled, qfalse otherwise.

```
// Disable menu
WNDsendMessage( mHwnd, WM_CONTROL, DESKTOP_MENU_ENABLED, qfalse );
… Processing …
// Enable menu
WNDsendMessage( mHwnd, WM_CONTROL, DESKTOP_MENU_ENABLED, qtrue );
```

DRAW_DESIGN_NAME

Instructs Omnis to draw the objects' name. Functionally the same as ECOdrawDesignName.

- **lParam** – The HDC to draw into.

```
WNDsendMessage( mHwnd, WM_CONTROL, DRAW_DESIGN_NAME, (LPARAM)hdc );
```

See also ECOdrawDesignName

DRAW_MULTIDESIGN_KNOBS

Instructs Omnis to draw the multi-selected design knobs. Functionally the same as ECOdrawMultiKnobs.

- **lParam** – The HDC to draw into.

```
WNDsendMessage( mHwnd, WM_CONTROL, DRAW_MULTIDESIGN_KNOBS, (LPARAM)hdc );
```

See also ECOdrawMultiKnobs

DRAW_NUMBER

Instructs Omnis to draw the objects' number. Functionally the same as ECOdrawNumber.

- **lParam** – The HDC to draw into.

```
WNDsendMessage( mHwnd, WM_CONTROL, DRAW_NUMBER, (LPARAM)hdc );
```

See also ECOdrawNumber

GET_MENUHANDLE (Windows only)

Returns the operating system menu handle for the Omnis menu.

- **lParam** - Menu handle required. Currently only MM_FILE is supported.

```
HMENU menuHandle = WNDsendMessage( mHwnd, WM_CONTROL, GET_MENUHANDLE, MM_FILE );
if ( menuHandle )
{
  qshort itemCount = GetMenuItemCount((HMENU)menuHandle );
}
```

GET_OMNIS_HPALETTE (Windows only)

Returns the Omnis palette handle.

```
HPALETTE omnisPalette = WNDsendMessage( mHwnd, WM_CONTROL, GET_OMNIS_PALETTE, 0 );
HPALETTE myObjectPalette = 0;
if (omnisPalette)
{
  // Create new palette using OMNIS palette
  HLOCAL hl; LOGPALETTE* Logpal;
  hl = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, sizeof(LOGPALETTE)+(256*sizeof(PALETTEENTRY)));
  if(hl)
```

```
  {
    Logpal = (LPLOGPALETTE) GlobalLock(hl);
    GetPaletteEntries(omnisPalette,0,256, Logpal->palPalEntry);
    Logpal->palVersion = 0x300;
    Logpal->palNumEntries = 256;
    myObjectPalette = CreatePalette(Logpal);
    GlobalUnlock(hl);
    GlobalFree(hl);
  }
}
```

HAS_FOCUS

Returns true if the object has the focus. Functionally the same as ECOhasFocus.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, HAS_FOCUS, 0 );
if ( result )
{
  // object currently has the focus
}
```

See also ECOhasFocus

HIDE_TOOLTIP

Instructs Omnis to hide the on-screen tool tip if it is shown. Functionally the same as ECOhideTooltip.

```
// hides tooltip
WNDsendMessage( mHwnd, WM_CONTROL, HIDE_TOOLTIP, 0 );
```

See also ECOhideTooltip

IS_FLD_EDITABLE

Returns true if the object is editable (i.e. in runtime and not read-only).

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, IS_FLD_EDITABLE, 0 );
if ( result )
{
  // object is in edit mode
}
```

IS_IN_DESIGN

Returns true if in design mode. Functionally the same as ECOisDesign.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, IS_IN_DESIGN, 0 );
if ( result )
{
  // object is in design mode.
}
```

See also ECOisDesign

IS_MULTISELECTED

Returns true if the object is currently one of many objects selected. Functionally the same as ECOisMultiSelected.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, IS_MULTISELECTED, 0 );
if ( result )
{
  // object is multi-selected.
}
```

See also ECOisMultiSelected

IS_OMNIS_IN_BUILDMODE

Returns qtrue if Omnis is currently in **build mode**. **Build mode** is the state when Omnis is debugging an Omnis method. During this state, components should not execute events (ECOsendEvent).

```
if ( WNDsendMessage( mHwnd, WM_CONTROL, IS_OMNIS_IN_BUILDMODE, 0 )==0 )
{
  // send my event
}
```

See also ECOisOMNISinTrueRuntime

IS_SELECTED

Returns true if the object is currently selected. Functionally the same as ECOisSelected.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, IS_SELECTED, 0 );
if ( result )
{
  // object is selected.
}
```

See also ECOisSelected

IS_SERIALISED (v3.1)

Asks Omnis if the component has been serialised and returns information about the serial number.

```
EXTserialise serInfo;
serInfo.mProductCode = str15("XXXX");
// mProductCode = first four alpha/numeric chars of serial number
qbool result = (qbool)WNDsendMessage( mHwnd, WM_CONTROL, IS_SERIALISED, (LAPARAM)&serInfo);
if ( result )
{
  // component has been serialised.
  // on return
  //   serInfo.mFunctionCode contains codes for enabled functions
  //   serInfo.mSerial contains the complete serial number
  //   serInfo.mNotes contains the serial number notes
}
```

See also ECOisSerialised, EXTserialise

IS_SETUP

Allows the component to inquire on the set-up state of the object. The set-up state of an object is false before properties have been initialized, true afterwards. Functionally the same as ECOisSetup.

```
qbool result = (qbool)WNDsendMessage( mHwnd, WM_CONTROL, IS_SETUP, 0);
if ( result )
{
  // object is setup and ready for action.
}
```

See also ECOisSetup

IS_SHOWNUMBER

Returns true if the object is in design-mode and 'Show number' is true. Functionally the same as ECOisShowNumber.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, IS_SHOWNUMBER, 0 );
if ( result )
{
  // Show number is on.
}
```

See also ECOisShowNumber

IS_WINDOW_TOP

Returns true if the object is a member of the top-most window. Functionally the same as ECOisWndTop.

```
qbool result = (qbool)WNDsendMessage( mHwnd, WM_CONTROL, IS_WINDOW_TOP, 0 );
if ( result )
{
  // object is at top
}
```

See also ECOisWndTop

LIST_SETLINEHEIGHT

Informs Omnis of a new line height for cObjType_List objects. Functionally the same as ECOlistSetLineHeight.

- **lParam** - qlong which represents the new line height for the list.

```
// Forces all lists lines in a derived picture component to be 50 pixels high.
WNDsendMessage( mHwnd, WM_CONTROL, LIST_SETLINEHEIGHT, 50 );
```

See also ECOlistSetLineHeight

OMNIS_IN_BACKGROUND

Returns true if the Omnis is currently a background application.

```
qlong result = WNDsendMessage( mHwnd, WM_CONTROL, OMNIS_IN_BACKGROUND, 0 );
if ( result==0 )
{
  // OMNIS is the foremost application
}
```

PICTURE_ERASEBKGROUND

Instructs the cObjType_Picture object to erase the background.

```
WNDsendMessage( mHwnd, WM_CONTROL, PICTURE_ERASEBKGROUND, 0 );
```

See also cObjType_Picture

PICTURE_UPDSCROLLRANGE

Instructs the cObjType_Picture object to recalculate the scroll range for the object. On receipt of this message, Omnis sends the component the ECM_GETPICTUREDIM message.

```
WNDsendMessage( mHwnd, WM_CONTROL, PICTURE_UPDSCROLLRANGE, 0 );
```

See also ECM_GETPICTUREDIM

RESET_METHOD_FLAGS

Instructs Omnis to reset all method flags. Omnis sends the component repeated ECM_NEWMETHODFLAGS for each method in the object.

```
WNDsendMessage( mHwnd, WM_CONTROL, RESET_METHOD_FLAGS, 0 );
```

See also ECM_NEWMETHODFLAGS

RESET_PROPERTY_FLAGS

Instructs Omnis to reset all property flags. Omnis sends the component repeated ECM_NEWPROPERTYFLAGS for each property in the object.

```
WNDsendMessage( mHwnd, WM_CONTROL, RESET_PROPERTY_FLAGS, 0 );
```

See also ECM_NEWPROPERTYFLAGS

SET_EDITMENU

Instructs Omnis to rebuild the edit menu.

```
WNDsendMessage( mHwnd, WM_CONTROL, SET_EDITMENU, 0 );
```

SET_PALETTE

Instructs Omnis that the objects' palette has altered. Functionally the same as GDIsetPalette.

- **lParam** - HPALETTE handle of the new palette.

```
WNDsendMessage( mHwnd, WM_CONTROL, SET_PALETTE, (LPARAM)myPalette );
```

See also GDIsetPalette

SET_STATUSBAR_TEXT

Updates the Omnis status bar with the specified text.

- **lParam** - Pointer to null terminated string.

```
str255 newStatusBarMsg = str255( "Text to go into the status bar" );
WNDsendMessage(mHwnd,WM_CONTROL,SET_STATUSBAR_TEXT,(LPARAM)newStatusBarMsg.cString());
```

SET_TOOLGRPS_VISIBLE

Instructs Omnis to set the visibility state of all desktop toolbars. This is useful if the component supports functionality similar to OLE in-place activation (as Omnis OLE does), whereby, during in-place activation, all Omnis toolbars should be removed to avoid confusion between Omnis and the activated application.

- **lParam** - qtrue if toolbars are visible, qfalse otherwise.

```
// Hide Toolbars
WNDsendMessage( mHwnd, WM_CONTROL, SET_TOOLGRPS_VISIBLE, qfalse );
… Processing …
// Show Toolbars
WNDsendMessage( mHwnd, WM_CONTROL, SET_TOOLGRPS_VISIBLE, qtrue );
```

SET_WINDOWS_VISIBLE

Instructs Omnis to set the visibility state of all windows, except the window which contains the external component. This is useful if the component supports functionality similar to OLE in-place activation (as Omnis OLE does), whereby, during in-place activation, all Omnis windows should be removed to avoid confusion between Omnis and the activated application.

- **lParam** - qtrue if windows are visible, qfalse otherwise.

```
// Hide Windows
WNDsendMessage( mHwnd, WM_CONTROL, SET_WINDOWS_VISIBLE, qfalse );
… Processing …
// Show Windows
WNDsendMessage( mHwnd, WM_CONTROL, SET_WINDOWS_VISIBLE, qtrue );
```

SETNOERASEFORPICTURES

This can only be used when deriving from an Omnis picture field (cObjType_Picture). This message instructs Omnis not to erase the picture field's client area when data changes. This gives you more control if, for example, you want to fade an image over the previous image. lParam is used to indicate if the erase should happen or not.

```
// disables erasing
WNDsendMessage( mHwnd, WM_CONTROL, SETNOERASEFORPICTURES, qtrue );
// enables erasing
WNDsendMessage( mHwnd, WM_CONTROL, SETNOERASEFORPICTURES, qfalse );
```

See also cObjType_Picture

UPDATE_PROPINSPECTOR

Instructs Omnis to update the Property Manager. Functionally the same as ECOupdatePropInsp.

- **lParam** - qlong which represents the property to update. Zero updates all properties.

```
// Update all properties
WNDsendMessage( mHwnd, WM_CONTROL, UPDATE_PROPINSPECTOR, 0 );
// Update myPropId
WNDsendMessage( mHwnd, WM_CONTROL, UPDATE_PROPINSPECTOR, myPropId );
```

See also ECOupdatePropInsp

General Functions

ECOaddParam()

| | |———————————————————————————| | EXTParamInfo* ECOaddParam(EXTCompInfo* pEci, EXTfldval* pFval, qlong pParamId = 0, qshort pParamType = 0, qlong pParamFlags = 0, qchar pParamNum=0, qlong pParamParent = 0) |

The ECOaddParam function adds a new parameter to EXTCompInfo structure allowing you to pass information to/from Omnis.

Normally a component calls this function passing only the pEci and pFval pointers. It should be noted that after ECOaddParam has been called the data contents (memory) of pFval belong to another object inside Omnis, so the deletion of the pFval causes no memory to be deleted.

pFval data belongs to Omnis and may be deleted in the component.

- **pEci** - Specifies the pointer to the EXTCompInfo structure.
- **pFval -** Specifies the pointer to the parameter data.
- **pParamId -** Specifies the id of this parameter. The default value of 0 indicates a returned parameter.
- **pParamType -** Specifies the parameter data type.
- **pParamFlags -** Specifies the parameter flags.
- **pParamNum -** Specifies the parameter number.
- **pParamParent -** Specifies the parameters' parent id.
- **returns** - Returns a pointer to the EXTParamInfo structure which contains the parameter.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_CONSTPREFIX:
    {
      EXTfldval prefixName;
      str15 prefixStr;
      prefixStr[0] = RESloadString( gInstLib, resourceID, &prefixStr[0], 15 );
      prefixName.setChar(prefixStr);
      ECOaddParam(eci,&prefixName);
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECOaddTraceLine()

void ECOaddTraceLine(str255* pString) void ECOaddTraceLine(str255 *pStr1, str255 *pStr2)

The ECOaddTraceLine function enables the component to add strings to the Omnis trace log.

- **pString** - The pointer to the str255 class which contains the string to be written to trace log column 2. When called with two parameters pStr1 populates column 1 & pStr2 populates column 2.

```
str255 myTraceLine("Some trace information");
ECOaddTraceLine( &myTraceLine );
```

ECOallowDefaultContextMenu()

qbool ECOallowDefaultContextMenu(HWND pHOmnisCompHwnd)

Sends an ALLOW_DEFAULT_CONTEXT_MENU messageto the specified HWND, (re-)enabling its default context menu.

- **pHOmnisCompHwnd** – The external component's window handle.

ECOBprocessRemoteFormDesignMessage()

void ECOBprocessRemoteFormDesignMessage(qlong pOmnisUnq, EXTfldval& pMessageListFval)

Called to process a remote form design message sent via the client obrowser web socket

- **pOmnisUnq** - mUnique of remote form editor
- **pMessageListFval** - EXTfldval containing the row representing the message

ECOBsendPropertyChange()

void ECOBsendPropertyChange(HWND pHwnd, attnumber pAnum, EXTfldval& pPropValue, qlong pMultiValueItem)

Called to send a property change to the obrowser rendering design mode - used when the property change originates in the C++ control e.g. when dragging data grid column separator in the data grid heading

- **pHwnd** - The HWND of the external component
- **pAnum** - The anum (external component value) of the property
- **pPropValue** - The new value of the property
- **pMultiValueItem** - If non-zero, this is a multi value item and this number is the 1-based index to the item e.g. $currentcolumn

```
qlong sizingColumn = setColumnHeaderCursor();
propValue.setLong(convertColumnWidthPropertyFromPixels(newWidth));
ECOBsendPropertyChange(hwnd(), anumColumnwidth, propValue, sizingColumn);
```

ECOBsetCurrentDesignItem()

```
void ECOBsetCurrentDesignItem (HWND pHwnd, rstrno pPropertyResource, EXTfldval& pValue)
```

Called to set the current design item property in the client obrowser being used to render a remote form.

- **pHwnd** - The HWND of the external component
- **pPropertyResource** - The resource number the current design item property
- **pValue** - The new value of the current design item property

ECOBsetScrollAmt()

```
void ECOBsetScrollAmt(HWND pHwnd, qbool pHorz, qshort pScrollAmt)
```

Sends a SET_SROLL_AMT message to the specified remote form HWND.

- **pHWnd** - The HWND of the external component
- **pHorz** – qtrue for horizontal scrolling
- **pScrollAmt** – The scroll size in pixels.

ECOBsetScrollInfo()

```
void ECOBsetScrollInfo(HWND pHwnd, qlong pScrollMax, qlong pScrollPos)
```

Called to set the design scroll info for an object in the client obrowser being used to render a remote form.

- **pHwnd** - The HWND of the external component
- **pScrollMax** - The maximum scroll amount required in pixels
- **pScrollPos** - The current scroll position (0 - pScrollMax)

ECOcanSendEvent() (web client only)

```
qbool ECOcanSendEvent( HWND pHwnd, qlong pEventID)
```

Use ECOcanSendEvent to test if an event can be send now.

- **pHwnd** - The HWND of the object.
- **pEventID** - The id of the event.
- **returns** - Returns true if the event can be send now. If this function returns false and the event must be send, the component should delay the sending by using a timer and checking again later.

See also ECOsendEvent

ECOclipboardGetPicture()

qbool ECOclipboardGetPicture(qHandle& pPicture)

This function retrieves picture data from the clipboard.

- pHandle – (output) the handle containing the clipboard data
- **returns** – true if the clipboard contained picture data.

See also ECOclipboardHasFormat, ECOclipboardSetPicture, ECOclipboardSetText, ECOclipboardGetText.

ECOclipboardGetPictureEx()

qbool ECOclipboardGetPictureEx(qHandle& pPicture)

This function retrieves a picture from the clipboard; with alpha support.

- pHandle – (output) the handle containing the clipboard data
- **returns** – true if the clipboard contained picture data.

```
// Paste from clipboard- excerpt from icon edit component
qHandle han;
if (ECOclipboardGetPictureEx(han) && han)
{
  qHandlePtr hp(han,0);
  qlong w1 = hp.dataLen();
  if ( w1>0 )
  {
    mPastePixMap = GDIHPIXMAPfromSharedPicture(*hp, w1);
    if ( mPastePixMap )
    {
      HPIXMAPinfo pixInfo; GDIgetHPIXMAPinfo( mPastePixMap, &pixInfo );
      mPastePixMap = convTo24(pixInfo,mPastePixMap);
      #ifdef ismacosx
        qbool isAlpha = qbool((**mPastePixMap).pixelFormat == k32RGBAPixelFormat);
      #endif
    }
  }
}
```

ECOclipboardGetText()

qbool ECOclipboardGetText(qHandle& pText)

This function retrieves text data from the clipboard.

- **pText** – reference to a qHandle.
- **returns** – true if the clipboard contained text data.

See also ECOclipboardHasFormat, ECOclipboardSetText, ECOclipboardGetPicture, ECOclipboardSetPicture

ECOclipboardHasFormat()

```
qbool ECOclipboardHasFormat(EXTclipType pType)
```

Use this function to check if the clipboard contains data of the specified type.

- **pType** – enum, one of the following
- **eExtClipText** – test the clipboard for text data
- **eExtClipPicture** – test the clipboard for picture data
- **returns** – true if the clipboard contains data of the specified type

See also EXTclipType, ECOclipboardGetPicture, ECOclipboardGetText

ECOclipboardSetPicture()

```
qbool ECOclipboardSetPicture(qHandle pPicture)
```

This function places the given data as a picture on the clipboard.

- **pText** – the picture data.
- **returns** – true if the call was successful.

See also ECOclipboardGetPicture , ECOclipboardGetText, ECOclipboardSetText

ECOclipboardSetText()

```
qbool ECOclipboardSetText(qHandle pText)
```

This function places the given data as text on the clipboard.

- **pText** – the text data.
- **returns** – true if the call was successful.

See also ECOclipboardGetText, ECOclipboardGetPicture, ECOclipboardSetPicture

ECOconvertHFSToPosix()

```
qlong ECOconvertHFSToPosix( strxxx& pSrcPath, strxxx& pDstPath )
```

Converts the supplied Mactintosh file/folder path from Hierarchical File System format (colon separators) to Posix format (forward slash separators).

- **pSrcPath** – a strxxx object containing the HFS formatted path string.
- **pDestPath** – a strxxx object which receives the Posix formatted path string.

ECOconvertPosixToHFS()

qlong ECOconvertPosixToHFS(qbyte *pSrcPath, CFStringEncoding pSrcEncoding, strxxx& pDstPath)

Converts the supplied Mactintosh file/folder path from Posix format (forward slash separators) to Hierarchical File System format (colon separators).

- **pSrcPath** – a buffer containing the null-terminated Posix formatted path string.
- **pSrcEncoding** – A constant describing the Unicode encoding of the source string.
- **pDestPath** – a strxxx object which receives the HFS formatted path string.

```
OpsErr err; EXTfldval srcpath; str255 sdstPath;
err = ECOconvertPosixToHFS(srcpath.getChar().cString(), kCFStringEncodingMacRoman, sdstPath);
```

ECOconvKnownJavaObjs()

qbool ECOconvKnownJavaObjs(tqappfile* pLib, qlong &pFlag)

Returns the object's behavior with regard to Java object types. (Used internally by the Java objects component). The value if pFlag after the call indicates the behavior:

- **pFlag –** (output) qfalse => traditional behaviour object references are returned, qtrue => known objects are converted to Omnis types.

```
tqappfile *app = ECOgetApp(pEci->mLocLocp);
qbool mConvKnownObjects;
if(app) ECOconvKnownJavaObjs(app, mConvKnownObjects);
```

ECOcreateThread()

qbool ECOcreateThread(EXTthreadHandle &pHandle, void *pAttribs, EXTstartRoutine pThreadProc, void *pParams)

Creates a new process thread and executes its start routine.

- pHandle – Receives the newly created thread handle on success.
- pAttribs – Atrributes can affect the thread behavior (ignored on Windows).
- pThreadProc – The address of the thread's start routine.
- pParams – Abstract pointer to one or more parameters to be passed to the threadProc.

```
void** _params = new void*[3];
_params[0] = this;
_params[1] = pEci;
_params[2] = database->mLogonConfig;
gSemaphoreThreadRunning = ECOcreateThread(mSemThreadHandle, NULL, (EXTstartRoutine)semaphoreProc, _params);

EXTPROC semaphoreProc(LPVOID lpParameter)
{…
}
```

ECOdecrypt()

```
void OMNISAPI ECOdecrypt(qbyte *pDataToDecrypt, qlong pDataToDecryptLen, qulong *pKey, EXTfldval &pDecryptedData)
```

Decrypts binary data previously encrypted using ECOencrypt().

- **pDataToDecrypt** – Pointer to the encrypted data.
- **pDataToDecryptLen** – Data length in bytes.
- **pKey** – The encryption key. Must address an array of dimension 4 of unsigned longs.
- **pDecryptedData** – Receives the decrypted data.

ECOdestroyCriticalSection()

```
qbool ECOdestroyCriticalSection(EXTcriticalSection *cs)
```

Destroys a critical section object created previously using ECOcreateCriticalSection(). The return value can be ignored.

- **cs** – Pointer to an existing EXTcriticalSection object.

ECOdoFind()

```
void ECOdoFind(str255 &pFindString, str255 &pStringToSearch, lsttype *pResultList)
```

Locates and returns a string in the components resources. Called in response to the ECM_FINDSTRINGS message.

```
case ECM_FINDSTRINGS: // search resources
{
  if (gInstLib)
  {
    str255 *findString = (str255 *) wParam;
    lsttype *resultList = (lsttype *) lParam;
    str255 resString;
    for (rstrno resNum = gFirstFindString; resNum <= gLastFindString; ++resNum)
    {
      RESloadString(gInstLib, resNum, resString);
      if (resString.length())
        ECOdoFind(*findString, resString, resultList);
    }
  }
  return 0;
}
```

ECOdoMethod()

```
qret ECOdoMethod(qobjinst pObjInst,strxxx* pMethod, EXTfldval* pParams = 0, qshort pParamCnt = 0,qbool pExecNow=qtrue,
EXTfldval *pReturnValue = 0)
```

Where an Omnis object is superclassed with a non-visual external component, the ECOdoMethod function can be used to invoke a method inside the object class. For example, if an email object has a method called '$newmail' then a component may wish to use ECOdoMethod to inform Omnis of new mail.

This function is basically a wrapper for ECOdoMethodECI.

- **pObjInst** - Pointer which was originally generated by Omnis and passed to the external's WNDPROC during ECM_OBJCONSTRUCT.
- **pMethod** – A strxxx object containing the name of the method to execute.
- **pParams** - Pointer to an array of EXTfldval which contain the parameters for the method.
- **pParamCnt** - Number of parameters for the method.
- **pExecNow** - True if the method should be processed by Omnis immediately, false otherwise.
- **pReturnValue** – Allows the Omnis method to pass a return value back to the component via the *Quit method* command.
- **returns** - Returns a qret data type containing the result.

```
// Inform sub-classed email object of new email
EXTfldval numOfEmail; str255 methodName("$newemail")
numOfEmail.setLong( number_of_new_emails );
ECOdoMethod( mObjInst, &methodName, &numOfEmail, 1 );
```

See also ECOdoMethodECI

ECOdoMethodECI()

qbool ECOdoMethodECI(qobjinst pObjInst, strxxx* pMethod, EXTCompInfo* pEci, qbool pExecNow=qtrue)

Where an Omnis object is superclassed with a non-visual external component, the ECOdoMethod function can be used to invoke a method inside the object class. For example, if an email object has a method called '$newmail' then a component may wish to use ECOdoMethod to inform Omnis of new mail.

Most components use ECOdoMethod in preference to this function.

- **pObjInst** - Pointer which was originally generated by Omnis and passed to the external during ECM_OBJCONSTRUCT.
- **pMethod** – A strxxx object containing the name of the method to execute.
- **pEci** - The EXTCompInfo structure which contains the method parameters.
- **pExecNow** - True if the method should be processed by Omnis immediately, false otherwise.
- **returns** - Returns a qret data type containing the result.

```
// Email event occurred. Invoke OMNIS objects' method
EXTCompInfo* eci = new EXTCompInfo();
eci->mParamFirst = 0;
// Add parameters to EXTCompInfo structure
EXTfldval myParam1;
myParam1.setlong( someData );
// Add parameter 1
ECOaddParam(eci,&myParam1,0,0,0,1,0);
// Invoke method
str255 methodName("$newemail")
qbool eventOk = ECOdoMethodECI( mObjInst, &methodName,eci, qtrue );
// Delete parameters from EXTCompInfo structure
ECOmemoryDeletion( eci );
// Delete eci structure
delete eci;
```

See also ECOdoMethod

ECOdrawDesignName()

qbool ECOdrawDesignName(HWND pHWnd, HDC pHDC)

Allows the component to draw the name in the specified device context. Will have no effect if the object is not in design mode.

- **pHWnd** - The HWND of the object.
- **pHDC** – The device context to draw into.

```
ECOdrawDesignName( mHwnd, hdc );
```

See also DRAW_DESIGN_NAME

ECOdrawMultiKnobs()

void ECOdrawMultiKnobs(HWND pHWnd, HDC pHDC)

Allows the component to draw the multi-select knobs in the specified device context. Will have no effect if only one object is selected or if the object is not selected.

- **pHWnd** - The HWND of the object.
- **pHDC** – The device context to draw into.

```
ECOdrawMultiKnobs( mHwnd, hdc );
```

See also DRAW_MULTIDESIGN_KNOBS

ECOdrawNumber()

qbool ECOdrawNumber(HWND pHWnd, HDC pHDC)

Allows the component to draw the number in the specified device context. Will have no effect if 'Show number' is not active.

- **pHWnd** - The HWND of the object.
- **pHDC** – The device context to draw into.

```
ECOdrawNumber( mHwnd, hdc );
```

See also DRAW_NUMBER

ECOdrawNumberEx()

void ECOdrawNumberEx(HWND pHWnd, HDC pHDC, qbool pEraseArea)

Like ECOdrawNumber, except the background is optionally erased and framed.

- **pHWnd** - The HWND of the object.
- **pHDC** – The device context to draw into.
- **pEraseArea** – Set to qtrue to erase the background.

ECOencrypt()

```
void OMNISAPI ECOencrypt(qbyte *pDataToEncrypt, qlong pDataToEncryptLen, qulong *pKey, EXTfldval &pEncryptedData)
```

Encrypts a block of binary data.

- **pDataToEncrypt** – Pointer to the data to be encrypted.
- **pDataToEncryptLen** – Length of the data block in bytes.
- **pKey** – The encryption key. Must address an array of dimension 4 of unsigned longs.
- **pEncryptedData** – Receives the encrypted data.

ECOenterCriticalSection()

```
qbool ECOenterCriticalSection(EXTcriticalSection *cs)
```

Begins a critical section; a platform-independent section designed to be enterable by a single thread. Exit the critical section using ECOleaveCriticalSection().

- **cs** – Pointer to an ExtCriticalSection initialized previously using ECOinitCriticalSection().

ECOexcludeToolTipRect()

```
void ECOexcludeToolTipRect( HWND pHWnd, HDC pHDC )
```

Allows the component to exclude the tool-tip rectangle from the device contexts' clipped drawing area.

- **pHWnd** - The HWND of the object.
- **pHDC** – The device context to exclude the tool-tip rectangle from.

See also ECOgetToolTipRect

ECOfindObject()

```
void* ECOfindObject( HINSTANCE pInstance, HWND pHWnd, WPARAM pWParam =0 )
```

Locates a pointer which has previously been stored via the ECOinsertObject function.

- **pInstance** - The Omnis instance. This may be NULL which results in the function searching all Omnis instances for the HWND.
- **pHWnd** - The HWND being searched for.
- **pWParam** - ***Background components only.*** The WPARAM which was passed in from Omnis, this should be passed for background components only.
- **returns** - Returns the pointer previously stored via the call to ECOinsertObject.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case WM_PAINT:
    {
      cObj* object = (cObj *)ECOfindObject(eci->mOmnisInstance, hwnd );
      if ( NULL!=object && object->paint() ) return qtrue;
        break;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOinsertObject

ECOfindNVObject()

void* ECOfindNVObject(HINSTANCE pInstance, LPARAM pInstPtr)

Locates a pointer which has previously been stored via the ECOinsertNVObject function.

- **pInstance** - The Omnis instance. This may be NULL which results in the function searching all Omnis instances for the HWND.
- **pInstPtr** – The unique object instance reference (as allocated by Omnis)
- **returns** - Returns the pointer previously stored via the call to ECOinsertNVObject.

See also ECOinsertNVObject, Non-visual components

ECOfindParamNum()

EXTParamInfo* ECOfindParamNum(EXTCompInfo* pEci, qlong pParamID)

Locates a parameter in the EXTCompInfo structure. This function should be used to locate method and property parameters.

- **pEci** - The pointer to the EXTCompInfo structure.
- **pParamID** - The id of the parameter to be located.
- **returns** - Returns the pointer to the EXTParamInfo structure if successful, NULL otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_METHODCALL:
    {
      // OMNIS code is calling your component method
      qlong methodID = ECOgetId(eci);
      switch(methodID)
      {
        case cMyMethod1:
        {
```

```
          EXTParamInfo* param1 = ECOfindParamNum( eci, 1);
          EXTParamInfo* param2 = ECOfindParamNum( eci, 2);
          if ( param1 && param2 )
          {
            // .. Do method processing …
          }
          return 1L;
        }
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECOfindString()

---

void ECOfindString(str255 &pFindString, str255 &pStringToSearch, lsttype *pResultList)

Accesses the Omnis string table editor and searches for pFindString inside pStringToSearch at the current find location. If found, a row is added to pResultList containing the current find location and pStringToSearch.

- **pFindString** – The string to search for.
- **pStringToSearch** – The string to be searched.
- **pResultList** – The out list which is appended with the search result.

ECOformatJSON()

---

void ECOformatJSON(EXTfldval &pJSONData, EXTfldval &pReturnedFormattedJSON)

Parses the JSON in pJSONData and returns a formatted representation. Used by OJSON.$formatjson().

- **pJSONData** – EXTfldval containing the source data
- **pReturnedFormattedJSON** – EXTfldval that receives the formatted data.

ECOfromUChar()

---

qlong ECOfromUChar(ECOconverter pConverter, char *pTarget, qlong pTargetLen, UChar *pSource, qlong pSourceLen, qlong *pSizeNeeded, qlong *pOutputLen)

Converts from UChar (UTF-16) to a code page identified by pConverter. ECOfromUChar will either convert all data or return the size required if the target buffer is not big enough.

- pConverter - The converter for the destination code page.
- pTarget - The address of the destination buffer.
- pTargetLen - Length of pTarget IN BYTES.
- pSource - The address of the source buffer.
- pSourceLen - Length of the source buffer in UChar units.

- pSizeNeeded - if the available buffer space is too small,this receives the size needed for the target buffer.
- pOutputLen - If the conversion is successful this receives the length of the output data in bytes.

Returns non-zero if an error occurred, zero if conversion was successful; if return value is -1 then *pSizeNeeded is the required buffer size in bytes.

ECOgetApp()

qapp ECOgetApp(locptype* pLocp)

Returns a reference to an Omnis application. The EXTCompInfo structure which is passed to external components contains two context pointers. The context pointer mInstLocp points to the context of the class instance which contains the component. The context pointer mLocLocp points to the context of the calling method.

- **pLocp** - The context pointer.
- **return** - The Omnis library reference.

```
// fetch the library reference which contains the instance of the component
qapp app = ECOgetApp( pEci->mInstLocp );
```

ECOgetBundleRef() Mac OSX only

void *ECOgetBundleRef(qlong pBundleID)

Returns a CFBundleRef dependant on the pBundleID.

- **pBundleID** - Should be either kXsocket or kCoreGraphics.

ECOgetCoch()

inline qchar ECOgetCoch()

Convenience function that returns the result of ECOreadSeparatorItem(2); the numeric thousands separator character.

ECOgetConstantDescriptionByValue()

void ECOgetConstantDescriptionByValue(qlong pValue, rstrno pFirstConstant, rstrno pLastConstant, EXTfldval &pDescription)

Get the description from the constant resource with the specified value, in the specified resource range. Each constant resource has the syntax

`<name>:<integer value>:<string value>:<description>`

- **pValue** – integer value to search for.
- **pFirstConstant** – start resource number.
- **pLastConstant** – end resource number.
- **pDescription** – Receives the description of the specified resource.

ECOgetCrbFieldInfo()

qbool ECOgetCrbFieldInfo(strxxx& pFieldName, locptype* pLocp, crbFieldInfo& pFInfo)

ECOgetCrbFieldInfo gets the specified fields full format information. See structure crbFieldInfo for full description of the information returned.

- **pFieldName** - The Omnis variable
- **pLocp** - The context pointer.
- **pFInfo** - Pointer the info structure
- **return** - Returns true if the Omnis variable was found.

```
crbFieldInfo info;
str255 fieldName("ivTheVariable");
if ( ECOgetCrbFieldInfo( fieldName, eci->mInstLocp, &info ) )
{
  qlong maxLen = info.fln;
}
```

See also struct crbFieldInfo in EXTfldval class reference

ECOgetDeviceParms()

PRIdestParmStruct* ECOgetDeviceParms(locptype* pLocp)

Returns a reference to the global device parameters structure. It is not a copy, and altering any values in the structure will effect the Omnis devices.

- **pLocp** - The context pointer. Currently not used.
- **return** - Points to Omnis device parameters.

```
// fetch a pointer to the global device parameters
PRIdestParmStruct *deviceParms = ECOgetDeviceParms( pEci->mInstLocp );
```

ECOgetDirectoryDialog()

qbool ECOgetDirectoryDialog(HINSTANCE pInstance, HWND pOwner, qlong pTitle, str255& pDirName, strxxx* pInitDir = 0)qbool ECOgetDirectoryDialog(HINSTANCE pInstance, HWND pOwner, strxxx& pTitle, str255& pDirName, strxxx* pInitDir = 0)

The ECOgetDirectoryDialog function enables the component to invoke a dialog to request a directory.

- **pInstance** - The instance which contains the string resources required. This would normally be gInstLib.
- **pOwner** - The HWND of the owner.
- **pTitle** - The resource id for the title OR a str255 object containing the title.
- **pDirName** - The str255 object which contains the directory name upon return, if successful.
- **pInitDir** - The pointer to the str255 object which specifies the initial directory. May be NULL.
- **returns** - Returns true if a directory has been selected, false otherwise.

Note: On MacOS make sure the component project contains the OMNISLIB.RSRC file.

```
str255 newDirectory;
if ( ECOgetDirectoryDialog( gInstLib,hwnd,5000,5001,newDirectory ) )
{
  … processing …
}
```

ECOgetDpCh()

inline qchar ECOgetDpCh()

Convenience function that returns the result of ECOreadSeparatorItem(1); the numeric decimal separator character.

ECOgetFirstCharFromJSON()

qchar ECOgetFirstCharFromJSON(EXTfldval &pJSONData)

ECOgetFont()

void ECOgetFont(HWND pHwnd , qfnt* pFnt, qshort pFntIndex, qshort pFntSize)

The ECOgetFont function enables the component to obtain font details for the given index and font size.

- **pHwnd** - The HWND of the object.
- **pFnt** - Pointer to the qfnt structure which is populated, if successful, by Omnis.
- **pFntIndex** - The index of the font required.
- **pFntSize** - The size of the font required.

```
// Create font from index & size (extract from CALENDAR example)
qfnt fnt = fntSmallFnt;
ECOgetFont( mHWnd, &fnt, mHeadingFont, mHeadingFontSize );
HFONT font = GDIcreateFont( &fnt, mHeadingBold ? styBold : styPlain );
… processing ..
GDIdeleteObject( font );
```

ECOgetFont()

void ECOgetFont(qapp pApp, qbool pReportFont, qfnt* pFnt, qshort pFntIndex, qshort pFntSize)

The ECOgetFont function enables the component to obtain font details for the given index and font size from the specified Omnis library. It also allows you to specify if you require a report font or windows font.

- **pApp** - Reference to the Omnis library. See ECOgetApp().
- **pReportFont** - Specify qtrue if you require a font from the libraries report font table.
- **pFnt** - Pointer to the qfnt structure which is populated, if successful, by Omnis.
- **pFntIndex** - The index of the font required.

- **pFntSize** - The size of the font required.

```
// sample function retrieves a report font from the library containing the
// instance of the external component.
HFONT myCreateFont( EXTCompInfo* pEci )
{
  qfnt fnt; qapp app = ECOgetApp( pEci->mInstLocp );
  ECOgetFont( app, qtrue, &fnt, 1, 12 );
  return GDIcreateFont( &fnt, styPlain );
}
```

ECOgetFontDPI()

qdim ECOgetFontDpi(HWND pHOmnisCompHwnd)

Get the DPI with which fonts are to be created (only relevant for controls on report design windows - returns zero otherwise).

- **pHOmnisCompHwnd** – The external component's HWND
- **Returns** – DPI value.

ECOgetFontIndex()

qshort ECOgetFontIndex(HWND pHwnd, EXTfldval& pFVal)

The ECOgetFontIndex function returns a font index from the specified font name.

- **pHwnd** – The HWND of the component control.
- **pFVal** – Specifies the EXTfldval which contains the font name in character format.
- **Returns** – Returns a font index from 1 to 31 if succeeded, 0 otherwise.

```
str80 s("Times Roman");
EXTfldval fval; fval.setChar( s );
qshort fntIndex = ECOgetFontIndex( hwnd, fval );
```

ECOgetId()

qlong ECOgetId(EXTCompInfo* pEci)

The ECOgetId function should be used to retrieve the id of the method or property.

- **pEci** - The pointer to the EXTCompInfo structure.
- **returns** - Returns the id of the method or property if successful, zero otherwise.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case ECM_METHODCALL:
    {
```

```
      // OMNIS code is calling your component method
      qlong methodID = ECOgetId(eci);
      switch(methodID)
      {
        // … Method 1
        case cMyMethod1:
        // … Method 2
        case cMyMethod2:
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECOgetFsCh()

inline qchar ECOgetFsCh()

Convenience function that returns the result of ECOreadSeparatorItem(3); the function parameter separator character.

ECOgetJSNativeDisplay()

qshort ECOgetJSNativeDisplay(HWND pHwnd)

For JavaScript Client remote forms.

- **pHwnd** – the Remote form window handle.
- **Returns** - the appearance type for the specified HWND, either kAppearanceiOS or kAppearanceAndroid.

ECOgetLocalIpAddress()

qulong ECOgetLocalIpAddress(void)

Returns the client machine's ethernet IP address as a hexadecimal long integer.

ECOgetOmnisErrorText()

void ECOgetOmnisErrorText(qret pErrorCode, EXTfldval &pErrorText)

Reads the Omnis error text resource string for the specified error code.

- **pErrorCode** –
- **pErrorText** – Receives the error text value.

```
EXTfldval errorText;
if (pErrorCode != e_ok)
  ECOgetOmnisErrorText(pErrorCode, errorText);
str255 errorInfo;
errorText.getChar(errorInfo);
```

ECOgetNVObject()

void *ECOgetNVObject(objectinst *pInst)

Searches for an external component instance in the chain of super instances of this object, returning the first instance found. If no external component instance is found, pInst is returned.

- **pInst** – The initial object instance.

```
EXTfldval fval; fftttype ftype1;
//….code excerpt from JavaObjs component
fval.getType(ftype1);
if (ftype1 == fftObjref)
{
  qobjinst objInst = fval.getObjRef();
  if (objInst) objInst = (qobjinst)ECOgetNVObject(objInst); // check for superinst..
  if ( objInst )
  {
    tqfJObjectContainer* object = (tqfJObjectContainer*)ECOFINDNVOBJECT(0, (LPARAM)objInst );
    if ( object && object->mObject )
    {
      EXTfldval fval1,fval2;
      ljline = ljlist->insertRow();
      ljlist->getColValRef(i,1,fval1,qtrue);
      fval1.setLong(object->mObject->mJObjID);
      ljlist->getColValRef(i,2,fval2,qtrue);
      fval2.setChar(lelemsig);
    }
  }
}
```

ECOgetOmnisVersionNumber() (v5.1.1)

qlong ECOgetOmnisVersionNumber(void)

Returns the Omnis Studio version number as a five digit long value.
For example: 5.1.1 is returned as 51100

ECOgetParamCount()

qshort ECOgetParamCount(EXTCompInfo* pEci)

The ECOgetParamCount function enables the component to inquire on how many parameters, which have ids sequentially from 1, are in the EXTCompInfo structure. This is especially useful during the ECM_METHODCALL message to ensure that the correct number of parameters have been supplied.

- **pEci** - The pointer to the EXTCompInfo structure.
- **returns** - Returns the number of parameters.

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
```

```
  {
    case ECM_METHODCALL:
    {
      // OMNIS code is calling your component method
      qlong methodID = ECOgetId(eci);
      switch(methodID)
      {
        case cMyMethod1:
        {
          if ( ECOgetParamCount(eci) != 2 )
          {
            // Error - Method needs two parameters
            return 0l;
          }
        }
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECOgetParamInfo()

qbool ECOgetParamInfo(EXTparamInfo* pParam, EXTparamTypeInfo& pInfo)

Returns additional type information about the parameter specified by pParam.

- **pParam** – Pointer to the parameter structure.
- **pInfo** – Reference to the structure which will receive the additional info.

See also EXTparamInfo, EXTparamTypeInfo

ECOgetProperty()

qbool ECOgetProperty(HWND pHwnd, qshort pAnum, EXTfldval& pFval)

The ECOgetProperty function enables the component to obtain information concerning Omnis standard object properties.

- **pHwnd** - The HWND of the object.
- **pAnum** - The anum of the property which is requested (See ANUMS.HE for the list of possible anums).
- **pFval** - The EXTfldval object which contains the property, if successful.
- **returns** - Returns true if successful, false otherwise.

```
// Get $dataname property
EXTfldval fldname;
if ( ECOgetProperty( mHwnd, anumFieldname, fldname ) )
{
  // Get the name from the fldval
  str255 str;
  fldname.getChar ( str );
}
```

ECOgetSepCh()

inline qchar ECOgetSepCh()

Convenience function that returns the result of ECOreadSeparatorItem(6); the calculation function separator & filename.fieldname separator character.

ECOgetStyle()

qbool ECOgetStyle(tqappfile* pApp, qchar* pStyleName, qshort pLen, GDItextSpecStruct* pTextSpec)

The ECOgetStyle function enables the component to obtain the field style information.

- **pApp** – The tqappfile pointer for the instance of the component.
- **pStyleName** – A pointer to the field style name.
- **pLen** – The length of the field style name.
- **pTextSpec** – A pointer to a GDItextSpecStruct which will be populated upon return.
- **returns** - Returns true if successful, false otherwise.

```
// Get the fieldstyle name
EXTfldval fval; ECOgetProperty(hwnd,anumFldStyle,fval);
str255 s; fval.getChar(s);
GDItextSpecStruct textSpec;
ECOgetStyle( app, &s[1], s[0], &textSpec );
```

ECOgetTickCount()

inline qlong ECOgetTickCount()

Returns number of 1/60sec since startup (will wrap eventually).

ECOgetToolTipRect()

qbool ECOgetToolTipRect(HWND pHwnd, qrect* pRect)

The ECOgetToolTipRect function enables the component to obtain the position of the tool tip (if visible).

- **pHwnd** - The HWND of the object.
- **pRect** – The pointer to a qrect object which will contain the tool-tip rectangle upon return (only is a tool-tip is currently visible).
- **returns** - Returns true if successful, false otherwise.

ECOgetUParamValue()

LPARAM ECOgetUParamValue(HWND pHOmnisCompHwnd)

Returns the value of mUParam; a temporary storage parameter

supplied to the external component, and stored by the last call to its WndProc().

- **pHOmnisCompHwnd** – The External Component's HWND.
- **Returns** – The value of mUParam.

ECOhasConfigurationItem()

qbool ECOhasConfigurationItem(const char *pGroupName, const char *pItemName)

Tests whether the specified configuration item exists in the specified group of the Omnis config.json file.

- **pGroupName** – Group name inside the config file.
- **pItemName** – Item name to search for inside the group.
- **Returns** – qtrue if the item exists, qfalse otherwise.

ECOhasFocus()

qbool ECOhasFocus(HWND pHWnd)

The ECOhasFocus function enables the component to inquire on the focus state of the object.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object currently has the focus, false otherwise.

```
qbool result = ECOhasFocus( mHwnd );
if ( result )
{
  // object currently has the focus
}
```

ECOhideTooltip()

void ECOhideTooltip(HWND pHwnd)

The ECOhideTooltip function can be used by the components to hide the on screen tool tip. The Omnis tool tip is drawn directly to the screen. It saves the bitmap where is it about to be displayed for later restoring when the tool tip is not needed.

As a result, if a tool tip is displayed and partly covers the control, the control paints due to a timer message for example, the bitmap saved by the tool tip that it uses for restoring could now be invalid.

To avoid this problem, controls can call this API, passing their components HWND to hide the tip.

- **pHwnd** - The HWND of the object.

ECOinitCriticalSection()

void ECOinitCriticalSection() qbool ECOinitCriticalSection(EXTcriticalSection *cs)

Intitialises a critical section object which must be done before it can be used by ECOenterCriticalSection() and ECOleaveCriticalSection(). When called without a parameter, the internal critical section; *sExtCriticalSection* is used.

- **cs** – A pointer to a static EXTcriticalSection object declared in your code.
- **Returns** – qtrue on success.

ECOinsertObject()

void ECOinsertObject(EXTCompInfo* pEci, HWND pHWnd, void* pObjPointer, WPARAM pWParam)

Stores a pointer for the specified HWND in a list of Omnis instances.

- **pInstance** - Specifies the Omnis instance to which this pointer should belong to.
- **pHWnd -** Specifies the HWND which is linked to the pointer.
- **pObjPointer -** Specifies the pointer to be stored.
- **pWParam** - ***Background components only.*** The WPARAM which was passed in from Omnis, this should be passed for background components only.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_OBJCONSTRUCT:
    {
      cObj* myNewObject = new cObj();
      if ( myNewObject )
      {
        ECOinsertObject(eci, hwnd, (void*) myNewObject);
      }
      else
      {
        // … Error - Out of memory …
      }
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECM_OBJCONSTRUCT

ECOinsertNVObject()

void ECOinsertNVObject(HINSTANCE pInstance, LPARAM pInstPtr, void* pObjPointer)

Stores a pointer for the specified HWND in a list of Omnis instances.

- **pInstance** - Specifies the Omnis instance to which this pointer should belong to.
- **pInstPtr –** Specifies the object instance pointer (as supplied by Omnis) to associate the pObjPointer with.
- **pObjPointer -** Specifies the pointer to be stored.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_OBJCONSTRUCT:
```

```
    {
      cObj* obj = new cObj();
      ECOinsertNVObject(eci->mOmnisInstance,lParam,(void*)obj);
      return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOfindNVObject, Non-visual components

ECOinvalBackObj()

void ECOinvalBackObj()

If the object is a background component, ECOinvalBackObj() invalidates the drawing area, causing it to be redrawn.

ECOisDesign()

qbool ECOisDesign(HWND pHWnd)

The ECOisDesign function enables the component to inquire on the design state of the object.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object is in design, false otherwise.

```
qbool result = ECOisDesign( mHwnd );
if ( result )
{
  // object is in design mode.
}
```

ECOisJSONArray()

qbool OMNISAPI ECOisJSONArray(EXTfldval &pFval, qlong &pArrayLength

Returns qtrue if pFval contains a single-line list that represents a JSON array. Columns should be of the same data type and should be named "___1", "___2", etc.

- **pFval** – Single line list or row stored in an EXTfldval.
- **pArrayLength** – receives the array length (column count).

ECOisJSONObject()

qbool ECOisJSONObject(EXTfldval &pFval)

Returns qtrue if pFval contains a single-line list that represents a JSON object. This will be the case if the list contains columns of different types.

- **pFval** – Single line list or row stored in an EXTfldval.

ECOisMultiSelected()

qbool ECOisMultiSelected(HWND pHWnd)

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object is currently multi-selected, false otherwise.

```
qbool result = ECOisMultiSelected( mHwnd );
if ( result )
{
  // object is selected as part of a group
}
```

See also IS_MULTISELECTED

ECOisOMNISinTrueRuntime()

qbool ECOisOMNISinTrueRuntime(HWND pHwnd)

Returns qtrue if Omnis is in a true runtime state. In this state it is safe for components to send events. In some other states it is not safe. For example, your component maybe a runtime component, but Omnis may be in **build mode** debugging another method. Omnis always tries to switch to the correct mode when executing a method/event. If you send an event during a debug session, Omnis brings your component to the front immediately, executes your event and returns to the debug session. For some controls such as a clock sending events every second, this is not what should happen.

- **pHwnd** - The HWND of the object.
- **returns** - qtrue if Omnis is in true runtime.

```
if (ECOisOMNISinTrueRuntime( mHWnd ) )
{
  // can send events
}
```

ECOisSelected()

qbool ECOisSelected(HWND pHWnd)

Allows the component to inquire on whether the object is currently selected.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object is currently selected, false otherwise.

```
qbool result = ECOisSelected( mHwnd );
if ( result )
{
  // object is selected
}
```

See also IS_SELECTED

ECOisSerialised()

qbool ECOisSerialised(HWND pHOmnisCompHwnd, qchar* pProductCode, qchar* pFunctionCode = NULL, qchar* pSerial = NULL, qchar* pNotes = NULL) qbool ECOisSerialised(qchar* pProductCode, qchar* pFunctionCode = NULL, qchar* pSerial = NULL, qchar* pNotes = NULL)

Asks Omnis if the component has been serialised and returns information about the serial number.

- **pHOmnisCompHwnd** – Components hwnd
- **pProductCode** – Product code supplied by component. Must be 4 alpha/numeric characters.
- **pFunctionCode** – Functionality code returned by Omnis. These consist of 4 alpha/numeric characters describing the enabled functionality.
- **pSerial** – Complete serial number. Returned by Omnis.
- **pNotes** – Notes as entered with the serial number by the user. Returned by Omnis.

See also IS_SERIALISED

ECOisSetup()

qbool ECOisSetup(HWND pHWnd)

Allows the component to inquire on the set-up state of the object. The set-up state of an object is false before properties have been initialized, true afterwards.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object is set-up, false otherwise.

```
qbool result = ECOisSetup( mHwnd );
if ( result )
{
  // object is setup and ready for action.
}
```

See also ECM_OBJINITIALIZE, IS_SETUP

ECOisShowNumber()

qbool ECOisShowNumber(HWND pHWnd)

Allows the component to inquire on whether the design-time option ‘Show number’ is on.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if ‘Show number’ is on, false otherwise.

```
qbool result = ECOisShowNumber( mHwnd );
if ( result )
{
  // Show number is on
}
```

See also IS_SHOWNUMBER

ECOisWndTop()

qbool ECOisWndTop(HWND pHWnd)

Allows the component to inquire on whether the object is a member of the top-most window.

- **pHWnd** - The HWND of the object.
- **returns** - Returns true if the object is a member of the top-most window, false otherwise.

```
qbool result = ECOisWndTop( mHwnd );
if ( result )
{
  // object is on top
}
```

See also IS_WINDOW_TOP

ECOleaveCriticalSection()

qbool ECOleaveCriticalSection(EXTcriticalSection *cs)

Exits a critical section; a platform-independent section designed to be enterable by a single thread. Counterpart to ECOenterCritical-Section().

- **cs** – An EXTcriticalSection object initialized previously using ECIinitCriticalSection().
- **returns** – qtrue if the code executes correctly.

ECOlistFonts()

void ECOlistFonts(EXTqlist *pList, qbool pReportFonts)

Allows the component to obtain a list of window or report fonts installed on the machine.

- **pList** - The list to populate.
- **pReportFonts** – True if a list of report fonts is required.

ECOlistJSONMembers()

void ECOlistJSONmembers(EXTfldval &pJSONList, EXTfldval &pMemberIdList)

Returns a list containing the IDs contained in a JSON object/array.

- **pJSONList** – A single-line list representing a JSON object or array.
- **pMemberIdList** – Output list containg member IDs. Row one is empty but contains the root ID as its name. Subsequent rows contain arrays of IDs.

ECOlistSetLineHeight()

void ECOlistSetLineHeight(HWND pHOmnisCompHwnd, qlong pLineHeight)

The ECOlistSetLineHeight function should be used by the component to specify the line height (in pixels) of objects which have previously been defined as cObjType_List.

- **pHOmnisCompHwnd** - The HWND of the object.
- **pLineHeight** - The list line height.

```
// Forces all lists lines in a derived picture component to be 50 pixels high.
ECOlistSetLineHeight(mHwnd,50);
```

See also WM_CONTROL - LIST_SETLINEHEIGHT, cObjType_List

ECOloadFileDialog()

qbool ECOloadFileDialog(HINSTANCE pInstance, HWND pOwner, qlong pResTitle, qlong pResFilter, str255& pFileName, str255* pInitDir = 0)qbool ECOloadFileDialog(HINSTANCE pInstance, HWND pOwner, strxxx& pTitle, strxxx& pFilter, str255& pFileName, str255* pInitDir = 0)

The ECOloadFileDialog function enables the component to invoke the operating system load file dialog.

- **pInstance** - The instance which contains the string resources required. This would normally be gInstLib.
- **pOwner** - The HWND of the owner.
- **pResTitle or pTitle** - The resource id or string for the title of the load file dialog.
- **pResFilter or pFilter** - The resource id or string for the filter string of the load file dialog. Any platform dependent filters are removed if not required. e.g.
- **pFileName** - The str255 object which contains the file name upon return, if successful.
- **pInitDir** - The pointer to the str255 object which specifies the initial folder. May be NULL.
- **returns** - Returns true if a file has been selected, false otherwise.

Note: On MacOS make sure the component project contains the OMNISLIB.RSRC file.

```
// Load file (extract from PCX example)
str255 newFile;
if ( ECOloadFileDialog( gInstLib,hwnd,5000,5001,newFile ) )
{
  object->mFile = newFile;
  object->readPCX();
  WNDinvalidateRect( hwnd, NULL );
  ECOupdatePropInsp(hwnd);
}
```

ECOlistSetLineHeight()

void ECOlistSetLineHeight(HWND pHOmnisCompHwnd, qlong pLineHeight)

Set the height used to draw list lines on the specified HWND.

- **pHOmnisCompHwnd** – The external component window handle.
- **pLineHeight** – Desired line height in pixels.

ECOlocpSetInst()

void ECOlocpSetInst(locptype *locp, objinst *objInst)

Assigns the supplied object instance pointer to the specified locptype structure (which is otherwise an abstract pointer in the component library). Used to associate an external component with a different object. The DML emulator uses this callback to register its session objects with the Omnis SQL Browser.

- **Locp** – Pointer to a component's abstract context structure normally supplied as part of the calling method's EXTCompInfo structure (pEci).
- **objInst** – Pointer to a newly created object instance.

```
str255 testStr(QTEXT("$extobjects.PGSQLDAM.$objects.PGSQLSESS.$new()"));
EXTfldval calcFld, resultFld;
qret retCode = calcFld.setCalculation(pEci->mInstLocp, ctyCalculation, &testStr[1], testStr[0]);
qbool status = calcFld.evalCalculation(resultFld, pEci->mInstLocp, NULL, qfalse);
qobjinst oinst = resultFld.getObjInst(qfalse);
LPARAM lParam = (LPARAM)oinst;
tqfDAMObjCont* object = (tqfDAMObjCont*)ECOfindNVObject(pEci->mOmnisInstance, lParam);
ECOlocpSetInst(pEci->mInstLocp, oinst);
```

ECOmapString()

qlong ECOmapString(qchar *pBuffer, qlong pBufferLen, qlong pLen)

Accesses the Omnis string table editor and searches for a string with ID matching the contents of pBuffer. If found, pBuffer is assigned the contents of the string table element and the character length is returned.

- **pBuffer** – On input- the ID of the string to match, on output- the contents of the string table element.
- **pBufferLen** – the length in bytes of the buffer (prevents overrun).
- **pLen** – the length in characters of the input ID string.

ECOmarkModified()

void ECOmarkModified(HWND pHOmnisCompHwnd, qbool pModified)

Marks the component HWND as modified. Use in design mode.

- **pHOmnisCompHwnd** – The external component window handle.
- **pModified** – Set this to indicate that the control should be redrawn.

ECOmemoryDeletion()

void ECOmemoryDeletion(EXTCompInfo* pEci)

Deletes memory previously allocated in the external component (returned parameters for example). WNDdefWindowProc processes the ECM_MEMORYDELETION message. See **ECOpushCompEvent** for an example of the use of ECOmemoryDeletion.

- **pEci** - Pointer to EXTCompInfo structure which contains the parameters to delete.

See also ECM_MEMORYDELETION

ECOmessageBox()

qbool ECOmessageBox(qulong pFlags,qbool pBell,str255& pMsg)

Provides external components with access to Omnis message box dialogs.

- **pFlags** - Determines the type of message box which can be: MSGBOX_OK, MSGBOX_YESNO, MSGBOX_NOYES, MSGBOXICON_OK, MSGBOXICON_YESNO, MSGBOXICON_NOYES, MSGBOXCANCEL_YESNO or MSGBOXCANCEL_NOYES
- **pBell** – If qtrue, indicates that the system bell should sound
- **pMsg** – The text for the message

```
RESloadString(gInstLib, needInitialConversion ? 9000 : 9001, msg);
msg.insertStr(strPathName);
if (ECOmessageBox(MSGBOXICON_NOYES, qfalse, msg))
{
  //add conditional processing here
}
```

ECOopenMenu()

void ECOopenMenu(HWND pHwnd, str255 &pMenuName, qlong pXOffset, qlong pYOffset, qbool pBottomAlign)

Opens the specified menu on the component's HWND.

- pHwnd – The component's window handle.
- pMenuName – The menu name.
- pXOffset – Desired horizontal offset in pixels.
- pYOffset – Desired verical offset in pixels.
- pBottomAlign – If qtrue, the menu is aligned to the bottom of the HWND.

ECOopenTraceLog()

void ECOopenTraceLog()

Opens the Omnis tace log window.

ECOnormalizeUChar()

qlong ECOnormalizeUChar(qbool pNFC, UChar *pDest, qlong pDestLen, UChar *pSource, qlong pSourceLen, qlong *pSizeNeeded, qlong *pOutputLen);

Normalizes a Unicode string by composing or decomposing accents. ECOnormalizeUchar will either normalize all data or return the size required if the destination buffer is not big enough.

- **pNFC** - qtrue to perform NFC, qfalse NFD
- **pDest** - The address of the destination buffer

- **pDestLen** - Length of pTarget in UChar units
- **pSource** - The address of the source buffer
- **pSourceLen** - Length of the source buffer in UChar units
- **pSizeNeeded** - If the available buffer space is too small,this receives the size needed for the destination buffer in UChar units
- **pOutputLen** - If the normalization is successful this receives the length of the output data in UChars
- **returns** - non-zero if an error occurred, zero if conversion was successful; if return value is -1 then *pSizeNeeded is the required buffer size in bytes

ECOnotifyObject()

---

qret ECOnotifyObject(qobjinst pObjInst,strxxx* pMethod, EXTfldval* pParams = 0, qshort pParamCnt = 0)

Notifies (pushes) a method on to the Omnis method stack. Used by non-visual worker objects to invoke their completion methods. This method packages its parameters into an EXTCompInfo structure then calls ECOnotifyObjectECI()

- **pObjInst** – The object instance pointer.
- **pMethod** – Name of the method to call inside the object instance.
- **pParams** – An optional array to EXTfldval pointers containing **paramters** to be passed to the method.
- **pParamCnt** – The number of parameters being sent.
- **returns** – e_ok on success.

```
EXTfldval retVal;
retVal.setList(getResultRow(qtrue), qtrue); //take ownership & transfer to retVal
str31 methodName(QTEXT("$completed"));
ECOnotifyObject( mObjPtr, &methodName, &retVal, 1);
```

ECOnotifyObjectECI()

---

qret ECOnotifyObjectECI(qobjinst pObjInst, strxxx* pMethod, EXTCompInfo* pEci)

Notifies (pushes) a method on to the Omnis method stack. Used by non-visual worker objects to invoke their completion methods.

- **pObjInst** – The object instance pointer.
- **pMethod** – Name of the method to call inside the object instance.
- **pEci** – Pointer to an EXTCompInfo structure containing the method parameters.
- **returns** – e_ok on success.

ECOpaintControlNotImplemented()

void ECOpaintControlNotImplemented(HDC pHdc, HWND pHwnd, rstrno pMessageResource, qcol pTextColor)

Paint control not implemented message in an external component. This is only present in order to support property get/set and the JSON library representation.

- **pHdc** – The device context to draw into.
- **pHwnd** – The components window handle.
- **pMessageResource** – Resource ID for the message to be displayed.
- **pTextColor** – Message is displayed in this colour.

```
qbool tqfWAV::paint(HWND hwnd, EXTCompInfo *eci)
{
  WNDpaintStruct paintStruct;
  WNDbeginPaint(hwnd, &paintStruct);
  GDIsetTextColor(paintStruct.hdc, GDI_COLOR_WINDOW);
  GDIfillRect(paintStruct.hdc, &paintStruct.rcPaint, GDIgetStockBrush(BLACK_BRUSH));
  ECOpaintControlNotImplemented(paintStruct.hdc, hwnd, 5100);
  WNDendPaint(hwnd, &paintStruct);
  return qtrue;
}
```

ECOpaintGrayFrame()

void ECOpaintGrayFrame(HDC pHdc, qrect &pRect)

Draws a gray frame around the control in design mode, so that the control is visible on the design window.

```
//Excerpt from the Accordion component paint() method
if (hwnd() == hWnd)
{
  qrect clientRect;
  WNDgetClientRect(hwnd(), &clientRect);
  qrect entryRect(clientRect);
  qdim clientWidth = clientRect.width();
  WNDpaintStruct paintStruct;
  WNDbeginPaint(mHWnd, &paintStruct);
  HDC hdc = paintStruct.hdc;
  qrect rcPaint = paintStruct.rcPaint;
  void *offscreenPaintInfo = GDIoffscreenPaintBegin(NULL, hdc, clientRect, rcPaint);
  if (offscreenPaintInfo)
  {
    WNDdefWindowProc(hwnd(), WM_ERASEBKGND, (WPARAM) hdc, 0, eci)
    qbool isDesign = ECOisDesign(mHWnd);
    if (isDesign)
    {
      // Draw design stuff
      ECOdrawDesignName(mHWnd, hdc);
      ECOdrawNumber(mHWnd, hdc);
      ECOdrawMultiKnobs(mHWnd, hdc);
      #ifndef isRCC
        // If there is no border, draw a gray frame so the object bounds are visible in design mode
        WNDborderStruct bs;
```

```
        WNDgetBorderSpec(hwnd(), &bs);
        if (WND_BORD_NONE == bs.mBorderStyle)
          ECOpaintGrayFrame(hdc, clientRect);
      #endif
    }
    else
    {
      //...
    }
    GDIoffscreenPaintEnd(offscreenPaintInfo);
  }
  WNDendPaint(mHWnd, &paintStruct);
}
```

ECOparseClientJSON()

---

qbool ECOparseClientJSON(qbyte *pJSON, qlong pJSONlen, qlong &pMessageType, EXTfldval &pReturnedList, EXTfldval &pErrorTextFval)

Parses a string of JSON text into a list. For use with JavaScript Client components.

- **pJSON** – Buffer containing non-unicode JSON.
- **pJSONlen** – Length of JSON text in bytes.
- **pMessageType** – An eWSmessageType (defined in obrowser.he)
- **pReturnedList** – Returns a list representation of the JSON string.
- **pErrorTextFval** – Returns an error message on failure.
- **returns** – qtrue on success, qfalse otherwise.

ECOparseJSON()

---

qbool ECOparseJSON(EXTfldval &pJSONData, EXTfldval &pReturnedListOrRow, EXTfldval &pReturnedErrorText, qbool pAllowArraysOfRows)

Parses a string of JSON text into a list.

- **pJSONData** – EXTfldval containing the source JSON.
- **pReturnedListOrRow** – Return value containing either a list or row variable, depending on how the source data is parsed.
- **pErrorTextFval** – Returns an error message on failure.
- **returns** – qtrue on success, qfalse otherwise.

ECOparseJSONArraryArray()

---

qbool ECOparseJSONArrayArray(EXTfldval &pJSONData, EXTfldval &pReturnedList, EXTfldval &pReturnedErrorText)

Parses a string of JSON text into a list. Where the supplied text contains an array of JSON arrays, the returned row represents an array of lists.

- **pJSONData** – EXTfldval containing the source text.
- **pReturnedList** – EXTfldval containing the returned list or row.
- **pReturnedErrorText** – An error message on failure.
- **returns** qtrue on success, qfalse otherwise.

ECOparseJSONObjectArrary()

qbool ECOsaveToJSONObjectArray(EXTfldval &pList, qlong pEncoding, EXTfldval &pReturnedJSON, EXTfldval &pReturnedErrorText)

Parses a string of JSON text into a list. Where the supplied text contains an array of JSON objects, the returned row represents an array of rows.

- **pJSONData** – EXTfldval containing the source text.
- **pReturnedList** – EXTfldval containing the returned list or row.
- **pReturnedErrorText** – An error message on failure.
- **returns** qtrue on success, qfalse otherwise.

ECOpostTaskMessage()

void ECOpostTaskMessage(qlongptr pTask, qlong pTaskUnq, char *pMessage, qfldval *pParams = 0, qshort pParamCount = 0)

Posts a message to Omnis by calling the specified method and passing zero or more parameters. Currently used to invoke the completion event for PDF reports.

- pTask – Task number obtained using a WNDsendMessage command.
- pTaskUnq – Unique identifier obtained using a WNDsendMessage command.
- pMessage – The method name to call.
- pParams – An array of zero or more EXTfldval pointers (qfldvals).
- pParamCount – the number of paramters being passed, must be 0-8 inclusive.

```
mTask = WNDsendMessage(0, WM_CONTROL, GET_TASK, (LPARAM) &mTaskUnq);
…

qfldval params[2];
params[0] = path.getFldVal();
params[1] = status.getFldVal();
ECOpostTaskMessage(output->mTask, output->mTaskUnq, (char*)"$pdfcomplete", params, 2);
```

ECOpromptForInput()

qret ECOpromptForInput(str255 &pPrompt, str255 &pTitle, qbool pCancel, EXTfldval &pReturnValue, qbool &pReturnValueOk);

Invokes a prompt dialog, allowing the external component to obtain an input value from the user. Note that the prompt and title must not contain the / character.

- **pPrompt** – Prompt text.
- **pTitle** – Title for the dialog window.
- **pCancel** – If qtrue, a cancel button is displayed.
- **pReturnValue** – EXTfldval containing the returen value.
- **pReturnValueOk** – qtrue if pReturnValue contains a valid item.
- **returns** – e_ok on success.

ECOpushWorkerCallback()

```
void ECOpushWorkerCallback(qobjinst pObjInst, strxxx *pMethod, lsttype *pResultList)
```

Pushes a method on to the Omnis method stack. Used by non-visual worker objects to invoke their completion methods.

- **pObjInst** – The object instance to be called.
- **pMethod** – Name of the method to call in the object.
- **pResultList** – A list parameter passed to the completion method.

ECOpushWorkerCallbackECI()

```
void *ECOpushWorkerCallbackECI() |
```

ECOreadBooleanConfigurationItem()

```
qbool ECOreadBooleanConfigurationItem(const char *pGroupName, const char *pItemName, qbool pDefaultValue)
```

Reads a boolean configuration value from the config.json file. Boolean items in the config file are formatted as follows:

```
"groupName": {
  "itemName1": false,
  "itemName2": true,
  etc...
}
```

- **pGroupName** – The group name inside the configuration file that contains pItemName.
- **pItemName** – The key value or item name to be read.
- **pDefaultValue** – The value to be returned if the configuration item cannot be found.
- **returns** – the Boolean configuration value.

ECOreadIntegerConfigurationItem()

```
qlong OMNISAPI ECOreadIntegerConfigurationItem(const char *pGroupName, const char *pItemName, qlong pDefaultValue)
```

Reads an integer configuration value from the config.json file. Integer items in the config file are formatted as follows:

```
"groupName": {
  "itemName1": 100,
  etc...
}
```

- **pGroupName** – The group name inside the configuration file that contains pItemName.
- **pItemName** – The key value or item name to be read.
- **pDefaultValue** – The value to be returned if the configuration item cannot be found.
- **returns** – the Integer configuration value.

ECOreadLocalisationItem()

qbool ECOreadLocalisationItem(EXTCompInfo *pEci, qshort pLocItemXn, str255 &pLocItemData)

Returns the localised text from the localisation database.

- **pEci** - Pointer to EXTCompInfo structure.
- **pLocItemXn** - identifies the localized item. This can be one of the cLOCxn constants. See source file LOCALISE.HE for a listing.
- **pLocItemData** - the localised text is returned in this parameter.
- **returns** - true if the item exists and text has been returned.

ECOreadSeparatorItem()

inline qchar ECOreadSeparatorItem(qshort pIndex)

Invokes the DAMgetSeparators callback and returns the separator at the specified index position: 1 - numeric decimal separator. 2 - numeric thousands separator. 3 - function parameter separator. 4 - import/export decimal separator. 5 - import/export param separator. 6 - calculation function separator & filename.fieldname separator. Other index values return zero.

ECOreadStringConfigurationItem()

void ECOreadStringConfigurationItem(const char *pGroupName, const char *pItemName, strxxx &pValue)

Reads a string configuration value from the config.json file. Strings in the config file are formatted as follows:

```
"groupName": {
  "itemName": "config value",
  etc...
}
```

- **pGroupName** – The group name inside the configuration file that contains pItemName.
- **pItemName** – The key value or item name to be read.
- **pValue** – Receives the return value, or empty if not found.

ECOreloadLibData()

qbool ECOreloadLibData(str80& pLibName)

Instructs the core to rebuild object lists, reloading icons, properties, events and constants for the specified component. The component's window object is closed if open.

- **pLibName** – object name, usually read from resource string 1000

ECOremoveObject()

void* ECOremoveObject(EXTCompInfo* pEci, HWND pHWnd, WPARAM pWParam)

Removes a pointer reference which had previously been stored via ECOinsertObject.

- **pInstance** - Specifies the Omnis instance which the pointer was originally inserted into.
- **pHWnd -** Specifies the HWND which is linked to the pointer.
- **pWParam** - ***Background components only.*** The WPARAM which was passed in from Omnis, this should be passed for background components only.
- **returns** - Returns the pointer originally passed into the ECOinsertObject function.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_OBJDESTRUCT:
    {
      CObj* myObject = (CObj *)ECOremoveObject( eci, hwnd );
      if ( NULL!= myObject)
        delete myObject;
        return 1L;
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOinsertObject, ECM_OBJDESTRUCT

ECOremoveNVObject()

void* ECOremoveNVObject(HINSTANCE pInstance,LPARAM pInstPtr)

Removes a pointer reference which had previously been stored via ECOinsertNVObject.

- **pInstance** - Specifies the Omnis instance which the pointer was originally inserted into.
- **pInstPtr –** Specifies the object instance pointer (as supplied by Omnis in LPARAM) which was originally used during ECOinsertNVObject.
- **returns** - Returns the pointer originally passed into the ECOinsertNVObject function.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_OBJDESTRUCT:
    {
      CObj* myObject = (CObj *)ECOremoveNVObject( eci->mOmnisInstance, lParam );
      if ( NULL!= myObject)
        delete myObject;
      return 1L;
```

```
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

See also ECOinsertNVObject, Non-visual components

ECOreplaceSeparators()

void ECOreplaceSeparators(strxxx &pString)

Replaces the Omnis separator characters (i.e. $root.$prefs.$sparators)

- **pString** is formatted as follows:
  Character 1 (dp): the decimal point character
  Character 2 (thou): the decimal number thousands separator
  Character 3 (func): the function parameter separator
  Character 4 (imdp): the import/export decimal point character
  Character 5 (imsep): the import/export comma delimiter character

ECOresetObjDetails()

qbool ECOresetObjDetails(qobjinst pObjInst, EXTfldval& pProps, EXTfldval& pMethods)

The ECOresetObjDetails function provides a means for non-visual components to dynamically alter the properties and methods which an object provides.

- **pObjInst** - Pointer which was originally generated by Omnis and passed to the external during ECM_OBJCONSTRUCT.
- **pProps** - A list containing the new properties for the object. This list should be in the format as returned by ECOreturnProperties. See the section on Control Handlers for more information on the exact structure of this list.
- **pMethods** - A list containing the new methods for the object. This list should be in the same format as returned by ECOreturn-Methods. See the section on Control Handlers for more information on the exact structure of this list.
- **Returns** - Returns true if successful, false otherwise.

See also Non-Visual components

ECOresolveJSONMember()

qlong ECOresolveJSONMember(EXTfldval &pJSONList, EXTfldval &pMember, EXTfldval &pResolvedMember, EXTfldval &pReturnedErrorText, qbool pWillAlter)

- **pJSONList** – An EXTfldval containing an EXTqlist of JSON items.
- **pMember** – The member name to be resolved.
- **pResolvedMember** – Receives the corresponding member value.
- **pReturnedErrorText** – On error, receives the error text
- **pWillAlter** – Set this to qtrue if you intend to alter pResolvedMember, in which case a reference(pointer) will be supplied.

ECOreturnCompID()

```
qlong ECOreturnCompID( HINSTANCE pInstance, EXTCompInfo* pEci, qshort pCompResNameID, qshort pCompType )
```

The ECOreturnCompID function provides support for the ECM_GETCOMPID message.

- **pInstance** - The instance which contains the resources(component name) for the component object. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pCompResNameID** - The resource id for the component name.
- **pCompType** - The component object base type. Of type cObjType_xxx and/or cRepObjType_xxx.
- **returns** - Returns the pCompType value which should returned to Omnis.

See also ECM_GETCOMPID

ECOreturnCompInfo()

```
qlong ECOreturnCompInfo( HINSTANCE pInstance, EXTCompInfo* pEci, qshort pLibNameResID, qshort pCompCount)
```

The ECOreturnCompInfo function provides support for the ECM_GETCOMPLIBINFO message.

- **pInstance** - The instance which contains the resources(library name) for the component library. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pLibNameResID** - The resource id for the component library name.
- **pCompCount** - The number of objects within the components' library.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETCOMPLIBINFO

ECOreturnConstants()

```
qbool ECOreturnConstants( HINSTANCE pInstance, EXTCompInfo* pEci, qlong pResStart, qlong pResEnd)
```

Provides support for the ECM_GETCONSTNAME message.

- **pInstance** - The instance which contains the resources for the constants. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pResStart** - Resource identifier for the first constant.
- **pResEnd** - Resource identifier of the last constant.
- **returns** - Returns true if successful, false otherwise.

It should be noted that this function is successful even if not all the resource slots between pResStart and pResEnd are populated. This would enable the component to easily modify groups of constants.

See also ECM_GETCONSTNAME

ECOreturnCStoreGrpName()

```
qbool ECOreturnCStoreGrpName( HINSTANCE pInstance, EXTCompInfo* pEci, qlong pResID )
```

The ECOreturnCStoreGrpName function provides support for the ECM_GETCOMPSTOREGROUP message.

- **pInstance** - The instance which contains the resources(custom component store group name). This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pResID** - The resource id for the custom component store group name.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETCOMPSTOREGROUP

ECOreturnEventMethod()

```
qbool ECOreturnEventMethod( HINSTANCE pInstance, EXTCompInfo* pEci, qlong pResStart)
```

The ECOreturnEventMethod function provides support for the ECM_GETEVENTMETHOD message.

- **pInstance** - The instance which contains the resources(method lines). This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pResStart** - The resource id for the start of the event method instructions. You should note that this function continues to add method lines until an empty string is located in the resources.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETEVENTMETHOD

ECOreturnEventMethod()

```
qbool ECOreturnEventMethod( HINSTANCE pInstance, EXTCompInfo* pEci, ECOmethodEvent* pTable, qshort pTableElements,
qbool pIncDesc = qtrue)
```

The ECOreturnEventMethod function provides support for the ECM_GETEVENTMETHOD message. This function generates an event method from the event table rather than from sequence of event lines in resources [see ECOreturnEventMethod(pInstance, pEci, pResStart) above]

- **pInstance** - The instance which contains the resources(method lines). This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pTable** - The pointer to the ECOmethodEvent structure.
- **pTableElements** - Number of events in the ECOmethodEvent structure.
- **pIncDesc** - True if description should be included as a comment in the event method.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETEVENTMETHOD

ECOreturnEvents()

```
qbool ECOreturnEvents( HINSTANCE pInstance, EXTCompInfo* pEci, ECOmethodEvent* pTable, qshort pTableElements )
```

The ECOreturnEvents function provides support for the ECM_GETEVENTNAME message.

- **pInstance** - The instance which contains the resources for the events. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pTable** - The pointer to the ECOmethodEvent structure.
- **pTableElements** - Number of events in the ECOmethodEvent structure.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETEVENTNAME, Component Events

ECOreturnIcon()

```
qbool ECOreturnIcon(HINSTANCE pInstance, EXTCompInfo* pEci, qshort pBitmapID )
```

The ECOreturnIcon function provides support for the ECM_GETCOMPICON message.

- **pInstance** - The instance which contains the resources(object icon) for the component object. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pBitmapID** - The resource id for the components' object icon.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETCOMPICON

ECOreturnMethodEvents

ECOreturnMethodEvents simply calls ECOreturnMethods.

ECOreturnMethods()

```
qbool ECOreturnMethods( HINSTANCE pInstance, EXTCompInfo* pEci, ECOmethodEvent* pTable, qshort pTableElements )
```

The ECOreturnMethods function provides support for the ECM_GETMETHODNAME message.

- **pInstance** - The instance which contains the resources for the methods. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pTable** - The pointer to the ECOmethodEvent structure.
- **pTableElements** - Number of functions or events in the ECOmethodEvent structure.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETMETHODNAME, Component Events

ECOreturnObjects()

```
qbool ECOreturnObjects( HINSTANCE pInstance, EXTCompInfo* pEci, ECOobject* pTable, qshort pTableElements )
```

The ECOreturnObjects function provides support for the ECM_GETOBJECT message.

- **pInstance** - The instance which contains the resources for the objects. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pTable** - The pointer to the ECOobject structure.
- **pTableElements** - Number of objects in the ECOobject structure.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETOBJECT, Non-Visual components

ECOreturnProperties()

```
qbool ECOreturnProperties( HINSTANCE pInstance, EXTCompInfo* pEci, ECOproperty* pPropTable, qshort pTableElements )
```

The ECOreturnProperties function provides support for the ECM_GETPROPNAME message.

- **pInstance** - The instance which contains the resources for the properties. This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pPropTable** - The pointer to the ECOproperty structure.
- **pTableElements** - Number of properties in the ECOproperty structure.
- **returns** - Returns true if successful, false otherwise.

See also ECM_GETPROPNAME, and the *Component Events* section.

ECOreturnVersion()

```
qlong ECOreturnVersion( qshort pMajorNumber, qshort pMinorNumber)
```

The ECOreturnVersion function provides support for the ECM_GETVERSION message.

- **pMajorNumber** - The major part of the components' version number.
- **pMinorNumber** - The minor part of the components' version number.

See also ECM_GETVERSION, GDIreadVersion

ECOreturnVersion()

qlong ECOreturnVersion(HINSTANCE pInst)

Web client components must use this mechanism to return the components version number from its resources. The component must have the following string resource

```
31020 "VER 1 5 %%ORFC_VER%%"
```

Please note the spaces. These are important. The 1 specifies the major version, and the 5 specifies the minor version. Non-web client components can also use this new mechanism to return the version number.

- **pInstance** - The instance which contains the string resources required. This would normally be gInstLib.

See also ECM_GETVERSION, GDIreadVersion

ECOrunCommand()

DWORD ECOrunCommand(TCHAR *pCmd)

Run a command prompt command on windows (Unix and OSX can just use the POSIX API system())

- **pCmd** – contains the DOS command to be executed.
- **returns** - the exit code from the command process.

ECOrunningInHeadlessServer()

qbool ECOrunningInHeadlessServer()

Returns qtrue if running using an Omnis headless server edition on Linux.

ECOsaveFileDialog()

qbool ECOsaveFileDialog(HINSTANCE pInstance, HWND pOwner, qlong pResTitle, qlong pResFilter, str255& pFileName, str255* pInitDir = 0) qbool ECOsaveFileDialog(HINSTANCE pInstance, HWND pOwner, strxxx& pTitle, strxxx pFilter, str255& pFileName, str255* pInitDir = 0)

The ECOsaveFileDialog function enables the component to invoke the operating system save file dialog.

- **pInstance** - The instance which contains the string resources required. This would normally be gInstLib.
- **pOwner** - The HWND of the owner.
- **pResTitle or pTitle** - The resource id or string for the title of the save file dialog.
- **pResFilter or pFilter** - The resource id or string for the filter string of the save file dialog. Any platform dependent filters are removed if not required. **Note: ONLY used on WINDOWS.**
- **pFileName** - The str255 object which contains the file name upon return, if successful.
- **pInitDir** - The pointer to the str255 object which specifies the initial folder. May be NULL.
- **returns** - Returns true if a file has been selected, false otherwise.

```
// Save file
str255 saveFile;
if ( ECOsaveFileDialog( gInstLib,hwnd,myResTitle,myResFilter, saveFile) )
{
  saveDataToFile( saveFile );
}
```

***ECOsaveToClientJSON()**

void ECOsaveToClientJSON(qlong pMessageType, EXTfldval &pList, EXTfldval &pReturnedJSON)

- pMessageType –
- pList –
- pReturnedJSON -

ECOsaveToJSON()

qbool OMNISAPI ECOsaveToJSON(EXTfldval &pListOrRow, qlong pEncoding, EXTfldval &pReturnedJSON, EXTfldval &pReturnedErrorText)

Converts the supplied list or row to JSON text.

- pListOrRow – List or row representing a JSON object or array.
- pEncoding – the Unicode encoding to be used, e.g. preUniTypeCharacter (see dmconst.he)
- pReturnedJSON – receives the “saved” JSON object.
- pReturnedErrorText – on error, contains the error text.

ECOsaveToJSONArrayArray()

qbool OMNISAPI ECOsaveToJSONArrayArray(EXTfldval &pList, qlong pEncoding, EXTfldval &pReturnedJSON, EXTfldval &pReturnedErrorText)

Converts the supplied list JSON text.

- pList – List representing a two-dimensional JSON array.
- pEncoding – the Unicode encoding to be used, e.g. preUniTypeCharacter (see dmconst.he)
- pReturnedJSON – receives the “saved” JSON object.
- pReturnedErrorText – on error, contains the error text.

ECOsaveToJSONObjectArray()

qbool ECOsaveToJSONObjectArray(EXTfldval &pList, qlong pEncoding, EXTfldval &pReturnedJSON, EXTfldval &pReturnedErrorText)

Converts the supplied list JSON text.

- pList – List representing a JSON object.
- pEncoding – the Unicode encoding to be used, e.g. preUniTypeCharacter (see dmconst.he)
- pReturnedJSON – receives the “saved” JSON object.
- pReturnedErrorText – on error, contains the error text.

ECOsendCompEvent()

qbool ECOsendCompEvent(HWND pHwnd, EXTCompInfo* pEci, qlong pEventID, qbool pExecNow)

The ECOsendCompEvent function enables the component to send Omnis object events. This function is useful for components which need to add the parameters manually to the EXTCompInfo structure. Most components use ECOsendEvent in preference to this function.

- **pHwnd** - The HWND of the object.
- **pEci** - The EXTCompInfo structure which contains the event parameters.
- **pEventID** - The id of the event.
- **pExecNow** - True if the event should be processed by Omnis immediately, false otherwise.
- **returns** - Returns true if the event has been processed by Omnis, false if it has been discarded. If pExecNow is false this function always returns true.

```
// Event myEvent1 occurred. Send event to OMNIS
EXTCompInfo* eci = new EXTCompInfo();
eci->mParamFirst = 0;
// Add parameters to EXTCompInfo structure
EXTfldval myParam1;
myParam1.setlong( someData );
// Add parameter 1
ECOaddParam(eci,&myParam1,0,0,0,1,0);
// Send event to OMNIS
qbool eventOk = ECOsendCompEvent( hwnd, eci, myEventId, qtrue );
// Delete parameters from EXTCompInfo structure
ECOmemoryDeletion( eci );
// Delete eci structure
delete eci;
```

See also ECOsendEvent

ECOsendEvent()

qbool ECOsendEvent(HWND pHwnd, qlong pEventID, EXTfldval* pParams = 0, qshort pParamCnt = 0, qbool pExecNow = EEN_EXEC_IMMEDIATE)

The ECOsendEvent function enables the component to send Omnis object events. This function is basically a wrapper for ECOsend-CompEvent.

- **pHwnd** - The HWND of the object.
- **pEventID** - The id of the event.
- **pParams** - Pointer to an array of EXTfldval which contain the parameters for the event.
- **pParamCnt** - Number of parameters for the event.
- **pExecNow** - can be one of the following

EEN_EXEC_LATER - the event should be processed by OMNIS later. The event is added to the end of the Omnis event queue

EEN_EXEC_IMMEDIATE - the event should be processed by Omnis immediately

EEN_EXEC_PUSH (v3.1) - the event should be pushed on the Omnis event queue in front off all existing events on the queue.

- **returns** - Returns true if the event has been processed by Omnis, false if it has been discarded. If pExecNow is false this function always returns true. When calling ECOsendEvent from Web Client components, ECOsendEvent will always return qtrue. The correct result is send to the component once the server returns control to the client. See ECM_EVENTRESULT.

```
// Send second event code to OMNIS (extract from CLOCK example)
EXTfldval newSeconds;
newSeconds.setLong(datetime->tm_sec);
ECOsendEvent( mHWnd, cClockEvSecs, &newSeconds, 1 );
```

See also ECOsendCompEvent

ECOsendTab()

```
void ECOsendTab(HWND pHOmnisCompHwnd, qbool pShiftTab)
```

Sends a TAB character to the component's HWND, to be detected by its event handler.

- pHOmnisCompHwnd – Handle to the component window.
- pShiftTab – Set to qtrue to send SHIFT + TAB.

ECOsetConnectFlags()

```
void ECOsetConnectFlags(LPARAM pConnectLparam)
```

Deprecated. This call is no longer in use (formerly used with OmnisX).

ECOsetCustomTabName()

```
qbool ECOsetCustomTabName( HINSTANCE pInstance, EXTCompInfo* pEci, qlong pResID )
```

The ECOsetCustomTabName function provides support for the ECM_CUSTOMTABNAME message.

- **pInstance** - The instance which contains the resources(custom tab name). This would normally be gInstLib.
- **pEci** - The pointer to EXTCompInfo structure.
- **pResID** - The resource id for the custom tab name.
- **returns** - Returns true if successful, false otherwise.

See also ECM_CUSTOMTABNAME

ECOsetDTformat()

```
void ECOsetDTformat( str80& pFormat, qshort pFormatType )
```

The ECOsetDTformat function enables the component to set the Omnis internal variables #FD, #FT, #FDT. This function is most useful in the Omnis Web Thin-Client so that controls can localize their date/time routines.

- **pFormat** – The new string format for the required format type. Please note that this variable will contain the old string on return.

- **pFormatType** – The required data type. This can be dpFdate1900, dpFdate1980, dpFdate2000 for #FD (date formatting); or dpFtime for #FT (time formatting); others types will be for #FDT (date and time formatting).

```
// Set the date formatting (#FD for European or American formatting)
str80 s;
if ( EuropeanDateSystem )
  s=str80("D m Y");
else
  s=str80("m D Y");
ECOsetDTformat(s, dpFdate2000 );
// Get the date string (which will be formatted appropriately)
str255 displayString; myDate.getChar( displayString );
// Set #FD back to the old value
ECOsetDTformat(s, dpFdate2000 );
```

ECOsetError()

void ECOsetError(qlong pErrNum, str255* pErrText)

The ECOsetError function enables the component to set the Omnis variables #ERRCODE and #ERRTEXT.

- **pErrNum** - The error number stored in #ERRCODE.
- **pErrText** - The pointer to the str255 object stored in #ERRTEXT.

```
// Set OMNIS #ERRCODE & #ERRTEXT variables
// #ERRCODE
qlong errCode = 1;
// #ERRTEXT
str255 errText("Something bad has happened");
ECOsetError( errCode, &errText );
```

ECOsetNotifyObjectWhenPushed()

void ECOsetNotifyObjectWhenPushed(qobjinst pInst)

Used by Worker::pushWorkerCallback, ECOsetNotifyObjectWhenPushed() is called when pushWorkerCallback() is called with a flag value of cPushWorkerCallbackFlag_NotifyObjectWhenPushed (see worker.he).
Forces the worker object's $notify() method to be called in addition to/before pushing the completion method on to the method stack.

- **pInst** – Worker object instance (qobjinst)

ECOsetParameterChanged()

void ECOsetParameterChanged(EXTCompInfo* pEci, qshort pParamNum)

The ECOsetParameterChanged function should be called by the component when a method parameter has been modified. Failure to call this function results in any modifications made to a method parameter being lost on return to Omnis. The method parameter must previously been defined with the EXTD_FLAG_PARAMALTER flag.

- **pEci** - The pointer to the EXTCompInfo structure containing the function parameters.
- **pParamNum -** The number of the parameter which has been modified.

See also ECM_METHODCALL, EXTD_FLAG_PARAMALTER

ECOsetParameterKeepTableInstance()

void ECOsetParameterKeepTableInstance(EXTCompInfo* pEci, qshort pParamNum)

Mark parameter as a list for which the original table instance is to be kept.

(Only applies if ECOsetParameterChanged is called as well for the parameter).

- **pEci** - The pointer to the EXTCompInfo structure containing the function parameters.
- **pParamNum -** The number of the parameter which has been modified.

ECOsetProperty()

qbool ECOsetProperty(HWND pHwnd, qshort pAnum, EXTfldval &pFval)

The ECOsetProperty enables the component to set the Omnis standard object properties.

- **pHwnd** - The HWND of the object.
- **pAnum** - The anum of the property which is go to be set (See ANUMS.HE for the list of possible anums).
- **pFval** - The EXTfldval object which contains the property, if successful.
- **returns** - Returns true if successful, false otherwise.

```
// Set the name from the fldval
str255 str("#S1");
fldname.setChar( str );
// Set $dataname property
EXTfldval fldname;
if ( ECOsetProperty( mHwnd, anumFieldname, fldname ) )
{
  // Successfully set the attribute
}
```

ECOsetupCallbacks()

void ECOsetupCallbacks(HWND pHwnd, EXTCompInfo* pEci)

The ECOsetupCallbacks function initializes the global array of pointers which contain the callback function pointers. This *must* be called upon entry to all window procedures that Omnis invokes.

- **pHwnd** - The HWND that received the message.
- **pEci** - The pointer to EXTCompInfo structure which contains the callback pointers.

```
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

ECOsleep()

void ECOsleep(qlong pMilliSeconds)

Puts the current thread to sleep/blocks for the specified time period.

- **pMilliSeconds** – Number of milliseconds.

ECOtlsSetValue()

void ECOtlsSetValue(threadStorageIndex pIndex, void* pValue)

Used by a worker object to set one of a Worker delegate's "thread local storage" parameters. These give the worker thread access to the main thread's global variables and method stack.

- **pIndex** – indicates which TLS value is required.
- **pValue** – receives a pointer to the core's storage item.

```
DAMthreadLocalStorage *tls = 0;
if (_worker->needTLS())
{
   tls = (DAMthreadLocalStorage *)_params[2];
   ECOtlsSetValue(tls->dmVarsIndex, tls->dmVars);
   ECOtlsSetValue(tls->gmVarsIndex, tls->gmVars);
   ECOtlsSetValue(tls->mStackIndex, tls->mStack);
}
```

ECOtoUChar()

qlong ECOtoUChar(ECOconverter pConverter, UChar *pTarget, qlong pTargetLen, const char *pSource, qlong pSourceLen, qlong *pSizeNeeded, qlong *pOutputLen)

Converts from a code page identified by pConverter to UChar (UTF-16). ECOtoUchar will either convert all data or return the size required if the target buffer is not big enough.

- **pConverter** - The converter for the source code page
- **pTarget** - The address of the destination buffer
- **pTargetLen** - Length of pTarget in UChar units
- **pSource** - The address of the source buffer
- **pSourceLen** - Length of the source buffer IN BYTES
- **pSizeNeeded** - If the available buffer space is too small,this receives the size needed for the target buffer in UChar units
- **pOutputLen** - If the conversion is successful this receives the length of the output data in UChars
- **returns** - non-zero if an error occurred, zero if conversion was successful; if return value is -1 then *pSizeNeeded is the required buffer size in UChar units

ECOtryEnterCriticalSection()

qbool ECOtryEnterCriticalSection(EXTcriticalSection *cs)

Tests whether the specified EXTcriticalSection is available and claims it if it is. ECOtryEnterCriticalSection() returns immediately, allowing the critical section to be polled repeatedly if required.

- **cs** – An EXTcriticalSection initialized previously using ECOinitCriticalSection().
- **returns** – qtrue if the critical section was claimed, qfalse otherwise.

ECOUpdateJSON()

qlong ECOupdateJSON(EXTfldval &pJSONList, EXTfldval &pMemberIdFval, EXTfldval *pValueFval, EXTfldval *pMemberNameToAddOrRemove, EXTfldval &pErrorText)

Call ECOupdateJSON in one of three ways:

a. With non-null pValueFval and null pMemberNameToAddOrRemove to set existing member identified by pMemberIdFval

b. With non-null pValueFval and non-null pMemberNameToAddOrRemove to add new member pMemberNameToAddOrRemove (with value pValueFval) to existing member identified by pMemberIdFval

c. With null pValueFval and non-null pMemberNameToAddOrRemove to remove existing member pMemberNameToAddOrRemove from existing member identified by pMemberIdFval.

- **pJSONList** – List containing exisitng JSON content.
- **pMemberIdFval** – ID of the member to be updated.
- **pValueFval** – List or row containing new JSON content.
- **pMemberNameToAddOrRemove** – see above.
- **pErrorText** – On error, receiuves the error text.
- **returns** – Zero on success.

```
void tqfOJSONobject::setnull(EXTCompInfo *pEci)
{
   qbool ok = qfalse;
   if (checkParameterCount(pEci, 1, 1))
   {
      EXTParamInfo *memberIdParam = ECOfindParamNum(pEci, 1);
      EXTfldval memberIdFval((qfldval) memberIdParam->mData);
      EXTfldval nullFval;
      nullFval.setNull(fftBinary, 0);
      mErrorCode = ECOupdateJSON(mJSONlistOrRowFval, memberIdFval, &nullFval, 0, mErrorText);
      if (!mErrorCode)
         ok = qtrue;
   }
   returnBoolean(pEci, ok);
}
```

ECOUpdateJSONArray()

qlong ECOupdateJSONArray(EXTfldval &pJSONList, EXTfldval &pMemberIdFval, EXTfldval *pValueFval, qbool pPush, EXTfldval &pErrorText)

Updates an existing JSON array or returns part of a JSON array.

If pPush is true, pValueFval is the value to add to the end of the array.

If pPush is false pValueFval receives the value removed from the end of the array (a pop operation).

- **pJSONList** – Exisitng list containing a JSON array.
- **pMemberIdFval** – Identifies the array member.
- **pValueFval** – Either supplies or receives the array element value.
- **pPush** – qtrue to add a new element to the array, qfalse to remove.
- **pErrorText** – On error, receives the error text.

ECOupdatePropInsp()

void ECOupdatePropInsp(HWND pHOmnisCompHwnd, qlong pPropId = 0)

The ECOupdatePropInsp function can be called by the component to update the Property Manager. This function may be called during either design or runtime.

- **pHOmnisCompHwnd** - The HWND of the object.
- **pPropId -** The property id which is updated. If the property id is not supplied all properties are updated.

```
// Update all properties
ECOupdatePropInsp( mHwnd );
// Update myPropId
ECOupdatePropInsp( mHwnd, myPropId );
```

See also WM_CONTROL - UPDATE_PROPINSPECTOR

ECOuprCmp()

ECOAPI qshort OMNISAPI ECOuprCmp(qchar* add1, qchar* add2, qlong len)

The ECOuprCmp function compares the text of two strings after converting to upper case. Returns 0 if the two strings are equal, -1 if add1 is less than add2, 1 if add1 is greater than add2. The two text strings are left unchanged.

- **add1**- The address of string 1.
- **add2** - The address of string 2.
- **len** - The number of characters to use in the comparison.

ECOwriteBooleanConfigurationItem()

```
void ECOwriteBooleanConfigurationItem(const char *pGroupName, const char *pItemName, qbool pValue)
```

Writes/updates a boolean configuration item in the config.json file.

- pGroupName – The group name that contains the item.
- pItemName – The item name to set.
- pValue – The item value.

ECOwriteIntegerConfigurationItem()

```
void ECOwriteIntegerConfigurationItem(const char *pGroupName, const char *pItemName, qlong pValue)
```

Writes/updates an integer configuration item in the config.json file.

- pGroupName – The group name that contains the item.
- pItemName – The item name to set.
- pValue – The item value.

ECOwriteStringConfigurationItem()

```
void ECOwriteStringConfigurationItem(const char *pGroupName, const char *pItemName, strxxx &pValue, qbool
pCanWriteInteger = qfalse);
```

Writes/updates a string configuration item in the config.json file.

- pGroupName – The group name that contains the item.
- pItemName – The item name to set.
- pValue – The item value.
- pCanWriteInteger – If qtrue, pValue can contain an integer value.

WNDdefWindowProc()

```
qbool WNDdefWindowProc( HWND pHwnd, LPARAM pMsg, WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci )
```

The WNDdefWindowProc function calls the default window processing. All messages not handled must be passed to this function.

- **pHwnd** - The HWND that received the message.
- **pMsg** - The window message.
- **wParam** - wParam of the message.
- **lParam** - lParam of the message.
- **pEci** - EXTCompInfo pointer that was passed into the window procedure.
- **returns** - The result of Omnis processing the message.

Memory Functions

When creating cross-platform external components, you may need to manipulate memory manually. As some objects may need to use greater than 64K of memory, for example imaging components, a set of memory functions are available to cope with the 16bit problems encountered under 16 bit Windows.

The MEM functions are cross-platform allowing your code to remain independent of the operating system in which you develop.

MEMcalloc()

```
qchar* MEMcalloc( qulong pSize )
```

Allocates a block of memory, and locks it in memory. The allocation can be greater than 64K. The memory allocated is initialized to 0.

- **pSize** - The amount of memory to allocate.
- **returns** - The locked memory address.

MEMdataLen()

```
qulong MEMdataLen( void* pBuffer )
```

Returns the size of a buffer.

- **pBuffer -** The buffer to return a size for. This buffer must have previous been allocated with MEMmalloc or MEMcalloc.
- **returns** - The length of the buffer.

MEMdecAddr()

```
qchar* MEMdecAddr( qchar* pAddress, qlong pOffset )
```

Decrements a memory address by an offset.

- **pAddress -** The address to decrement.
- **pOffset** - The amount to decrement by.
- **returns** - A new address.

Note: This function is very important under Windows 16bit due to 64K segments. When handling large memory blocks, this function must be used to adjust pointers.

You can use MEMdecAddr() and MEMincAddr() with the result of MEMglobalLock.

MEMfree()

```
void MEMfree( void* pBuffer )
```

Reclaims the memory previous allocated from a MEMmalloc or MEMcalloc call.

- **pBuffer -** The buffer to destroy. This buffer must have previous been allocated with MEMmalloc or MEMcalloc.

MEMglobalAlloc()

```
HGLOBAL MEMglobalAlloc ( qlong pLength, qbool pZeroInited = qfalse )
```

Allocates a block of memory.

- **pLength** - The amount of memory to allocate.
- **pZeroInited** - qtrue if the memory should be cleared to 0.
- **returns** - A new HGLOBAL handle.

MEMglobalFree()

```
void MEMglobalFree ( HGLOBAL pMemory )
```

Reclaims the memory previously allocated by a MEMglobalAlloc. The data must be in an unlocked state.

- **pMemory** - The memory to be destroyed.

MEMglobalHandle()

```
HGLOBAL MEMglobalHandle ( void* pAddress )
```

Returns a memory handle given an address.

- **pAddress** - An address to return the memory handle for.
- **returns** - A memory handle.

MEMglobalLock()

```
void* MEMglobalLock (HGLOBAL pMemory )
```

Locks a memory handle, increments the lock count and returns the address of the handles first byte.

- **pMemory** - The memory handle to lock
- **returns** - The address of the first byte of memory associated with the memory handle.

MEMglobalReAlloc()

```
HGLOBAL MEMglobalReAlloc ( HGLOBAL pMemory, qlong pNewLength )
```

Reallocates a block of memory.

- **pMemory** - The old memory handle.
- **pNewLength** - The new size of the memory block.
- **returns** - A new HGLOBAL handle.

MEMglobalSize()

```
qlong MEMglobalSize ( HGLOBAL pMemory )
```

Returns the size of a memory handle.

- **pMemory -** The memory handle.
- **returns** - The length of the handles data.

MEMglobalUnlock()

```
void MEMglobalUnlock ( HGLOBAL pMemory )
```

Unlocks a memory handle and decrement the lock count.

- **pMemory** - The memory handle to unlock.

MEMincAddr()

```
qchar* MEMincAddr( qchar* pAddress, qlong pOffset )
```

Increments a memory address by an offset.

- **pAddress -** The address to be incremented.
- **pOffset** - The amount to increment by.
- **returns** - A new address.

Note: This function is very important under Windows 16bit due to 64K segments. When handling large memory blocks, this function must be used to adjust pointers.

MEMmalloc()

```
qchar* MEMmalloc( qulong pSize )
```

Allocates a block of memory, and locks it in memory. The allocation can be greater than 64K.

- **pSize** - The amount of memory to allocate.
- **returns** - The locked memory address.

MEMmemcmp()

qint2 MEMmemcmp(void* pAddress1, void* pAddress2, qlong pTestLen)

Compares two blocks of memory.

- **pAddress1** - Points to the starting address of the first block of memory.
- **pAddress2** - Points to the starting address of the second block of memory.
- **pTestLen** - The size of the memory blocks, in bytes, to compare.
- **returns** - 0, -1 or 1.

Returns 0 if both memory blocks match.

Returns -1 if memory block 1 is less than memory block 2.

Returns 1 if memory block 1 is greater than memory block 2.

MEMmemFill()

void MEMmemFill(void* pFillAddress, qint4 pFillLen, qchar pFillChar)

Fills memory with a specified character.

- **pFillAddress** - The address in memory to fill.
- **pFillLen** - The number of bytes to fill.
- **pLen** - The character to be used to fill memory.

Example:

```
qchar stringOne[] = "????";
MEMmemFill(&stringOne[0],4,'*');
// Would result in stringOne = ****
```

MEMmovel()

void MEMmovel(void* pSrc, void* pDst, qlong pLen)

Move memory from source to destination copying data from left to right (start to end).

- **pSrc** - The source address.
- **pDst** - The destination address.
- **pLen** - The number of bytes to copy.

Example:

```
qchar stringOne[] = "*OMNIS*";
MEMmovel(&stringOne[1],&stringOne[0],6);
// Would result in stringOne = OMNIS**
```

MEMmover()

void MEMmover(void* pSrc, void* pDst, qlong pLen)

Move memory from source to destination copying data from right to left (end to start)

- **pSrc** - The source address.
- **pDst** - The destination address.
- **pLen** - The number of bytes to copy.

Example:

```
qchar stringOne[] = "*OMNIS*";
MEMmoveR(&stringOne[0],&stringOne[1],6);
// Would result in stringOne = **OMNIS
```

MEMrealloc()

qchar* MEMrealloc(void* pBuffer, qulong pNewLen)

Alters the size of the buffer to a different size.

- **pBuffer -** The buffer to be re-allocated. This buffer must have previously been allocated with MEMmalloc or MEMcalloc.
- **pNewLen** - The new size for the buffer.
- **returns** - A pointer to the reallocated buffer. The original pointer and new pointer may be different.

MEMscanf()

qlong MEMscanf(qshort pDirection, qlong pLen, qchar pScanChar, const void * pScanAddress)

Scans a memory location for a character.

- **pDirection** - If positive, the scan is performed from the beginning to the end of memory block, otherwise the scan is performed from the end to the beginning.
- **pLen** - The number of characters to scan. If this is positive the search is forward, if this is negative the search is from the end of the buffer (the length is added to the buffer before scan starts).
- **pScanChar** - The character to scan for.
- **pScanAddress** - The address to scan.
- **returns -** The index position from the start of the scan or pLen if failed to locate character.

Example:

```
// Find character N in string
qchar stringOne[] = "OMNIS";
qlong posOfN = MEMscanf(qtrue,5,'N',&stringOne[0]);
// Would result in posOfN = 2
qlong posOfA = MEMscanf(qtrue,5,'A',&stringOne[0]);
// Result in posOfA = 5 as MEMscanf failed to find A in memory
```

The following set of memory functions all support greater than 64K allocation blocks. The memory is automatically locked and pointers to the memory are returned. For more control when the memory is locked, use the memory handling functions.

HANglobalAlloc()

```
qHandle HANglobalAlloc ( qlong pLength, qbool pZeroInited = qfalse )
```

Allocates a block of memory, from Omnis the internal memory cache.

- **pLength** - The amount of memory to allocate.
- **pZeroInited** - qtrue if the memory should be cleared to 0.
- **returns** - A new qHandle.

HANglobalReAlloc()

```
qHandle HANglobalReAlloc(qHandle pHandle, qlong pNewLen )
```

Reallocates a block of Omnis memory.

- **pMemory** - The old memory handle.
- **pNewLength** - The new size of the memory block.
- **returns** - A new qHandle.

HANglobalSize()

```
qlong HANglobalSize ( qHandle pGlobal, qlong pNewLen )
```

Returns the size of a memory handle.

Note: This could be bigger than the data length.

- **pMemory -** The memory handle.
- **returns** - The length of the handles data.

HANglobalFree()

```
void HANglobalFree ( qHandle pHandle )
```

Reclaims the memory previously allocated by a HANglobalAlloc.

- **pMemory** - The memory to be handed back into the Omnis memory cache.

qHandlePtr Class

The qHandlePtr class gives your external components convenient ways to manipulate Omnis cache memory easily.

qHandlePtr::qHandlePtr

qHandlePtr:: qHandlePtr()

Creates an empty qHandlePtr class.

qHandlePtr::qHandlePtr()

qHandlePtr(qHandle pHandle, qlong pOffset)

Constructs a qHandlePtr class.

- **pHandle**- The memory to be handed back into the Omnis memory cache.
- **pOffset** - The offset into the memory.

qHandlePtr::qHandlePtr()

qHandlePtr (const qHandlePtr& pHptr)

Constructs a qHandlePtr class from an existing qHandlePtr.

- **pHptr**- an Existing qHandlePtr class.

qHandlePtr::operator =()

void operator =(qniltype qnil1)

Assigns the handle of the qHandlePtr to zero.

qHandlePtr::operator =()

void qHandlePtr:: operator =(const qHandlePtr& pHptr)

Duplicates an existing qHandlePtr.

- **pHptr-** an Existing qHandlePtr class.

qHandlePtr::operator +=()

void operator +=(qlong pInc)

Increments the offset in to memory block.

- **pInc-** The amount to increment the offset.

qHandlePtr::operator -=()

void operator -=(qlong pDec)

Decrements the offset in to memory block.

- **pDec-** The amount to decrement the offset.

qHandlePtr::operator +()

qHandlePtr operator +(qlong pDel)

Makes a copy of itself and increments the copy specified by pDel.

- **pInc-** The amount to increment the offset in the copy.

qHandlePtr::operator -()

qHandlePtr operator +(qlong pDel)

Makes a copy of itself and decrements the copy specified by pDel.

- **pDec-** The amount to decrement the offset in the copy.

qHandlePtr::operator !()

qbool operator !()

Tests whether the handle is non-zero.

qHandlePtr::operator *()

qchar* operator *()

Return a qchar pointer which is calculated as :-

- **returns** - Memory block base + Offset.

qHandlePtr::operator *()

qchar* operator *(qlong pDel)

Return a qchar pointer which is calculated as :-

- **returns** - Memory block base + Offset + pDel.

qHandlePtr::operator []()

qchar& operator [](qlong pDel)

Return a qchar reference which is calculated as

- **returns** - Memory block base + Offset + pDel.

qHandlePtr::dataLen()

qulong dataLen()

Return the actual length of the data contained in the handle.

N.B. This might not be the same as the result of HANglobalSize, this is because the data contained in this memory block might not occupy all of it.

- **returns** - Data Length of the Handle.

qHandlePtr::dataLen()

void dataLen(qulong pSize)

Sets the actual length of the data contain in the handle.

- **pSize** - Sets the Data Length of the handle.

qHandlePtr::getOffset()

qulong getOffset()

Returns the current offset into the memory block

- **returns** - offset into the memory block.

qHandlePtr::getHandle()

void getHandle(qHandle& pHandle)

Returns the handle of the qhandleptr

- **pHan** - a qHandle memory block.

qHandlePtr::set()

void set(qHandle pHandle, qlong pOffset)

Sets the qhandleptr from the provided parameters

- **pHandle** - a qHandle memory block.
- **pOffset** - Offset into the memory block.

qHandlePtr::setOffset()

```
void setOffset(qlong pOffset)
```

Set the Offset of the qhandleptr.

- **pOffset**- Offset into the memory block.

qHandlePtr::setNull()

```
void setNull()
```

Set the handle to zero.

Resource Functions

The following set of RES or Resource functions allow cross-platform access to your external components resources.

REScloseLibrary()

```
void REScloseLibrary ( HINSTANCE pInstance)
```

Closes an instance of a DLL previously opened with RESopenLibrary.

- **pInstance -** An instance of a library already opened with RESopenLibrary.

See also RESopenLibrary

REScloseResourceFork() (MacOS only)

```
void REScloseResourceFork( qshort pResFileNum )
```

Closes a Macintosh resource file.

- **pResFileNum -** The number returned from the **RESopenResourceFork** API.

See also RESopenResourceFork

```
HINSTANCE RESgetOmnisDAT( EXTCompInfo* pEci )
```

Returns an instance to the Omnis resources library (OMNISDAT.DLL on Windows).

- **pEci** - The pointer to the EXTCompInfo structure.
- **returns -** An instance to the Omnis resources.

Note: The instance returned *must not* be closed (i.e. via REScloseLibrary).

RESloadBitmap()

HBITMAP RESloadBitmap(HINSTANCE pLibrary, qlong pBmpID)

Retrieves a HBITMAP object from the resources.

- **pLibrary** - The library to extract a bitmap from.
- **pBmpID** - The resource id of the bitmap.
- **returns -** A bitmap object.

Note: The bitmap object must be deleted with GDIdeleteBitmap.

RESloadDialog()

qHandle RESloadDialog(HINSTANCE pInstance, qlong pResID)

Retrieves a dialog resource for use with custom output devices. RESloadDialog should be called in response to a PM_OUT_GETPARMDLG message (see print manager reference).

- **pInstance** - The library to extract a bitmap from.
- **pResID** - The resource id of the dialog.
- **returns -** An Omnis handle.

Note: The bitmap object must be deleted with GDIdeleteBitmap.

RESloadString()

qlong RESloadString(HINSTANCE pInstance, qlong pResID, qchar* pBuffer, qlong pBufferLen)

Retrieves a string from an open library resources.

- **pInstance** - The library to extract a string from.
- **pResID** - The resource id of the string.
- **pBuffer -** The address to receive the string
- **pBufferLen** - The maximum number of bytes allowed to copy into **pBuffer**
- **returns -** The actual number of bytes copied into **pBuffer**

RESloadString()

qlong RESloadString(HINSTANCE pInstance, qlong pResID, strxxx& pString)

Retrieves a string from an open library resources.

- **pInstance** - The library to extract a string from.
- **pResID** - The resource id of the string.
- **pString -** The string variable to receive the string.
- **returns -** The actual number of bytes copied into **pString**

RESopenLibrary()

HINSTANCE RESopenLibrary (strxxx& pLibraryPath)

Opens another library file. This can be used if, for example, you keep resources in another file.

The component must call REScloseLibrary when it is finished with the library file.

- **pInstance -** The name of the library file to open
- **returns -** An instance to the opened library if successful, zero otherwise.

See also REScloseLibrary

RESopenResourceFork() (MacOS only)

qshort RESopenResourceFork(HINSTANCE pInstance)

This function should be used on the Macintosh if a Macintosh Resource Manager API needs to be called, for example, GetResource. The HINSTANCE can be that normal global component instance gInstLib, or another HINSTANCE that was returned from RESopenLibrary.

```
str255 path = str255( "HD:Another File" );
HINSTANCE anotherInst = RESopenLibrary( path );
if ( anotherInst )
{
  qshort resRefNum = RESopenResourceFork(anotherInst );
  Handle macHandle = GetResource( 'TYPE', id );
  REScloseResourceFork(resRefNum);
  REScloseLibrary(anotherInst);
}
else
{
  // Open library failed
}
```

- **pInstance** - The instance of the library.
- **returns -** Returns the number of the resource fork.

See also RESopenLibrary, REScloseResourceFork

Bit Functions

You can use the following functions for bit operations. The bit index range for all of the functions is 0-31.

bitClear()

void bitClear(qint4& pValue, qshort pBit)

Clears a bit in a value.

- **pValue** - The value to clear a bit in
- **pBit** - The bit index to clear

bitSet()

void bitSet(qint4& pValue, qshort pBit)

Sets a bit in a value.

- **pValue** - The value to set a bit in
- **pBit** - The bit index to set

bitSet()

void bitset(qint4& pValue, qshort pBit, qbool pState)

Alters the state of a bit in a value.

- **pValue** - The value to set a bit in
- **pBit** - The bit index to set
- **pState** - The new state for the bit index

bitTest()

qbool bitTest(qint4 pValue, qshort pBit)

Tests a bit in a value.

- **pValue** - The value to use for bit testing.
- **pBit** - The bit index to test
- **returns -** qtrue if the bit is set and qfalse if the bit is clear

Example:

```
qlong newValue = 28, oldValue = 28;
if ( bitTest(newValue,4 ) )
{
  bitClear( newValue,4 );
  if ( newValue==12 )
  {
    bitSet( newValue, 1 )
  }
}
if ( newValue==14 )
{
  bitSet( newValue,1, qfalse );
  bitSet( newValue,4, qtrue );
}
if ( newValue==oldValue )
{
  // all is OK.
}
```

ObjInst Functions

You can use the following functions in order to construct new instances of Omnis objects.

EXTobjinst()

qobjinst EXTobjinst(EXTCompInfo* pEci)

EXTobjinst constructs a new qobjinst (for use with EXTfldval::setObjInst) from the supplied EXTCompInfo structure. The new qobjinst is an empty external object which is associated with the external library which created it but it has no subtype.

- **pEci** – Pointer to an EXTCompInfo structure which Omnis uses to associate the object with the appropriate external library. EXTCompInfo member mCompId will be used as an identifier for that object.
- **returns** – Returns a new qobjinst pointer if successful, zero otherwise. ECOresetObjDetails can then be used to add properties and/or methods to this dynamic object.

```
// Example of returning a dynamic object to OMNIS
// First setup mCompId so when we are required to do processing later,
// during WndProc, we know what the object is!
pEci->mCompId = myObjectRef;
qobjinst myNewObj = EXTobjinst( pEci );
if ( myNewObj )
{ // Succeeded, now pass the new object to OMNIS (transferring ownership)
  EXTfldval RtnVal;
  RtnVal.setObjInst( myNewObj, qtrue );
  ECOaddParam( pEci, &RtnVal );
}
else
{ // Failed (usually because of lack of memory )
}
```

See also ECOresetObjDetails,EXTfldval::setObjInst

EXTobjinst()

qobjinst EXTobjinst(qobjinst pObjInst)

This EXTobjinst function duplicates the supplied qobjinst to return a new qobjinst pointer.

- **pObjInst** – qobjinst pointer to duplicate.
- **Returns** – returns a new qobjinst if successful, zero otherwise.

```
// Example of new operator for the supplied objinst
qobjinst myNewObj = EXTobjinst( sourceObjInst );
if ( myNewObj )
{ // Succeeded, now pass the new object to OMNIS (transferring ownership)
  EXTfldval RtnVal;
  RtnVal.setObjInst( myNewObj, qtrue );
  ECOaddParam( pEci, &RtnVal );
}
else
{ // Failed (usually because of lack of memory )
}
```

See also EXTfldval::setObjInst

EXTobjinst()

qobjinst EXTobjinst(qapp pApp,str255* pClassName)

This EXTobjinst function creates a new instance of an object from the specified class name.

- **pApp** – qapp pointer which is a unique pointer to the library in Omnis.
- **pClassName** – str255 pointer which contains the class to create.
- **Returns** – returns a new qobjinst if successful, zero otherwise.

```
// Example of constructing a new ‘oMy_OMNIS_Object’
// Get qapp from locpinst held in EXTCompInfo structure
qapp myLibraryApp = ECOgetApp( pEci->mInstLocp );
// Set up the classname from which to construct the new object
str255 myClassName("oMy_OMNIS_Object");
// Create the new object
qobjinst myNewObj = EXTobjinst( myLibraryApp, &myClassName );
if ( myNewObj )
{ // Succeeded, now pass the new object to OMNIS (transferring ownership)
  EXTfldval RtnVal;
  RtnVal.setObjInst( myNewObj, qtrue );
  ECOaddParam( pEci, &RtnVal );
}
else
{ // Failed. Maybe due to lack of memory or that
  // oMy_OMNIS_Object doesn’t exist in the specified qapp
}
```

See also EXTfldval::setObjInst, ECOgetApp

Chapter 3—strxxx Class Reference

The strxxx class gives your external components convenient ways to manipulate strings. Once your string is encapsulated inside the string class, it can be passed back and forth to OMNIS or have various string operations performed on it.

The string class is split into three real classes, each derived from a base class strxxx. You should not need to access the strxxx base class directly. Three classes are derived from strxxx: str15, str80, and str255. Each can hold the maximum number of characters as specified by the class name.

Characters in the string class are indexed using a range 1 to n. Index 0 is used to store the real length of the string.

Member Functions strxxx Class

strxxx::strxxx()

The strxxx class has various constructors called from the three derived classes.

strxxx::assign()

void strxxx::assign(const strxxx& pAssignFrom)

Assigns one strxxx class to another.

- **pAssignFrom** - The string to be copied into this object.

strxxx::compare()

void strxxx::compare(const strxxx& pCompare)

Compares two strings, this string and the string passed.

- **pCompare** - This string to compare against.

strxxx::concat()

void strxxx::concat(const strxxx& pNewString)

Concatenates two strings together.

- **pNewString** - String to be concatenated on to this string.

strxxx::concat()

void strxxx::concat(qchar pChar)

Concatenates a single character on to this string.

1. **pChar** - The character to be concatenated on to this string.

strxxx::concat()

void strxxx::concat(const strxxx& pString1, const strxxx& pString2) void strxxx::concat(const strxxx& pString1, const strxxx& pString2, const strxxx& pString3) void strxxx::concat(const strxxx& pString1, const strxxx& pString2, const strxxx& pString3, const strxxx& pString4)

Concatenates a group of strings together on to this string.

- **pString1**- String 1 to be concatenated.
- **pString2**- String 2 to be concatenated.
- **pString3**- String 3 to be concatenated.
- **pString4**- String 4 to be concatenated.

strxxx::copy()

void strxxx::copy(const strxxx& pExtractFrom, qshort pStart, qshort pLen)

Copies a ranges of characters from the passed string, and uses them to set the contents of this.

- **pExtractFrom** - The string to extract characters from.
- **pStart** - The starting index in **pExtractFrom.**
- **pExtractFrom** - The number of characters to copy from **pExtractFrom.**

strxxx::cString()

qchar* strxxx::cString()

Returns the address of a c-style string. This function converts this string into a c-style string first. A c-style string uses a null terminator, character 0x0 to represent the end of the strings data.

- **return** - The address to a c-style string.

strxxx::deleet()

void strxxx::deleet(qshort pPos, qshort pLen)

Deletes a range of characters from a starting point in the string. (Note it is named deleet so it does not interfere with the default C++ delete.)

- **pPos** - The starting index to delete from.
- **pLen** - The number of characters to be deleted.

strxxx::insert()

void strxxx::insert(const strxxx& pInsertString ,qshort pPos)

Inserts a string at an index position.

- **pInsertString** - The string to be inserted.
- **pPos** - The index at which to insert the string.

strxxx::insert()

void strxxx::insert(qchar pInsertChar ,qshort pPos)

Inserts a single character at an index position.

- **pInsertChar** - The character to be inserted.
- **pPos** - The index at which to insert the character.

strxxx::insertStr()

void strxxx::insertStr(const strxxx& pInsertString)

Searches the string for a '$' and inserts a sub-string **pInsertString** replacing the '$'.

- **pInsertString** - The string to be inserted.

strxxx::insertStr0()

void strxxx::insertStr0(const strxxx& pInsertString)

Similar to strxxx::insertStr(), except that it searches for the character 0x0 instead of '$'.

strxxx::length()

qshort strxxx::length()

Returns the length of the string stored in the object.

- **returns** - The length of the string.

strxxx::maxLength()

qshort strxxx::maxLength()

Returns the maximum length that can be stored in the string.

- **returns** - The maximum length of the string.

strxxx::operator !()

qbool strxxx::operator ! ()

Test is this string is not empty.

- **return** - Returns qtrue if the string contains some data.

strxxx::operator != ()

qbool strxxx::operator !=(const strxxx& pCompare)

Compares two strings.

1. **return** - qtrue if the strings do not match and qtrue if the strings are the same.

strxxx::operator [ ]()

qchar& strxxx::operator [ ] (qshort pIndex)

Returns the character from the string at the passed index.

- **pIndex** - The index to return a character from.
- **return** - The character from index **[pIndex].**

strxxx::operator <()

```
qbool strxxx::operator < ( const strxxx& pCompare )
```

Compares two strings.

- **return** - qtrue if this string is less than **pCompare**.

strxxx::operator <=()

```
qbool strxxx::operator <=( const strxxx& pCompare )
```

Compares two strings.

- **return** - qtrue if this string is less than or equal to **pCompare**.

strxxx::operator =()

```
void strxxx::operator = ( const strxxx& pNewString )
```

Assigns a string.

- **pNewString** - Assigned **pNewString** to this string.

strxxx::operator =(qniltype qnil)

```
void strxxx::operator =(qniltype qnil1)
```

Sets the length of the string to 0.

strxxx::operator ==()

```
qbool strxxx::operator ==( const strxxx& pCompare )
```

Compares two strings.

- **return** - qtrue if the strings match and qfalse if the strings are different.

strxxx::operator >()

```
qbool strxxx::operator >( const strxxx& pCompare )
```

Compares two strings.

- **return** - qtrue if this string is greater than **pCompare**.

strxxx::operator >=()

qbool strxxx::operator >=(const strxxx& pCompare)

Compares two strings.

- **return** - qtrue if this string is greater than or equal to **pCompare.**

strxxx::pos()

qshort strxxx::pos(const strxxx& pFind)

Looks for the string pFind inside this.

- **pFind** - The string to search for.
- **returns** - The index if the string is found. 0 is returned if the string is not found.

strxxx::pos()

qshort strxxx::pos(qchar pFindChar)

Looks for the first occurrence of pFindChar inside this.

- **pFindChar** - The character to search for.
- **returns** - The index if the string is found. 0 is returned if the string is not found.

strxxx::pString()

qchar* strxxx::pString()

Returns the address of a Pascal-style string. This function converts this string into a Pascal string first. A Pascal-style string uses the first byte of the string, index 0 as a length byte. The following characters, index 1 to n, are string data.

- **return** - The address to a Pascal string.

strxxx::repWith0()

void strxxx::repWith0()

Replaces all '$' characters with a 0x0 character.

strxxx::upps()

void strxxx::upps()

Converts this to uppercase.

strxxx::uprCmp()

void strxxx::uprCmp(const strxxx& pCompare)

Performs a case-insensitive comparison.

- **pCompare** - This string to compare against.
- **return** - This function returns:
  0 if the strings match.
  1 if this string is greater than pCompare.
  -1 if this string is less than pCompare.

Member Functions str15 Class

str15::str15()

str15::str15()

Constructor for an empty str15 string class.

str15::str15()

str15::str15(const str15& pCopyFrom)

Constructor for a new str15 object duplicating the contents of another str15 object.

str15::str15()

str15::str15(const strxxx& pCopyFrom)

Constructor for a new str15 object duplicating the contents of another strxxx object.

- **pCopyFrom** - The string to copy the initial value from to a maximum of 15 characters.

str15::str15()

str15::str15(const void* pData)

Constructor for a new str15 object setting an initial value.

- **pData** - This must be a null-terminated, c-style string. The new string has stores a maximum of 15 characters.

str15::str15()

str15::str15(qshort pLen, const void* pData)

Constructor for a new str15 object setting an initial value.

- **pLen** - The number of characters to copy from pData.
- **pData** - The source of the initial data for the new string.

str15::str15()

str15::str15(qchar pChar)

Constructor for a new str15 object setting an initial value.

- **pChar** - The initial value for the new string.

Member Functions str80 Class

str80::str80()

str80::str80()

Constructor for an empty str80 string class.

str80::str80()

str80::str80(const str80& pCopyFrom)

Constructor for a new str80 object duplicating the contents of another str80 object.

str80::str80()

str80::str80(const strxxx& pCopyFrom)

Constructor for a new str80 object duplicating the contents of another strxxx object.

- **pCopyFrom** - The string to copy the initial value from to a maximum of 80 characters.

str80::str80()

str80::str80(const void* pData)

Constructor for a new str80 object setting an initial value.

1. **pData** - This must be a null-terminated, c-style string. The new string has a maximum of 80 characters.

str80::str80()

str80::str80(qshort pLen, const void* pData)

Constructor for a new str80 object setting an initial value.

- **pLen** - The number of character to copy from pData.
- **pData** - The source of the initial data for the new string.

str80::str80()

str80::str80(qchar pChar)

Constructor for a new str80 object setting an initial value.

Member Functions str255 Class

str255::str255()

str255::str255()

Constructor for an empty str255 string class.

str255::str255()

str255::str255(const str255& pCopyFrom)

Constructor for a new str255 object duplicating the contents of another str255 object.

str255::str255()

str255::str255(const strxxx& pCopyFrom)

Constructor for a new str255 object duplicating the contents of another strxxx object.

1. **pCopyFrom** - The string to copy the initial value from to a maximum of 255 characters.

str255::str255()

str255::str255(const void* pData)

Constructor for a new str255 object setting an initial value.

- **pData** - This must be a null-terminated, c-style string. The new string has a maximum of 255 characters.

str255::str255()

str255::str255(qshort pLen, const void* pData)

Constructor for a new str255 object setting an initial value.

- **pLen** - The number of character to copy from pData.
- **pData** - The source of the initial data for the new string.

str255::str255()

str255::str255(qchar pChar)

Constructor for a new str255 object setting an initial value.

- **pChar** - The initial value for the new string.

Other Functions

qlongToString()

void qlongToString(qlong pVal, strxxx& pString)

Converts a numeric value into a string value.

- **pVal** - The number to convert.
- **pString** - The string to receive the converted result.

qrealToString()

void qrealToString(qreal pVal, qshort pDecimalPlace, strxxx& pString, qshort pSigDecimalPlace)

Converts a numeric value into a string value.

- **pVal** - The number to convert.
- **pDecimalPlace** - The number of decimal places to convert to.
- **pString** - The string to contain the converted result.
- **pSigDecimalPlace** - This is the number of significant digits the string is converted to if the decimal places passed is larger than or equal to 24.

stringToQlong()

qbool stringToQlong(const strxxx& pString, qlong& pVal)

Converts a string into a numeric value.

- **pString** - The string to convert.
- **pVal** - The numeric result.
- **returns** - qtrue if the string could be converted, and qfalse if the string could not be converted.

stringToQreal()

```
qbool stringToQreal(const strxxx& pString, qreal& pVal, qshort& pDecimalPlace )
```

Converts a string into a numeric value.

- **pString** - The string to convert.
- **pVal** - The numeric result.
- **pDecimalPlace** - Returns the number of decimal the converted value has.
- **returns** - qtrue if the string could be converted, and qfalse if the string could not be converted.

lowC()

```
qchar lowC( qchar pChar )
```

Converts a single character to lowercase.

- **pChar** - The character to be converted.
- **returns** - The new lowercase character.

uppC()

```
qchar uppC( qchar pChar )
```

Converts a single character to uppercase.

- **pString** - The character to be converted.
- **returns** - The new uppercase character.

uppC()

```
void uppC( qchar* pAddress, qlong pLen )
```

Converts a range of characters to uppercase.

- **pAddress** - The address of a buffer of characters to be uppercased.
- **pLen** - The number of characters to uppercase.

uprCmp()

qshort uprCmp(qchar* pAddress, qchar* pAddress2, qlong pLen)

Performs a case insensitive comparison on two buffers for a specified length.

- **pAddress1** - The address to a buffer of characters.
- **pAddress2** - The address to a buffer of characters.
- **pLen** - The number of characters to uppercase in both strings.
- **return** - This function returns:
  0 if the strings match.
  1 if this string is greater than pCompare.
  -1 if this string is less than pCompare.

Chapter 4—Unicode Character Conversion

Introduction

This section provides the reference information you need to convert your Omnis External Components to Unicode so they will run in Omnis Studio 5.0, which is a Unicode-only release. The information here is also useful for developers using Studio 4.x versions who wish to create External Components for the Unicode version of Studio 4.x.

When building Unicode components for Omnis Studio, the following pre-processor definitions should be added to the project settings: ***isunicode***, ***UNICODE*** and ***_UNICODE***. These enable wide character versions of certain system functions and Omnis API calls.

To maintain backwards compatibility with the non-Unicode version of Omnis Studio, you should create separate targets for the Unicode-Debug and Unicode-Release versions of your components.

In this way, you can maintain a single set of source files for both Unicode and non-Unicode targets by making use of conditional-compilation statements where necessary, i.e.

```
#ifdef isunicode
  // Unicode specific code here
#else
  // Non-Unicode specific code here
#endif
```

In the Unicode version of Omnis Studio, all character data exchanged with external components should use the UTF-32 encoding (4 bytes per character).

There are a number of utility classes and helper functions provided by the component library and these can be found in chrbasic.he, omstring.h & omstring.c.

Unicode Data Types

The following data types are used by the component library for handling character data.

qchar

When *isunicode* is defined, the qchar data type is defined as unsigned long (4 bytes) and is used to contain UTF-32 data. For non-Unicode targets, qchar defaults to unsigned char.

qoschar

When *isunicode* is defined, the qoschar data type is set to match the operating system API encoding. For Windows and Mac OS X, this is UTF-16. For Linux this is UTF-8. Thus for Windows and Mac OS X, qoschar is defined as unsigned short and for Linux, qoschar is defined as char. When *isunicode* is not defined, qoschar is defined as char.

qbyte

The qbyte data type is always defined as unsigned char and is used for binary data and to distinguish ASCII character data from Unicode data.

Utility Classes

CHRconvToOs

This class converts a string of qchar data to the operating system API encoding.

CHRconvToOs::CHRconvToOs()

```
CHRconvToOs::CHRconvToOs(strxxx &pString)
```

Creates a CHRconvToOs object from the supplied strxxx object.

CHRconvToOs::CHRconvToOs()

```
CHRconvToOs::CHRconvToOs(qchar *pAdd, qlong pLen)
```

Creates a CHRconvToOs object from the supplied qchar character buffer.

- pAdd point to the buffer containing UTF-32 data
- pLen is the length of the source data in characters

CHRconvToOs::CHRconvToOs()

```
CHRconvToOs::CHRconvToOs(qchar *pAdd)
```

Creates a CHRconvToOs object from the supplied qchar buffer. The buffer must be null-terminated.

CHRconvToOs::convToOs()

```
qlong CHRconvToOs::convToOs(qchar *pAdd, qlong pLen, qoschar
*pDestBuffer)
```

Converts the supplied qchar buffer to qoschars, returning the result in pDestBuffer.

- pAdd is the source buffer containing UTF-32 data
- pLen is the length of the source data in characters
- pDestBuffer is a user-allocated destination buffer, which must be large enough to accommodate the converted data.

CHRconvToOs::dataPtr()

```
qoschar* CHRconvToOs::dataPtr()
```

Returns a pointer to the converted data. The memory associated with this pointer is managed by the object.

CHRconvToOs::len()

```
qlong CHRconvToOs::len()
```

Returns the length in bytes of the converted data contained inside the object.

CHRconvFromOs

This class converts a string of characters from the operating system encoding to the Omnis internal encoding (qchars).

CHRconvFromOs::CHRconvFromOs()

```
CHRconvFromOs::CHRconvFromOs(qoschar *pAdd, qlong pLen)
```

Creates a CHRconvFromOs object from a buffer of qoschars.

- pLen is the length in characters of the source data

CHRconvFromOs::CHRconvFromOs()

```
CHRconvFromOs::CHRconvFromOs(qoschar *pAdd)
```

Creates a CHRconvFromOs object from a null-terminated string of qoschars, i.e. terminated by two consecutive null bytes when qoschar is defined as unsigned short.

CHRconvFromOs::CHRconvFromOs() Mac OS X only

```
CHRconvFromOs::CHRconvFromOs(CFStringRef pCFStringRef)
```

Creates a CHRconvFromOs object from the supplied CFStringRef parameter.

CHRconvFromOs::convFromOs()

```
qlong CHRconvFromOs::convFromOs(qoschar *pSrcAdd, qlong pSrcLen,
qchar *pDestAdd, qlong pDestMaxLen)
```

Converts the supplied source data, writing the converted data into pDestAdd. Returns the number of characters converted.

- pSrcAdd points to the buffer containing the source data
- pSrcLen is the length of the source data in characters
- pDestAdd points to the user-allocated destination buffer which must be large enough to contain the converted data
- pDestMaxLen is the maximum length of the destination buffer in characters

CHRconvFromOs::pascalStringFromOs()

```
void CHRconvFromOs::pascalStringFromOs(qoschar *pSrcAdd, qlong pSrcLen,
qchar *pDestStr, qlong pDestMaxLen)
```

Converts the supplied source data, writing the converted data into pDestStr. Character position zero of the converted data contains the length in characters (0-255).

- pSrcAdd points to a buffer containing the source data
- pSrcLen is the length of the source data in characters
- pDestStr is a user allocated buffer which must be large enough to contain the converted data
- pDestStr is the maximum size of the destination buffer in character units.

CHRconvFromOs::dataPtr()

```
qchar* CHRconvFromOs::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromOs::len()

```
qlong CHRconvFromOs::len()
```

Returns the length of the converted data in character units.

CHRconvToAscii

This class converts a string of qchar data to ASCII bytes and assumes that the source data contains 7-bit ASCII compatible characters.

CHRconvToAscii::CHRconvToAscii()

```
CHRconvToAscii::CHRconvToAscii(strxxx &pString)
```

Creates a CHRconvToAscii object from the supplied strxxx object.

CHRconvToAscii::dataPtr()

```
char* CHRconvToAscii::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToAscii::len()

```
qlong CHRconvToAscii::len()
```

Returns the length of the converted data.

CHRunicode

This class provides conversion functions between different Unicode encodings.

CHRunicode::utf8EncodeChar()

```
qlong CHRunicode::utf8EncodeChar(qulong pChar, qbyte *pOutUtf8, qbool
pCanFatal)
```

Encodes a single character as UTF-8, and returns the encoded length in bytes.

- pChar is a single UTF-32 character value
- pOutUtf8 is a user-allocated destination buffer which must be at least 4 bytes in size
- pCanFatal allows the object to generate a fatal error on conversion failure

CHRunicode::getUtf8EncodedChar()

```
qulong CHRunicode::getUtf8EncodedChar(qbyte *pBuffer, qlong pInLen,
qlong &pIndex, qbool pAlwaysUTF8 = qfalse)
```

Gets a UTF-8 encoded character from the source buffer. Returns the converted character value as UTF-32.

- pBuffer points to the address of a UTF-8 character string
- pInLen is the length in bytes of the entire UTF-8 string
- pIndex is the byte offset from pBuffer to the start of the UTF-8 character
- pAlwaysUTF8 has no effect when isunicode is defined. Passing the value qfalse when isunicode is not defined maps the UTF-32 character to the Omnis 8 bit character set (or 0xc0 (inverted question mark) if the UTF-32 is not in the Omnis 8 bit character set)

CHRunicode::charToUtf8()

```
qlong CHRunicode::charToUtf8(qchar *pInChar, qlong pInLen, qbyte
*pOutUtf8)
```

Converts a string of Unicode characters to UTF-8. The output buffer length must be >= UTF8_MAX_BYTES_PER_CHAR*pInLen bytes. Returns the encoded length.

- pInChar points to a buffer of UTF-32 characters
- pInLen is the size of the source buffer in character units
- pOutUtf8 points to a user-allocated destination buffer

CHRunicode::utf8ToChar()

```
qlong CHRunicode::utf8ToChar(qbyte *pInUtf8, qlong pInLen, qchar
*pOutChar, qlong pOutBufLen = 0)
```

Converts UTF-8 encoded data to Unicode (UTF-32). Returns the length of the converted data in character units.

- pInUtf8 points to the source buffer
- pInLen is the length of the source data in bytes
- pOutChar points to a user-allocated destination buffer
- pOutBufLen is the maximum size of the destination buffer in characters

CHRunicode::convertOmnisToUnicode()

```
void CHRunicode::convertOmnisToUnicode(qbyte *pOmnisDataChars, qlong
pLen, strxxx &pDestStr)
```

Converts Omnis non-Unicode data, and stores the result in pDestStr.

- pOmnisDataChars points to a string of 8 bit data in the Omnis character set
- pLen is the length of the source data
- pDestStr is a strxxx object passed by reference

CHRunicode::convertOmnisToUnicode()

```
void CHRunicode::convertOmnisToUnicode(qbyte *pOmnisDataChars, qlong
pLen, handle &pDest)
```

Converts Omnis non-Unicode data, and stores result in handle memory.

- pOmnisDataChars points to a string of 8 bit data in the Omnis character set
- pLen is the length of the source data
- pDestStr is a handle passed by reference

CHRunicode::convertOmnisToUnicode()

```
void CHRunicode::convertOmnisToUnicode(qbyte *pOmnisDataChars, qlong
pLen, qchar *pDest, qlong pDestBufLen = 0)
```

Converts Omnis non-Unicode data, and stores the result in pDest

- pOmnisDataChars points to a string of 8 bit data in the Omnis character set
- pLen is the length of the source data
- pDestStr points to a user-allocated buffer; large enough to contain the converted data
- If supplied, pDestBufLen specifies the maximum length of the destination buffer in bytes

CHRunicode::encodedCharactersToChar()

```
qlong CHRunicode::encodedCharactersToChar(qbool pAlwaysUtf8, qbyte
*pInEncChar, qlong pInLen, qchar *pOutChar, qlong pOutBufLen = 0)
```

Converts UTF-8/Omnis non-Unicode characters to UTF-32/qchar. Returns the length of the converted data in character units.

- pAlwaysUtf8 specifies that the source data is UTF-8 encoded data. If qfalse, it is assumed to be in the Omnis 8 bit character set
- pInEncChar points to a buffer containing the source data
- pInLen is the length in bytes of the source data
- pOutChar is a user-allocated destination buffer
- If supplied, pOutBufLen specifies the maximum length of the destination buffer in bytes

CHRunicode::charToEncodedCharacters()

```
qlong CHRunicode::charToEncodedCharacters(qbool pAlwaysUtf8, qchar
*pInChar, qlong pInLen, qbyte *pOutEncChar)
```

Converts qchar characters to UTF-8/Omnis characters. Returns the length in bytes of the converted data.

- pAlwaysUtf8 specifies that UTF-8 should be generated as the output; otherwise, the data is converted to the Omnis 8 bit character set
- pInChar points to a buffer containing the source data
- pInLen is the length of the source data in character units
- pOutEncChar is a user-allocated buffer large enough to contain the converted data

CHRunicode::setEncodingMode()

```
void CHRunicode::setEncodingMode(qbool pUtf8)
```

Sets the encoding mode for encodedCharactersToChar and charToEncodedCharacters (UTF-8 or Omnis).

If qtrue, this setting overrides *pAlwaysUtf8* and specifies that conversion to/from UTF-8 is required.

CHRunicode::isBigEndian()

```
qbool CHRunicode::isBigEndian()
```

Returns qtrue if the *ordermsb* preprocessor definition was used (i.e. if multi-byte characters are stored with the most significant byte first), qfalse otherwise.

CHRunicode::is7Bit()

```
qbool CHRunicode::is7Bit(qchar *pAdd, qlong pLen)
```

Returns qtrue if the source data contains entirely 7-bit data (such that UTF-8 and Omnis encodings are identical), qfalse otherwise.

- pAdd points to a buffer containing UTF-32 data
- pLen is the length of the source data in character units

CHRunicode::isUtf8Data()

```
qbool CHRunicode::isUtf8Data(qbyte *pAdd, qlong pLen)
```

Returns qtrue if the data satisfies the UTF-8 encoding rules. Note that this does not preclude the possibility that a non-UTF-8 string may pass this check where the source string contains extended ASCII characters and these coincide with UTF-8 encoding bytes.

- pAdd points to a buffer containing 8 bit/UTF-8 data
- pLen is the length of the source data in bytes

CHRconvToUtf16

This class converts a string of UTF-8 data to the UTF-16 encoding.

CHRconvToUtf16:: CHRconvToUtf16()

```
CHRconvToUtf16::CHRconvToUtf16(qbyte *pAdd, qlong pLen, qbool pSwap =
qfalse, qbool pAddBom = qfalse)
```

Creates a CHRconvToUtf16 object from the supplied source data.

- pAdd points to a buffer containing UTF-8 data
- pLen is the length of the source data in bytes
- pSwap specifies that the output byte ordering should be reversed (See CHRunicode::isBigEndian())
- pAddBom specifies that an additional Byte-Order-Marker should be placed at element zero of the converted data

CHRconvToUtf16::dataPtr()

```
UChar * CHRconvToUtf16::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToUtf16::len()

```
qlong CHRconvToUtf16::len()
```

Returns the length of the converted data in bytes.

CHRconvFromUtf16

This class converts a string of UTF-16 encoded data to UTF-8

CHRconvFromUtf16:: CHRconvFromUtf16()

```
CHRconvFromUtf16::CHRconvFromUtf16(UChar *pAdd, qlong pLen, qbool
pSwap = qfalse)
```

Creates a CHRconvFromUtf16 object from the supplied source data.

- pAdd points to a buffer containing UTF-16 encoded data
- pLen is the length of the source data in bytes
- pSwap specifies that the byte ordering of the source data is opposite to the platform default

CHRconvFromUtf16::dataPtr()

```
qbyte* CHRconvFromUtf16::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromUtf16::len()

qlong CHRconvFromUtf16::len()

Returns the length of the converted data in bytes.

CHRconvToBytes

This class converts a character buffer to a stream of bytes. For Unicode targets, the characters are encoded using UTF-8; in the non-Unicode version, the characters are unchanged.

CHRconvToBytes::CHRconvToBytes()

CHRconvToBytes::CHRconvToBytes (qchar *pAdd, qlong pLen)

Creates a CHRconvToBytes object from the supplied source data.

- pLen is the length of the data pointed to by pAdd in character units

CHRconvToBytes::CHRconvToBytes()

CHRconvToBytes::CHRconvToBytes (qchar *pAdd)

Creates a CHRconvToBytes object from the supplied source data.

- pAdd points to a null-terminated string of qchars

CHRconvToBytes::dataPtr()

qbyte * CHRconvToBytes::dataPtr()

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToBytes::len()

qlong CHRconvToBytes::len()

Returns the length of the converted data in bytes.

CHRconvToBytes::makeCanonical() Mac OS X only

void CHRconvToBytes::makeCanonical()

Makes the UTF-8 representation canonical, which is the required representation for Mac OS X file system calls. The canonical representation decomposes all composed characters (e.g. e+acute accent) into their components (e.g. the letter e, followed by acute accent symbol).

CHRconvToBytes::makeUtf8PascalString()

```
void CHRconvToBytes::makeUtf8PascalString(qchar *pAdd, qlong pLen, qbyte
*pPascalString, qlong pPascalStringBufferLength)
```

Converts the supplied source data to UTF-8 with a length byte at element zero , hence the length of the source data is limited to 255 characters.

- pAdd points to a buffer containing the source data
- pLen is the length of the source data in characters. 255 maximum
- pPascalString points to a user-allocated destination buffer
- pPascalStringBufferLength is the maximum size of the destination buffer in bytes

CHRconvFromBytes

This class converts a buffer of 8 bit/UTF-8 encoded characters to qchars . For Unicode targets, the source data can be UTF-8. For non-Unicode targets, the characters are unchanged.

CHRconvFromBytes::CHRconvFromBytes()

```
CHRconvFromBytes::CHRconvFromBytes (qbyte *pAdd, qlong pLen)
```

Creates a CHRconvFromBytes object from the supplied source data.

- pAdd points to a buffer containing the 8 bit/UTF-8 data
- pLen is the length of the source data in bytes

CHRconvFromBytes::CHRconvFromBytes()

```
CHRconvFromBytes::CHRconvFromBytes (qbyte *pAdd)
```

Creates a CHRconvFromBytes object from the supplied source data.

- pAdd points to a null-terminated string of 8 bit/UTF-8 data

CHRconvFromBytes::dataPtr()

```
qchar * CHRconvFromBytes::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromBytes::len()

```
qlong CHRconvFromBytes::len()
```

Returns the length of the converted data in character units.

CHRconvFromLatin1ApiBytes

This class converts a string of Windows Latin 1 bytes to qchars.

CHRconvFromLatin1ApiBytes::CHRconvFromLatin1ApiBytes()

```
CHRconvFromLatin1ApiBytes::CHRconvFromLatin1ApiBytes(qbyte *pAdd,
qlong pLen)
```

Creates a CHRconvFromLatin1ApiBytes object from the supplied source data.

- pAdd points to a buffer containing the Windows Latin 1 encoded data
- pLen is the length of the source data in bytes

CHRconvFromLatin1ApiBytes::CHRconvFromLatin1ApiBytes()

```
CHRconvFromLatin1ApiBytes::CHRconvFromLatin1ApiBytes(qbyte *pAdd)
```

Creates a CHRconvFromLatin1ApiBytes object from the supplied source data.

- pAdd points to a null terminated string of Windows Latin 1 encoded data

CHRconvFromLatin1ApiBytes::dataPtr()

```
qchar * CHRconvFromLatin1ApiBytes::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromLatin1ApiBytes::len()

```
qlong CHRconvFromLatin1ApiBytes::len()
```

Returns the length of the converted data in character units.

CHRconvToLatin1ApiBytes

This class converts a string of qchar data to the Windows/Latin1 code page.

CHRconvToLatin1ApiBytes::CHRconvToLatin1ApiBytes()

```
CHRconvToLatin1ApiBytes::CHRconvToLatin1ApiBytes(qchar *pAdd, qlong
pLen)
```

Creates a CHRconvToLatin1ApiBytes object from the supplied source data.

- pAdd points to the source buffer containing qchar data
- pLen is the length of the source data in character units

CHRconvToLatin1ApiBytes::CHRconvToLatin1ApiBytes()

```
CHRconvToLatin1ApiBytes::CHRconvToLatin1ApiBytes(qchar *pAdd)
```

Creates a CHRconvToLatin1ApiBytes object from the supplied source data.

- pAdd points to a null terminated string of qchar data

CHRconvToLatin1ApiBytes::dataPtr()

```
qbyte * CHRconvToLatin1ApiBytes::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToLatin1ApiBytes::len()

```
qlong CHRconvToLatin1ApiBytes::len()
```

Returns the length of the converted data in bytes.

CHRconvToEncodedCharacters

This class converts a string of qchar data to UTF-8 or Omnis 8 bit data.

CHRconvToEncodedCharacters::CHRconvToEncodedCharacters()

```
CHRconvToEncodedCharacters::CHRconvToEncodedCharacters(qbool
pAlwaysUtf8, qchar *pAdd, qlong pLen, csettype pSrcCset = csetOdata)
```

Creates a CHRconvToEncodedCharacters object from the supplied source data.

- pAlwaysUtf8 specifies that the source data always contains Unicode characters
- pAdd points to a buffer containing rthe source data
- pLen is the length of the source data in character units
- For non-Unicode data, pSrcCset specifies the Omnis character set used for the source data. Character set constants are defined in basics.h

CHRconvToEncodedCharacters::dataPtr()

```
qbyte * CHRconvToEncodedCharacters::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToEncodedCharacters::len()

```
qlong CHRconvToEncodedCharacters::len()
```

Returns the length of the converted data in bytes.

CHRconvToEncodedCharacters::makeCanonical() Mac OS X only

```
void CHRconvToEncodedCharacters ::makeCanonical()
```

Makes the UTF-8 representation canonical, for MacOSX file system calls. Assumes that the buffer contains UTF-8 data.

CHRconvFromEncodedCharacters

This class converts a string of Omnis 8 bit or UTF-8 encoded data to qchars.

CHRconvFromEncodedCharacters::CHRconvFromEncodedCharacters()

```
CHRconvFromEncodedCharacters::CHRconvFromEncodedCharacters (qbool
pAlwaysUtf8, qbyte *pAdd, qlong pLen, csettype pDestCset = csetOdata)
```

Creates a CHRconvFromEncodedCharacters from the supplied source data.

- pAlwaysUtf8 specifies that conversion from UTF-8 will definitely be required
- pAdd points to a buffer containing the source data
- pLen contains the length of the source data in bytes
- pDestCset specifies the Omnis character set to be assumed when handling ASCII data. Omnis character set constants are defined in basics.h

CHRconvFromEncodedCharacters::CHRconvFromEncodedCharacters()

```
CHRconvFromEncodedCharacters::CHRconvFromEncodedCharacters (qbool
pAlwaysUtf8, qbyte *pAdd)
```

Creates a CHRconvFromEncodedCharacters from the supplied source data.

- pAdd points to a null terminated string of UTF-8 characters

CHRconvFromEncodedCharacters::dataPtr()

```
qchar * CHRconvFromEncodedCharacters ::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromEncodedCharacters::len()

```
qlong CHRconvFromEncodedCharacters::len()
```

Returns the length of the converted data in character units.

CHRconvToOmnis

This class converts a string of qchar data to the 8 bit Omnis character set (csetOdata). No conversion is performed for non-Unicode targets.

CHRconvToOmnis:: CHRconvToOmnis()

```
CHRconvToOmnis::CHRconvToOmnis(qchar *pAdd, qlong pLen)
```

Creates a CHRconvToOmnis object from the supplied source data.

- pAdd points to a buffer containing the source data
- pLen contains the length of the source data in character units

CHRconvToOmnis::dataPtr()

```
qbyte * CHRconvToOmnis::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToOmnis::len()

```
qlong CHRconvToOmnis::len()
```

Returns the length of the converted data in bytes.

CHRconvFromOmnis

This class converts a string of 8 bit Omnis character set data to qchars. The source data is assumed to be from the Omnis character set (csetOdata). No conversion is performed for non-Unicode targets.

CHRconvFromOmnis::CHRconvFromOmnis()

```
CHRconvFromOmnis::CHRconvFromOmnis(qbyte *pAdd, qlong pLen)
```

Creates a CHRconvFromOmnis object from the supplied source data.

- pAdd points to a buffer containing the source data
- pLen contains the length of the source data in bytes

CHRconvFromOmnis::dataPtr()

```
qchar * CHRconvFromOmnis ::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromOmnis::len()

```
qlong CHRconvFromOmnis::len()
```

Returns the length of the converted data in character units.

CHRconvToUniChar

This class converts from qchar to UniChar (16 bit Unicode).

CHRconvToUniChar::CHRconvToUniChar() Mac OS X only

```
CHRconvToUniChar::CHRconvToUniChar()
```

Creates an empty CHRconvToUniChar object for subsequent initialisation.

CHRconvToUniChar::set() Mac OS X only

```
void CHRconvToUniChar::set(qchar *pAdd, qlong pLen)
```

Initialises the CHRconvToUniChar object from the supplied source data.

- pAdd points to a buffer containing the source data (qchars)
- pLen contains the length of the source data in character units

CHRconvToUniChar::CHRconvToUniChar()

```
CHRconvToUniChar::CHRconvToUniChar(qchar *pAdd, qlong pLen)
```

Creates a CHRconvToUniChar using the supplied source data. The source data must contain characters in the csetApi character set.

- pAdd points to a buffer containing the source data
- pLen contains the length of the source data in character units

CHRconvToUniChar::dataPtr()

```
UniChar * CHRconvToUniChar ::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToUniChar::len()

```
qlong CHRconvToUniChar::len()
```

Returns the length of the converted data in UniChar units.

CHRconvFromCodePage

This class converts a string of 8 bit encoded character data in the specified code page to qchars. Code page constants (*preUniType ...)* can be found in dmconst.he

CHRconvFromCodePage::CHRconvFromCodePage()

```
CHRconvFromCodePage::CHRconvFromCodePage(preconst pCodePage,
qbyte *pAdd, qlong pLen)
```

Creates a CHRconvFromCodePage object from the supplied source data.

- pCodePage specifies the code page used by the source data
- pAdd points to a buffer containing rthe source data
- pLen contains the length of the source data in bytes

CHRconvFromCodePage::dataPtr()

```
qchar * CHRconvFromCodePage::dataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromCodePage::len()

```
qlong CHRconvFromCodePage::len()
```

Returns the length of the converted data in character units.

CHRconvFromCodePage::codePageOk()

```
qbool CHRconvFromCodePage::codePageOk()
```

Returns qtrue if the object successfully retrieved the specified code page information, qfalse if the specified code page is not supported.

CHRconvFromCodePage::getCodePage()

```
qushort * CHRconvFromCodePage::getCodePage(preconst pCodePage)
```

Returns a code page array of 256 unsigned shorts that are used to provide the mapping from the code page to UTF-32. Each code page has its own mapping indexed by the 8 bit data values for the code page.

CHRconvToCodePage

This class converts a string of qchars the specified 8 bit code page. Source characters are assumed to be from the specified code page and are mapped accordingly. Any characters not present in the specified code page are mapped to '.'.

CHRconvToCodePage::CHRconvToCodePage()

```
CHRconvToCodePage::CHRconvToCodePage(preconst pCodePage, qchar
*pAdd, qlong pLen)
```

Creates a CHRconvToCodePage object from the supplied source data.

- pCodePage specifies the destination code page to be assumed. See dmconst.he for a list of *preUniType...* constants.
- pAdd points to a buffer containing the source data
- pLen contains the length of the source data in character units

CHRconvToCodePage::dataPtr()

qbyte * CHRconvToCodePage::dataPtr()

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToCodePage::len()

qlong CHRconvToCodePage::len()

Returns the length of the converted ASCII data in bytes.

CHRconvToCodePage::codePageOk()

qbool CHRconvToCodePage::codePageOk()

Returns qtrue if the object successfully retrieved the specified code page information, qfalse if the specified code page is not supported.

CHRconvToCodePage::getCodePage()

qbyte * CHRconvToCodePage::getCodePage(preconst pCodePage)

Returns the reverse code page mapping table; an array which is indexed by Unicode character values. The first 4 bytes of the array (cast to a long) indicate the number of significant bytes in the array. Unicode characters past the end of the array do not exist in the code page, and are mapped as a dot.

CHRconvFromUnicodeEncoding

This class converts a string of data from the specified encoding to the Omnis internal encoding. The encoding is specified using one of the *preUniType...* constants defined in dmconst.he

CHRconvFromUnicodeEncoding::CHRconvFromUnicodeEncoding()

CHRconvFromUnicodeEncoding::CHRconvFromUnicodeEncoding(preconst pReadEncoding, qbyte *pData, qlong pByteLen)

Creates a CHRconvFromUnicodeEncoding object from the supplied source data.

- pReadEncoding specifies the encoding of the source data, for example; *preUniTypeNativeCharacters*
- pData points to a buffer containing the source data (cast as qbyte *)
- pLen specifies the length of the source data in bytes

CHRconvFromUnicodeEncoding::isChar()

qbool CHRconvFromUnicodeEncoding::isChar()

Returns qtrue if the data after conversion is character data as opposed to binary data.

CHRconvFromUnicodeEncoding::charDataPtr()

```
qchar * CHRconvFromUnicodeEncoding::charDataPtr()
```

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromUnicodeEncoding::charLen()

```
qlong CHRconvFromUnicodeEncoding::charLen()
```

Returns the length of the converted data in character units.

CHRconvFromUnicodeEncoding::dataPtr()

```
qbyte * CHRconvFromUnicodeEncoding::dataPtr()
```

Returns a pointer to the raw converted data (cast as qbytes), the memory for which is managed by the object.

CHRconvFromUnicodeEncoding::len()

```
qlong CHRconvFromUnicodeEncoding::len()
```

Returns the length of the converted data in bytes.

CHRconvFromUnicodeEncoding::getCset()

```
Csettype CHRconvFromUnicodeEncoding::getCset()
```

Returns a *preUniType...* constant representing the character set used to perform the conversion.

CHRconvToUnicodeEncoding

This class converts a string of qchars to the specified Unicode encoding. The encoding is specified using one of the *preUniType* constants defined by dmconst.he. Character data for the non-Unicode version must be in the Omnis character set (except when writing native characters or binary data).

CHRconvToUnicodeEncoding::CHRconvToUnicodeEncoding()

```
CHRconvToUnicodeEncoding::CHRconvToUnicodeEncoding(preconst
pWriteEncoding, qbyte *pData, qlong pByteLen, qbool pAddBom = qtrue)
```

Creates a CHRconvToUnicodeEncoding object from the supplied source data.

- pWriteEncoding specifies the target encoding
- pData is a pointer to the source data (cast as qbyte *)
- pByteLen specifies the length of the source data in bytes.
- pAddBom specifies that element zero of the output should contain a Byte-Order-Marker, used for example when writing Unicode data to external files.

CHRconvToUnicodeEncoding::dataPtr()

qbyte * CHRconvToUnicodeEncoding::dataPtr()

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvToUnicodeEncoding::len()

qlong CHRconvToUnicodeEncoding::len()

Returns the length of the converted data in bytes.

CHRconvToUtf32FromChar

Intended for use with non-Unicode targets, this class operates on a string of qchar data, converting it to the UTF-32 encoding.

CHRconvToUtf32FromChar::CHRconvToUtf32FromChar()

CHRconvToUtf32FromChar::CHRconvToUtf32FromChar(qchar *pData, qlong pLen, qbool pOppositeEndian, qbool pAddBom = qfalse)

Creates a CHRconvToUtf32FromChar object from the supplied source data.

- pData is a pointer to the source data
- pLen specifies the length of the source data in character units
- pOppositeEndian specifies that the byte-endian order of the output characters should be the opposite of the platform default
- pAddBom specifies that a Byte-Order-Marker should be placed at element zero of the converted data, used for example when writing Unicode data to external files

CHRconvToUtf32FromChar::dataPtr()

U32Char * CHRconvToUtf32FromChar::dataPtr()

Returns a pointer to the converted UTF-32 data, the memory for which is managed by the object.

CHRconvToUtf32FromChar::len()

qlong CHRconvToUtf32FromChar::len()

Returns the length of the converted data in charcter units.

CHRconvFromUtf32ToChar

This class operates on a string of encoded UTF-32 data, stripping out any Byte-Order-Marker and optionally reversing the byte-endian order. Intended for use with non-Unicode targets.

CHRconvFromUtf32ToChar::CHRconvFromUtf32ToChar()

CHRconvFromUtf32ToChar ::CHRconvFromUtf32ToChar(U32Char *pData, qlong pLen, qbool pOppositeEndian)

Creates a CHRconvFromUtf32ToChar object for the supplied source data.

- pData specifies a pointer to the source data
- pLen specifies the length of the source data in charcter units
- pOppositeEndian specifies that the byte-order of the source data should be read in the opposite order to the platform default

CHRconvFromUtf32ToChar::dataPtr()

qchar * CHRconvFromUtf32ToChar::dataPtr()

Returns a pointer to the converted data, the memory for which is managed by the object.

CHRconvFromUtf32ToChar::len()

qlong CHRconvFromUtf32ToChar::len()

Returns the length of the converted data in character units.

Other Functions

The following functions are found in omstring.h and provide additional support for Unicode (UTF-32) character strings.

OMstr... Functions

There are a number of Omnis string functions to mirror the standard C string functions. These operate on null-terminated strings of qchars and are prefixed to distinguish them from their ASCII counterparts. Functions include:

```
qulong OMstrlen(const qchar *pString)
qchar* OMstrcpy(qchar *pDest, const qchar *pSource)
qchar* OMstrncpy(qchar *pDest, const qchar *pSource, qlong pCount)
qchar* OMstrcat(qchar *pDest, const qchar *pSource)
qchar* OMstrncat(qchar *pDest, const qchar *pSource, qlong pCount)
qbool OMstrequal(const qchar *pString1, const qchar *pString2)
qchar* OMstrstr(const qchar *pString, const qchar *pStrCharSet)
qchar* OMstrchr(const qchar *pString, qchar pChar)
qchar* OMstrrchr(const qchar *pString, qchar pChar)
qlong OMstrcspn(const qchar *pString, const qchar *pStrCharSet )
qlong OMstrcmp(const qchar *pString1, const qchar *pString2)
qlong OMstrncmp(const qchar *pString1, const qchar *pString2, qlong pCount)
qlong OMstrspn(const qchar *pString, const qchar *pStrCharSet )
qchar* OMstrpbrk(const qchar *pString, const qchar *pStrCharSet)
qchar* OMstrtok(OMstrtokContext *pContext, CHR *pStrToken, const CHR *pStrDelimit)
```

The OMstrtok() function expects a *context* parameter. This has been added in order to make the function thread safe. Whareas the single-threaded C equivalent of this function is re-entrant, it assumes the same previous/partially tokenised source string when pStrToken is supplied as NULL, OMstrtok() uses pContext to store the partially tokenised source string. When called in a multi-threaded environment, this ensures that OMstrtok() is always using the correct (partially tokenised) source string. Note that the context struct must remain in scope for all calls to OMstrtok(). Example:

```
OMstrtokContext cxt;    //str255 serverInfoStr; e.g. "3.1.4"
qchar *major = OMstrtok(&cxt, &serverInfoStr[1], (qchar*)".");
qchar *minor = OMstrtok(&cxt,(qchar *)NULL, (qchar*)".");
```

There are also functions to convert between character strings and integers:

```
qchar* OMlongToString(qchar *pDest, qlong pLong) qulong OMstrtoul(qchar *pText, qchar **pTextEnd, qlong pBase)
```

QTEXT() Macro

This is useful for creating and supplying literal string values inside components. When _UNICODE is defined, QTEXT() appends the L ## escape sequence onto the supplied text. This instructs the compiler to treat the resulting text as a string of qoschars. QTEXT() can be used anywhere where a qoschar* argument is required, for example:

```
str255 myString( QTEXT("Default Value") ); //call the qoschar* constructor
```

QCHARLEN() and QOSCHARLEN() Macros

These provide a simple conversion from a supplied byte length to the corresponding qchar or qoschar character length respectively. It should be noted that they do not operate on strings or arrays of characters directly. They simply divide the supplied parameter by 4 in the case of QCHARLEN() or 2 (or 1) in the case of QOSCHARLEN().

QBYTELEN() and QOSBYTELEN() Macros

These provide a simple conversion from a supplied *character length* to the corresponding UTF-32 or UTF-16/UTF-8 *byte length* respectively. It should be noted that they do not operate on strings or arrays of characters directly. They simply multiply the supplied parameter by 4 in the case of QCHARLEN() or 2 (or 1) in the case of QOSCHARLEN().

Chapter 5—EXTBMPref & EXTCURref

Introduction

The EXTBMPref class gives your external components access to the OMNISPIC.DF1 and USERPIC.DF1 data files handled by Omnis. These data files normally reside in the ICONS subdirectory of your Omnis installation. All icons in Omnis are referenced by an icon identifier, or $iconid, which can be modified in the Omnis icon editor.

With the exception of custom cursors and full page bitmaps, each icon in Omnis can have three drawing sizes, 16x16, 32x32 and 48x48. Each icon has a pre-set default size that it uses unless another size is specified. This default size can also be modified using the Omnis icon editor. Some icons also have drawing modes. For example, checkbox icons have various modes, normal, checked, highlighted etc.

Enumerations

ePicModes (EXTBMPref only)

An enum defining the drawing modes supported by the icon-drawing function in this class.

- picNormal The icon is drawn in its normal state.
- picChecke The icon is drawn in its checked state.
- picHilited The icon is drawn in its hilited state.
- picCheckedHilited The icon is drawn in its checked and hilited state.

ePicSize (EXTBMPref only)

An enum defining the drawing size supported by the icon drawing function in this class.

- ePicDef The size of the icon depends on the default size set in the Omnis icon editor.
- ePic16 The 16x16 version of the icon is drawn.
- ePic32 The 32x32 version of the icon is drawn.
- ePic48 The 48x48 version of the icon is drawn.

EXTBMPref Class Reference

EXTBMPref::EXTBMPref()

EXTBMPref::EXTBMPref(qlong pIconID, qlong pDefault = 0, qapp pApp = 0)

The constructor for the external bitmap class. After construction, the class can be used to interrogate the icon or draw the icon. When you have finished manipulating the icon, the class should be destructed.

- **pIconID** - the icon associated with this class.
- **pDefault** - the default icon id is used when pIconID is zero.
- **pApp** - this parameter must be specified for web client components. See ECOgetApp.

EXTBMPref::~EXTBMPref()

EXTBMPref::~EXTBMPref()

The destructor for the external bitmap class. The destruction of the class informs Omnis that you have finished with the icon.

EXTBMPref::addBmpSize()

qlong EXTBMPref::addBmpSize(qlong pIconID, ePicSize pSize)

Returns a new icon id with the specified pSize added.

- **pIconId** - The icon id to add a size to.
- **pSize** - The size to be added to the icon id.
- **return** - A new icon id with the icon size pSize embedded.

Note: This is a static member function.

EXTBMPref::copyImage()

```
HBITMAP EXTBMPref::copyImage( ePicSize pSize = ePicDef )HBITMAP
EXTBMPref::copyImage( qcol pFillColor, ePicSize pSize = ePicDef )
```

Returns a bitmap for the icon this class refers.

- **pFillColor** - When calling copyImage specifying a fill color, the transparent pixels of the image are replaced with the given color.
- **pSize** - The icon size to return.
- **return** - Returns a new HBITMAP object if successful.

Note: The returned HBITMAP must be destroyed with GDIdeleteBitmap.

EXTBMPref::copyImage()

```
HBITMAP EXTBMPref::copyImage( qcol pFillColor, ePicSize pSize = ePicDef )
```

Returns a bitmap for the icon this class references. The transparent color of the bitmap is replaced with the given color.

- **pFillColor** - The replacement color for the transparent color of the bitmap.
- **pSize** - The icon size to return.
- **return** - Returns a new HBITMAP object if successful.

Note: The returned HBITMAP must be destroyed with GDIdeleteBitmap.

EXTBMPref::copyMask()

```
HBITMAPMASK EXTBMPref::copyMask( ePicSize pSize = ePicDef )
```

Returns a bitmap mask for the icon this class refers.

- **pSize** - The icon size to return a mask for.
- **return** - Returns a new HBITMAPMASK object if successful.

Note: The returned HBITMAPMASK must be destroyed with GDIdeleteBitmap.

EXTBMPref::draw()

```
void EXTBMPref::draw( HDC pHdc, qrect* pRect, ePicSize pSize = ePicDef, ePicModes pWhich
= picNormal, qbool pDisabled = qfalse, qcol pHilited = colNone, qbool pScale = qfalse, qjst
pJstHoriz = jstLeft, qjst pJstVert = jstLeft )
```

Draws the icon's image into a device context.

- **pHdc** - The drawing device to draw into.
- **pRect** - The destination drawing rectangle.

- **pSize** - The icon size to draw.
- **pWhich** - The icon drawing mode to use.
- **pDisabled** - If qtrue the image is drawn in a disabled state.
- **pHilited** - Controls how the icon is highlighted.
- **pScale** - If qtrue the icon is scaled to the full size of pRect.
- **pJstHoriz** - The horizontal drawing justification. This is ignored if pScale is qtrue.
- **pJstVert** - The horizontal drawing justification. This is ignored if pScale is qtrue.

EXTBMPref::drawMask()

```
void EXTBMPref::drawMask( HDC pHdc, qrect* pRect, ePicSize pSize = ePicDef, ePicModes
pWhich = picNormal, qbool pScale = qfalse, qjst pJstHoriz = jstLeft, qjst pJstVert = jstLeft)
```

Draws the icons mask image into a device context.

- **pHdc** - The drawing device to draw into.
- **pRect** - The destination drawing rectangle.
- **pSize** - The icon size to draw.
- **pWhich** - The icon drawing mode to use.
- **pScale** - If qtrue the icon is scaled to the full size of pRect.
- **pJstHoriz** - The horizontal drawing justification. This is ignored if pScale is qtrue.
- **pJstVert** - The horizontal drawing justification. This is ignored if pScale is qtrue.

EXTBMPref::getBmpSize()

```
ePicSize EXTBMPref::getBmpSize( qlong pIconID )
```

Returns the icon size extracted from the icon id passed.

- **pIconId** - The icon id to extract an icon size from.
- **return** - The icon's size.

EXTBMPref::getIconId()

```
qlong EXTBMPref::getIconId()
```

Returns the icon id that was associated with this class at construction.

- **return** - Returns the icon id.

EXTBMPref::getRect()

void EXTBMPref::getRect(qrect* pRect, ePicSize pSize = ePic16)

Returns a bounding rectangle for the icon. The resulting size depends on the passed size parameter.

- **pRect** - set to the correct bounding rectangle size.
- **pSize** - Controls the returned size. This parameter defaults to the 16x16 size.

EXTBMPref::hasMode()

qbool EXTBMPref::hasMode(ePicModes pMode = picNormal)

Used to determine if the icon this class refers to supports a particular drawing mode.

- **pMode** - The icon drawing mode to test against. This parameter defaults to the normal drawing mode.
- **return** - Returns qtrue if the icon does support **pMode**, and return qfalse if it does not.

EXTBMPref::hasSize()

qbool EXTBMPref::hasSize(ePicSize pSize = ePic16)

Used to determine if the icon this class refers to has a particular icon size.

- **pSize** - The icon size to test against. This parameter defaults to the 16x16 size.
- **return** - Returns qtrue if the icon does support **pSize**, and return qfalse it does not.

Example:

```
// This example gets a bitmap from Omnis using icon reference number 1000.
// The icon reference is asked how big it should draw by default. The draw
// method is called to draw the icon in a rectangle. The icon is centered
// both vertically and horizontally in the rect. NOTE: it is not clipped to
// the rectangle. It is very important to delete the bitmap reference
// when you are finished.
WNDpaintStruct paintStruct;
WNDbeginPaint( mHWnd, &paintStruct );
EXTBMPref bmpRef( 1000 );
ePicSize defaultSize = EXTBMPref::getBmpSize( 1000 );
bmpRef.draw( paintStruct.hdc, &drawRect , defaultSize, picNormal, qfalse, colNone, qfalse, jstCenter, jstCente
WNDendPaint( mHWnd, &paintStruct );
```

EXTBMPref::transparentColor()

qcol EXTBMPref::transparentColor()

Used to get the transparent color of the bitmap image.

EXTCURref Class Reference (v2.2)

EXTCURref::EXTCURref()

```
EXTCURref::EXTCURref( qlong pCursorID, qlong pDefault = 0, qapp pApp = 0 )
```

The constructor for the external cursor class. After construction, the class can be used to create a HCURSOR. When you have finished with the cursor reference, the class should be destructed. Destructing the class will not destroy the HCURSOR which was created from it.

- **pCursorID** - the cursor associated with this class.
- **pDefault** - the default cursor id is used when pCursorID is zero.
- **pApp** - this parameter must be specified for web client components. See ECOgetApp.

EXTCURref::~EXTCURref()

```
EXTBMPref::~EXTBMPref()
```

The destructor for the external cursor class. The destruction of the class informs Omnis that you have finished with the cursor.

EXTCURref::getCursor()

```
HCURSOR EXTCURref::getCursor()
```

The getCursor function creates and returns a HCURSOR. You can effect the screen cursor by calling WNDsetCursor or WNDsetWindowCursor.

EXTCURref::getCursorId()

```
qlong EXTCURref::getCursorId()
```

Returns the cursor ID.

Chapter 6—qkey Reference

Introduction

The QKEY class gives your external component access to keyboard messages and some keyboard checking functions. It refers to two kinds of key, a VCHAR and a PCHAR. A VCHAR is a virtual key code for special keys such as the PageUp key. PCHAR refers to printable characters.

Keyboard messages WM_KEYDOWN and WM_KEYUP pass a pointer to a qkey object.

Enumerations

vChar

An enum defining some virtual keyboard values.

- vcF1 The F1 key on the keyboard
- vcUp The up arrow key on the keyboard
- vcDown The down arrow key on the keyboard
- vcLeft The left arrow key on the keyboard
- vcRight The right arrow key on the keyboard
- vcPup The page up key on the keyboard
- vcPdown The page down key on the keyboard
- vcPleft The page left key on the keyboard
- vcPright The page right key on the keyboard
- vcHome The home key on the keyboard
- vcEnd The end key on the keyboard
- vcTab The tab key on the keyboard
- vcReturn The return key on the keyboard
- vcEnter The enter key on the keyboard
- vcBack The backspace key on the keyboard
- vcClear The clear key on the keyboard
- vcCancel The escape key on the keyboard
- vcDel The forward delete key on the keyboard
- vcIns The insert key on the keyboard

qkey Class Reference

qkey::qkey()

qkey::qkey(LPARAM pKeyValue)

The constructor for the external keyboard class. After construction, the class can be used to interrogate the keyboard message.

- **pKeyValue** - This is the keyboard scan value passed in LPARAM on a WM_KEYDOWN, WM_KEYUP message.

qkey::qkey()

```
qkey::qkey( pchar pPchar, qbool pShift,
qbool pOption, qbool pControl )
```

Creates a qkey object from the printable character and key states passed.

- **pPchar** - The printable character to be added into the new qkey.
- **pShift** - The state of the shift key for the new qkey object.
- **pOption** - The state of the option key for the new qkey object.
- **pControl** - The state of the control key for the new qkey object.
- **return** - Returns a new qkey object.

See also qkey::getPChar()

qkey::qkey()

```
qkey::qkey( vchar pVchar, qbool pShift, qbool
pOption, qbool pControl )
```

Creates a qkey object from the virtual key code and key states passed.

- **pVchar** - The virtual keyboard value to be added into the new qkey.
- **pShift** - The state of the shift key for the new qkey object.
- **pOption** - The state of the option key for the new qkey object.
- **pControl** - The state of the control key for the new qkey object.
- **return** - Returns a new qkey object.

qkey::qkey()

```
qkey::qkey( )
```

Creates a qkey object with only the modifier states (SHIFT, CONTROL and OPTION) set.

- **return** - Returns a new qkey object.

qkey::getPChar()

```
pchar qkey::getPChar()
```

Returns the printable character from the key message.

- **returns** - Returns the character.

qkey::getVChar()

vchar qkey::getVChar()

Returns the virtual key code from the key message.

- **returns** - Returns the key code.

qkey::isAlt()

qbool qkey::isAlt()

Returns the state of the ALT key for this keyboard message.

- **returns** - Returns qtrue if the ALT key is down.

qkey::isControl()

qbool qkey::isControl()

Returns the state of the CONTROL key for this keyboard message.

- **returns** - Returns qtrue if the CONTROL key is down.

qkey::isShift()

qbool qkey::isShift()

Returns the state of the SHIFT key for this keyboard message.

- **returns** - Returns qtrue if the SHIFT key is down.

qkey::operator !()

qbool qkey::operator ! ( )

Tests if the qkey object is invalid.

- **return** - qtrue if the qkey object is invalid and qfalse if the object is valid.

qkey::operator !=()

qbool qkey::operator != (const qkey& pTestKey)

Compares the key message stored in this qkey object with the key message passed in.

- **pTestKey** - The qkey object to compare against.
- **return** - qtrue if the qkey key messages are not the same.

qkey::operator ==()

qbool qkey::operator == (const qkey& pTestKey)

Compares the key message stored in this qkey object with the key message passed in.

- **pTestKey** - The qkey object to compare against.
- **return** - qtrue if the qkey key messages match and qfalse if the objects are different.

qkey::uppc()

void qkey::uppc()

Uppercases the printable character stored in the qkey object.

See also qkey::getPChar()

Other Functions

isShift()

qbool isShift()

Returns the current state of the SHIFT key.

- **returns** - Returns qtrue if the SHIFT key is down and qfalse if up.

isAlt()

qbool isAlt()

Returns the current state of the ALT key.

- **returns** - Returns qtrue if the ALT key is down and qfalse if up.

Example:

```
extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo
{
  ECOsetupCallbacks(hwnd,eci);
  switch (Msg)
  {
    case WM_KEYDOWN:
    case WM_KEYUP:
    {
      qkey* keyMessage = (qkey*)lParam;
      if ( keyMessage->getPChar()=='P' )
      {
        // The 'P' key was pressed.
        return 0L; // tell Omnis we have processed the key
      }
```

```
      else if ( keyMessage->isShift() && keyMessage->getPChar()=='L' )
      {
        // The 'L' key and 'SHIFT' keys were pressed.
        return 0L; /// tell Omnis we have processed the key
      }
      return 1L; // let Omnis process the key
    }
  }
  return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci);
}
```

Chapter 7—EXTfile Reference

Introduction

The FILE API functions give your external components general file handling functionality.

The EXTfile class is a wrapper for the FILExxx functions. It is generally safer to use the class, but sometimes it can be more convenient to call the API functions directly.

API Functions

These functions are defined in EXTFILE.HE

FILEclose()

```
void FILEclose( qfileptr pFileInstance )
```

Closes the file.

- **pFileInstance** – The file instance which contains the file handle to close.

FILEcreate()

```
qret FILEcreate(qfileptr pFileInstance,
strxxx& pName, qbool pExclusive )
```

Creates a new file and then opens it.

- **pFileInstance** – The file instance used to create the file and hold the file handle.
- **pName** – strxxx reference which contains the name of the file to create.
- **pExclusive** – True if the file should be opened in exclusive mode after it is created.
- **returns** – qret error code.

FILEcreateInst()

```
qfileptr FILEcreateInst( )
```

Constructs a new file instance.

- **returns** – A new file instance. This must be deleted by FILEdestroyInst.

See also FILEdestroyInst

FILEcreateTemp()

```
qret FILEcreateTemp(qfileptr pFileInstance )
```

Constructs a new temporary file and then opens it (in exclusive mode).

- **pFileInstance** – The file instance used to create the file and hold the file handle.
- **returns** – qret error code.

FILEdelete()

```
qret FILEdelete(strxxx& pName)
```

Deletes the specified file.

- **pName** – A strxxx reference which contains the name of the file to delete.
- **returns** – qret error code.

FILEdestroyInst()

```
void FILEdestroyInst(qfileptr pFileInstance)
```

Destroys a file instance.

- **pFileInstance** – The file instance. This was previously created by FILEcreateInst.

See also FILEcreateInst

FILEexists()

```
qbool FILEexists(strxxx& pName, qbool pIsFolder = qfalse )
```

Tests whether the specified file (or folder) exists or not.

- **pName** – strxxx reference which contains the name of the file (or folder).
- **pIsFolder** – True if the pName is a folder, false if it is a file name.
- **returns** – True if the file (or folder) exists, false otherwise.

FILEfullName()

```
void FILEfullName(strxxx& pName, filevref pMacVolRef = 0)
```

Obtains the full name of the file.

- **pName** – The filename to obtain the full name for.
- **pMacVolRef** – The Macintosh volume reference. Not required for Windows..

FILEgetLength()

```
qlong FILEgetLength(qfileptr pFileInstance )
```

Obtains the length of the file.

- **pFileInstance** – The file instance which contains the file handle.
- **returns** – The length of the file.

FILEgetName()

```
void FILEgetName(qfileptr pFileInstance,
strxxx& pName, qbool pIncPath = qtrue )
```

Obtains the name of the file.

- **pFileInstance** – The file instance which contains the file handle.
- **pName** – strxxx reference which will contain the name of the file after the function call.
- **pIncPath** – True if the pName should also contain the path of the file.

FILEgetOmnisFolder() (v3.1)

```
void FILEgetOmnisFolder( qfileptr pFilePtr, str255& pFilename )
```

Returns the path to the Omnis folder; the folder which usually contains the main executable and support folders. FILEgetOmnisFolder is passed a pointer to a qfile class object.

- **pFilePtr** – Pointer to a qfile class object.
- **pFilename** – (output) The folder name containing the Omnis support files.

FILEgetOmnisProgramFolder() (v4.3)

```
void doQfile_getOmnisProgramFolder(qfile
*pFile, str255 &pFilename)
```

Returns the path to the Omnis executable; the folder containing the main executable. FILEgetOmnisProgramFolder is passed a pointer to a qfile class object.

- **pFilePtr** – Pointer to a qfile class object.
- **pFilename** – (output) The folder name containing the Omnis executable.

FILEgetPosition()

```
qlong FILEgetPosition( qfileptr pFileInstance )
```

Constructs a new file instance.

- **pFileInstance** – The file instance which contains the file handle.
- **returns** – The current position in the file.

FILEopen()

```
qret FILEopen(qfileptr pFileInstance, strxxx&
pName, qbool pReadOnly, qbool pExclusive)
```

Opens the specified file.

- **pFileInstance** – The file instance which will contain the opened file handle.
- **pName** – strxxx reference which contains the file to open.
- **pReadOnly** – True if the file should be opened in read-only mode.
- **pExclusive** – True if the file should be opened in exclusive mode.
- **returns** – qret error code.

FILEopenResources() (v3.1)

```
qret FILEopenResources(qfileptr
pFileInstance, strxxx& pName, qbool
pReadOnly, qbool pExclusive)
```

Opens the Macintosh resources fork of the specified file as a data file.

- **pFileInstance** – The file instance which will contain the opened file handle.
- **pName** – strxxx reference which contains the file to open.
- **pReadOnly** – True if the file should be opened in read-only mode.
- **pExclusive** – True if the file should be opened in exclusive mode.
- **returns** – qret error code. If 1 is returned, the function is not implemented.

FILEread()

```
qret FILEread(qfileptr pFileInstance, void*
pData, qlong pOffset, qlong pLength)
```

Reads from the file.

- **pFileInstance** – The file instance which contains the file handle.
- **pData** – Pointer to read into.
- **pOffset** – Offset into the file to use.
- **pLength** – Amount of bytes to read.
- **returns** – qret error code.

FILEread()

```
qret FILEread(qfileptr pFileInstance, void*
pData, qlong pOffset, qlong pMaxLength,
qlong& pActLength )
```

Reads from the file.

- **pFileInstance** – The file instance which contains the file handle.
- **pData** – Pointer to read into.
- **pOffset** – Offset into the file to use.
- **pMaxLength** – Number of bytes to read.
- **pActLength** – Actual number of bytes read.
- **returns** – qret error code.

FILEsetEmpty()

```
qret FILEsetEmpty(qfileptr pFileInstance )
```

FILEsetEmpty sets the length of the file to zero bytes.

- **pFileInstance** – The file instance which contains the file handle.
- **returns** – qret error code.

FILEsetLength()

```
qret FILEsetLength(qfileptr pFileInstance, qlong pLength )
```

FILEsetLength sets the length of the file to the specified length.

- **pFileInstance** – The file instance which contains the file handle.
- **pLength** – The new length of the file.
- **returns** – qret error code.

FILEsetMacTypeCreator()

```
void FILEsetMacTypeCreator(qfileptr
pFileInstance, qint4 pMacType, qint4
pMacCreator)
```

Sets the Macintosh file-systems' creator information.

- **pFileInstance** – The file instance which contains the file handle.
- **pMacType** – The new Mac type info.
- **pMacCreator** – The new Mac creator info.

FILEsetMacTypeCreator()

```
void FILEsetMacTypeCreator(strxxx&
pName,qint4 pMacType,qint4 pMacCreator)
```

Sets the Macintosh file-systems' creator information.

- **pName** – The file name.
- **pMacType** – The new Mac type info.
- **pMacCreator** – The new Mac creator info.

FILEsetPosition()

```
qret FILEsetPosition(qfileptr pFileInstance, qlong pPosition)
```

Sets the current position of the file.

- **pFileInstance** - The file instance which contains the file handle.
- **pPosition** – The new position.
- **returns** – A qret error code.

FILEwrite()

```
qret FILEwrite( qfileptr pFileInstance, void*
pData, qlong pOffset, qlong pLength)
```

Writes data to the file.

- **pFileInstance** - The file instance which contains the file handle.
- **pData** – The address of the data to write.
- **pOffset** – The offset into the file of where to write from.
- **pLength** – The number of bytes to write.
- **returns** – A qret error code.

EXTfile Class Reference

EXTfile::EXTfile()

```
EXTfile::EXTfile( )
```

The constructor for a file object.

EXTfile::~EXTfile()

EXTfile::~EXTfile()

The destructor for an EXTfile object.

EXTfile::close()

void EXTfile::close()

Closes the file.

EXTfile::create()

qret EXTfile::create(strxxx& pName, qbool pExclusive)

Creates and then opens the specified file.

- **pName –** The file to create.
- **pExclusive –** True if the file should be opened in exclusive mode.
- **Returns** – qret error code.

EXTfile::createTemp()

qret EXTfile::createTemp()

Creates and then opens, in exclusive mode, a temporary file.

- **returns –** qret error code.

EXTfile::deleet()

static qret EXTfile::deleet(strxxx& pName, qbool deleteContents = qfalse, qbool recursive = qfalse)

Deletes the specified file, and/or folder(s) depending on combination of deleteContents and recursive parameters.

- **pName -** The file to delete
- **deleteContents** – True to delete contents of a folder, default is false
- **recursive** - default is false
- **returns –** qret error code.

EXTfile file;

file.deleet(path) - same as file.deleet(path, qfalse, qfalse), deleteContents and recursive default to kFalse.

EXTfile file;

file.deleet(path, qtrue, qfalse) - if path is a folder, all files inside the folder will be deleted if deleteContents is true. Note that any subfolders will not be deleted. Furthermore, if the folder is empty, the folder will not be deleted in this case as we are telling it to delete only the contents of the type file.

EXTfile file;

file.deleet(path, qtrue, qtrue) - if path is a folder, it will be wiped off the disk with its contents, if any are present.

EXTfile file;

file.deleet(path, qfalse, qfalse) - if path is a folder, the folder will be deleted only if empty i.e. no files or directories are present in the folder at the time of the command being executed. On macOS, this will check if the only file is .DS_Store and will delete the folder in that case. It's important to note that if there are hidden files in the folder, the folder and the hidden files will not be deleted.

EXTfile file;

file.deleet(path, qfalse, qtrue) - if path is a folder, it will be deleted only if empty. If there are any contents, those will not be deleted as deleteContents is kFalse, even if recursive is kTrue.

EXTfile file;

file.deleet(file) - if file is a file, the file will be removed regardless of the deleteContents or recursive values - the new parameters do not impact the behavior of deleting files, only the behavior of deleting folders.

EXTfile::exists()

static qbool EXTfile::exists(strxxx& pName, qbool pIsFolder)

Tests whether the specified file (or folder) exists or not.

- **pName** – strxxx reference which contains the name of the file (or folder).
- **pIsFolder** – True if the pName is a folder, false if it is a file name.
- **returns** – True if the file (or folder) exists, false otherwise.

EXTfile::fullName()

static EXTfile::fullName(strxxx& pName, filevref pMacVolRef=0)

Obtains the full name of the file.

- **pName** – The filename to obtain the full name for.
- **pMacVolRef** – The Macintosh volume reference. Not required for Windows.

EXTfile::getLength()

qlong EXTfile::getLength()

Obtains the length (in bytes) of the file.

- **returns –** The length, in bytes, of the file.

EXTfile::getName()

```
void EXTfile::getName(strxxx& pFilename, qbool pInclPath = qtrue)
```

Obtains the name of the file.

- **pName** – strxxx reference which will contain the name of the file after the function call.
- **pInclPath** – True if the pName should also contain the path of the file.

EXTfile::getOmnisFolder()

```
void EXTfile::getOmnisFolder( str255& pFilename )
```

Returns the path to the Omnis folder; the folder which usually contains the main executable and support folders.

- **pFilename** – (output) The folder name containing the Omnis support files.

EXTfile::getOmnisProgramFolder() (v4.3)

```
void EXTfile::getOmnisProgramFolder( str255& pFilename )
```

Returns the path to the Omnis executable; the folder containing the main executable.

- **pFilename** – (output) The folder name containing the Omnis executable.

EXTfile::getPosition()

```
qlong EXTfile::getPosition()
```

Obtains the current position in the file.

- **returns –** Returns the current position in the file.

EXTfile::open()

```
qret EXTfile::open(strxxx& pName, qbool
pReadOnly, qbool pExclusive)
```

Opens the specified file.

- **pName –** The file to open.
- **pReadOnly –** True if the file should be opened in read-only mode.
- **pExclusive** – True if the file should be opened in exclusive mode.
- **returns** – qret error code.

EXTfile::openResources() (v3.1)

```
qret EXTfile::openResources(strxxx& pName,
qbool pReadOnly, qbool pExclusive)
```

Opens the Macintosh resources fork of the specified file as a data file.

- **pName –** The file to open.
- **pReadOnly –** True if the file should be opened in read-only mode.
- **pExclusive** – True if the file should be opened in exclusive mode.
- **returns** – qret error code.

EXTfile::read()

```
qret EXTfile::read(void* pData, qlong pOffset, qlong pLength)
```

Reads from the file.

- **pData** – Pointer to read into.
- **pOffset** – Offset into the file to use.
- **pLength** – Amount of bytes to read.

EXTfile::read()

```
qret EXTfile::read(void* pData, qlong pOffset,
qlong pMaxLength, qlong& pActLength )
```

Reads from the file.

- **pData** – Pointer to read into.
- **pOffset** – Offset into the file to use.
- **pMaxLength** – Number of bytes to read.
- **pActLength** – Actual number of bytes read.

EXTfile::readCharacterData() (v4.0)

```
qret EXTfile::readCharacterData(qHandle
&pHan, FILEconversionType pConvType)
```

Reads file containing character data into a handle which becomes an array of qchars.

- **pHan** - Handle to read file into.
- **pConvType** - An EXTfile constant specifying the conversion required.

```
EXTfile file; qret e = file.open( document, qtrue, qfalse );
if ( e == e_ok )
{
  mDocHan = 0;
  e = file.readCharacterData(mDocHan, EXTfile::eFILEconvertFromLatin1Api);
  if ( e == e_ok )
  {
    mDocPtr = qHandleTextPtr( mDocHan, 0 );
    parse(pSrchWords);
  }
  else mDocPtr.setNull();
  file.close();
}
```

EXTfile::readIntoHandle() (v4.0)

qret EXTfile::readIntoHandle(qHandle &pHan)

Reads the raw contents of a file into a handle.

- **pHan** – Handle to read file into.

EXTfile::setEmpty()

qret EXTfile::setEmpty()

setEmpty sets the length of the file to zero bytes.

- **returns** – qret error code.

EXTfile::setLength()

qret EXTfile::setLength(qlong pLength)

setLength sets the length of the file to the specified length.

- **pLength** – The new length of the file.
- **returns** – qret error code.

EXTfile::setMacTypeCreator()

void EXTfile::setMacTypeCreator(qint4 pMacType,qint4 pMacCreator)

Sets the Macintosh file-systems' creator information.

- **pMacType** – The new Mac type info.
- **pMacCreator** – The new Mac creator info.

EXTfile::setMacTypeCreator()

```
static void
EXTfile::setMacTypeCreator(strxxx&
pName,qint4 MacType,qint4 pMacCreator)
```

Sets the Macintosh file-systems' creator information.

- **pName** – The file name.
- **pMacType** – The new Mac type info.
- **pMacCreator** – The new Mac creator info.

EXTfile::setPosition()

```
qret EXTfile::setPosition(qlong pPosition)
```

Sets the current position of the file.

- **pPosition** – The new position.
- **returns** – A qret error code.

EXTfile::write()

```
qret EXTfile::write(void* pData, qlong pOffset, qlong pLength)
```

Writes data to the file.

- **pData** – The address of the data to write.
- **pOffset** – The offset into the file of where to write from.
- **pLength** – The number of bytes to write.
- **returns** – A qret error code.

Chapter 8—CRB Reference

Introduction

The CRB API functions are a set of functions which allow you to create and manage Omnis data collections. An Omnis data collection is a block of data with a variable number of data items. A CRB can store number data, list data, text data, etc, in any order and combination. A CRB is self extending. In other words you can simply set the data for a given index, and the CRB is extended to store the given data at the specified index position. The collection of data in a CRB can be converted to and from disk-based format for storing on and retrieving from disk. You can also assign CRB data to and retrieve from an EXTfldval, see EXTfldval::getCrbRef and EXTfldval::setCrbRef. This is useful if you want to exchange CRB data with the Omnis 4GL.

The EXTcrb class is a wrapper for the CRBxxx functions. It is generally safer to use the class, but sometimes it can be more convenient to call the API functions directly.

API Functions

These functions are defined in EXTCRB.HE

CRBcreate()

qcrb CRBcreate()

Creates a new empty CRB instance. When you have finished with the instance you must destroy it, unless you have transferred ownership when calling EXTfldval::setCrbRef.

- **returns** - The pointer to the CRB instance.

See also CRBdestroy

CRBdestroy()

void CRBdestroy(qcrb pCrb)

Destroys the given CRB instance. The instance must have been created with CRBcreate.

- **pCrb** - pointer to the CRB instance to be destroyed.

See also CRBcreate

CRBduplicate()

qcrb CRBduplicate(qcrb pCrb)

Makes a copy of the given CRB instance. When you have finished with the copy, you must destroy it, unless you have transferred ownership when calling EXTfldval::setCrbRef.

- **pCrb** - pointer to the CRB instance to be duplicated.
- **returns** - new pointer to a CRB instance.

See also CRBcreate, CRBdestroy

CRBflatten()

qlong CRBflatten(qcrb pCrb, qchar* pBuffer, qlong pBufferLen)

Converts the data in a CRB instance into a cross-platform flat format which is suitable for storing on disk. You must allocate a sufficiently large buffer to receive the data. You can call CRBgetFlatSize prior to calling CRBflatten, to tell you the size of the required buffer.

- **pCrb** - pointer to the CRB instance to be flattened.
- **pBuffer** - pointer to the buffer which is to receive the flattened data.
- **pBufferLen** - buffer size in bytes.
- **returns** - length of the flattened data.

Example:

```
// *** store some cross platform data on disk ***
// create the crb instance
qcrb crb = CRBcreate();
// store some text at index 1
EXTfldval fvalp( CRBgetDataRef( crb, 1, qtrue ) );
fvalp.setChar("Some text to be stored on disk")
// store some numbers at the next 3 index positions
CRBsetReal( crb, 2, 4.999 );
CRBsetLong( crb, 3, 255 );
CRBsetLong( crb, 4, 1000 );
// allocate the buffer which will receive the flattened data
qlong bufferLen = CRBgetFlatSize( crb );
qchar* buffer = new qchar[ bufferLen ];
// flatten the data.
// Note: in our case dataLen should be identical to bufferLen
qlong dataLen = CRBflatten( crb, buffer, bufferLen );
// now we can write the data to disk
EXTfile file; file.create( str255("FileName"), qtrue );
file.write( buffer, 0, dataLen);
file.close();
// destroy the buffer and the crb
delete [] buffer;
CRBdestroy( crb );
// *** end ***
```

See also CRBgetFlatSize, CRBunflatten

CRBgetCrbRef()

qcrb CRBgetCrbRef(qcrb pCrb, qcrb pTmpCrb, qcrbindex pIndex, qbool pWillAlter)

It is possible to store data collections within data collections. You can do this by calling this function. If required, when calling this function, the data at the given index is converted to an Omnis data collection. If you have several data collections stored in a CRB, you can optimize performance by creating your own temp CRB for manipulating the nested data collections, which you can specify for the pTmpCrb parameter. If you do not specify your own temp CRB, Omnis will create a CRB instance for each data collection stored in the parent CRB. Specifying your own temp CRB works, because Omnis only stores the data collection as a handle inside another CRB, and not the CRB instance itself, which is only used for manipulating the data. If you want to change the contents of the data collection, specify qtrue for pWillAlter.

- **pCrb** - pointer to the CRB instance.
- **pTmpCrb** - temp CRB instance to be used for managing the data collection.
- **pIndex** - index into CRB starting from 1.
- **pWillAlter** - if qtrue, you can change the data collection at the index by assigning new data to index positions of the returned CRB.
- **returns** - pointer to a CRB instance. The CRB instance belongs to the parent CRB and there is no need to destroy it. If you have passed a temp crb in the pTmpCrb parameter, your temp CRB instance is returned instead.

Example:

```
// ** store two data collections in our CRB **
// create our parent CRB
qcrb crb = CRBcreate();
```

```
// create out temp crb for manipulating our child data collections
qcrb tempCrb = CRBcreate();
// fetch our first data collection and set some data in it
// Note: in our case childCrb will be identical to tempCrb
qcrb childCrb = CRBgetCrbRef( crb, tempCrb, 1, qtrue );
CRBsetLong( childCrb, 1, 15 );
CRBsetLong( childCrb, 2, 120 );
CRBsetReal( childCrb, 3, 1.5234 );
// fetch our second data collection and set some data in it
// in our first column we will store some text
childCrb = CRBgetCrbRef( crb, tempCrb, 2, qtrue );
EXTfldval fvalp( CRBgetDataRef( childCrb, 1, qtrue ) );
fvalp.setChar("Hello World");
CRBsetLong( childCrb, 2, 1024 );
// We must remember to destroy our tempCrb
CRBdestroy( tempCrb );
```

Note: You can nest data collections many levels deep.

See also CRBgetDataRef

CRBgetData()

void CRBgetData(qcrb pCrb, qcrbindex pIndex, qshort pFft, qshort pFdp, qfldval pCrbVal)

Retrieves a copy of the data stored at the specified index position in the CRB. You must specify the data type and sub type of the data to be returned as. If the data in the CRB is of a different type, Omnis will convert the data to the specified type.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **pFft** - the data type to return the data as.
- **pFdp** - the sub data type to return the data as.
- **pCrbVal** - pointer to an Omnis data item. You can retrieve this pointer from an EXTfldval by calling EXTfldval::getFldVal.

Example:

```
EXTfldval fval;
CRBgetData( crb, 2, fftCharacter, dpDefault, fval.getFldVal() );
```

See also EXTfldval::getFldVal, CRBsetData, CRBgetDataRef

CRBgetDataRef()

qfldval CRBgetDataRef(qcrb pCrb, qcrbindex pIndex, qbool pWillAlter)

Returns a reference to the index in the CRB. This is more efficient than calling CRBgetData, since the data is not copied. You can construct a EXTfldval from the returned Omnis data pointer. You can use CRBgetDataRef to change the data at the given index, if you specify qtrue for pWillAlter.

- **pCrb** - pointer to the CRB instance.

- **pIndex** - index into CRB starting from 1.
- **pWillAlter** - if qtrue, you can change the data at the index by assigning new data to the EXTfldval.
- **returns** - pointer to an Omnis data item.

Example:

```
// change the data at index 2 using CRBgetDataRef
qcrb crb = CRBcreate();
EXTfldval fvalp( CRBgetDataRef( crb, 2, qtrue ) );
fvalp.setChar("Hello World");
```

See also CRBgetData, CRBsetData, CRBgetCrbRef

CRBgetFlatSize()

qlong CRBgetFlatSize(qcrb pCrb)

Calculates the flattened size of the data in the given CRB instance. You will need to allocate a buffer of the returned size before you can flatten the data.

- **pCrb** - pointer to the CRB instance.
- **returns** - required size of the buffer for flattening the CRB data.

See also CRBflatten, CRBunflatten

CRBgetIndexCount()

qshort CRBgetIndexCount(qcrb pCrb)

Returns the number of data items in the CRB. The index count will usually be in multiples of 10. CRBgetIndexCount does not return a count of the entries which have been used, it returns the count of allocated indexes.

Note: Indexing starts from 1.

- **pCrb** - pointer to the CRB instance.
- **returns** - the index count.

CRBgetLong()

qlong CRBgetLong(qcrb pCrb, qcrbindex pIndex)

Returns the data stored at specified index as a long integer value. If the data stored at the index is of a different type, the data is converted to a long integer.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **returns** - the data as a long value.

See also CRBsetLong

CRBgetLong64()

Same as CRBgetLong() except that it uses a qlong64 rather than qlong for getting the value.

See also CRBsetLong64

CRBgetReal()

qreal CRBgetReal(qcrb pCrb, qcrbindex pIndex)

Returns the data stored at specified index as a floating point number. If the data stored at the index is of a different type, the data is converted to a floating point number.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **returns** - the data as a floating point number.

See also CRBsetReal

CRBsetData()

void CRBsetData(qcrb pCrb, qcrbindex pIndex, qfldval pCrbVal)

Sets the data in the CRB at the specified index position.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **pCrbVal** - pointer to an Omnis data item. You can retrieve this pointer from an EXTfldval by calling EXTfldval::getFldVal.

Example:

```
// the following example stores text at index 1
// and a list with two rows at index 2
EXTfldval fval;
qcrb crb = CRBcreate();
fval.setChar("Hello World");
CRBsetData( crb, 1, fval.getFldVal() );
EXTqlist lst(listScol);
lst.insertRow( 0, str255("Row one"), 1 );
lst.insertRow( 0, str255("Row two"), 2 );
fval.setList( &lst, qtrue );
CRBsetData( crb, 2, fval.getFldVal() );
```

See also CRBgetData, CRBgetDataRef

CRBsetLong()

```
void CRBsetLong( qcrb pCrb, qcrbindex pIndex, qlong pLongValue )
```

Sets the data at the specified index to the given long integer value.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **pLongValue** - the long integer value to store.

See also CRBgetLong

CRBsetLong64()

Same as CRBsetLong() except that it uses a qlong64 rather than qlong for setting the value.

See also CRBgetLong64

CRBsetReal()

```
void CRBsetReal( qcrb pCrb, qcrbindex pIndex, qreal pRealValue )
```

Sets the data at the specified index to the given floating point number.

- **pCrb** - pointer to the CRB instance.
- **pIndex** - index into CRB starting from 1.
- **pRealValue** - the floating point value to store.

See also CRBgetReal

CRBunflatten()

```
qbool CRBunflatten( qcrb pCrb, qchar* pBuffer, qlong pBufferLen )
```

Converts flattened block of data back into a form suitable for a CRB instance to manage. The block of data must have been flattened previously by a call to CRBflatten.

- **pCrb** - pointer to the CRB instance which is to receive the new data.
- **pBuffer** - pointer to the flattened data.
- **pBufferLen** - length of the flattened data.
- **returns** - if the data was successfully converted, qtrue is returned.

Example:

```
// *** load some data we previously written to disk. ***
// *** See CRBflatten example ***
// read the data from disk into our buffer
EXTfile file(); file.open( str255("fileName"), qtrue, qtrue );
qlong dataLen = file.getLength();
qchar* buffer = new qchar[ dataLen ];
file.read( buffer, 0, dataLen );
file.close();
// create our CRB instance and unflatten the data
qcrb crb = CRBcreate();
if ( CRBunflatten( crb, buffer, dataLen ) )
  // success
else
  // failure
// when we have finished with the crb and buffer, destroy them
delete [] buffer;
CRBdestroy( crb );
// *** end ***
```

See also CRBflatten

EXTcrb Class Reference

EXTcrb::EXTcrb()

EXTcrb::EXTcrb()

The constructor for an EXTcrb object. It constructs an empty Omnis CRB and data collection.

EXTcrb::EXTcrb()

EXTcrb::EXTcrb(qcrb pCrb)

Constructs a EXTcrb object, from an existing CRB instance. You may already have a data collection in a EXTfldval for example. You can use EXTfldval::getCrbRef to retrieve the CRB instance from the EXTfldval and construct a EXTcrb object from it. The EXTcrb object makes the assumption that it does not own the CRB instance, and will not destroy it when the EXTcrb object is destructed. You can always call EXTcrb::makeMine later, if you wish to work with a copy of the CRB instance.

1. **pCrb** - pointer to a CRB instance.

Example:

```
// get existing CRB instance from EXTfldval. Do not make a copy
EXTcrb crb( fval.getCrbRef( qfalse) );
// if we want a copy call makeMine
crb.makeMine();
```

See also EXTfldval::getCrbRef, EXTcrb::makeMine

EXTcrb()::~EXTcrb()

EXTcrb()::~EXTcrb()

The destructor for an EXTcrb object.

EXTcrb::copy()

void EXTcrb::copy(EXTcrb& pCrb)

Copies the CRB instance and data from the given EXTcrb object to this EXTcrb object.

- **pCrb** - the EXTcrb object from which to copy the CRB instance and data.

EXTcrb::crb()

qcrb EXTcrb::crb()

Returns the pointer to the CRB instance. You will need this function when you want to store a data collection in an EXTfldval.

1. **returns** - pointer to the CRB instance.

Example:

```
EXTcrb crb;
EXTfldval fval;
fval.setCrbRef( crb.crb(), qfalse );
```

EXTcrb::flatten()

qlong EXTcrb::flatten(qchar* pBuffer, qlong pBufferLen)

Converts the data in a CRB instance into a cross-platform flat format which is suitable for storing on disk. You must allocate a sufficiently large buffer to receive the data. You can call EXTcrb::getFlatSize prior to calling EXTcrb::flatten, to tell you the size of the required buffer.

- **pBuffer** - pointer to the buffer which is to receive the flattened data.
- **pBufferLen** - buffer size in bytes.
- **returns** - length of the flattened data.

Example:

```
// *** store some cross platform data on disk ***
// create the crb instance
EXTcrb crb;
// store some text at index 1
EXTfldval fvalp( crb.getDataRef( 1, qtrue ) );
fvalp.setChar(str255("Some text to be stored on disk"))
// store some numbers at the next 3 index positions
crb.setReal( 2, 4.999 );
crb.setLong( 3, 255 );
crb.setLong( 4, 1000 );
// allocate the buffer which will receive the flattened data
qlong bufferLen = crb.getFlatSize();
qchar* buffer = new qchar[ bufferLen ];
// flatten the data.
// Note: in our case dataLen should be identical to bufferLen
```

```
qlong dataLen = crb.flatten( buffer, bufferLen );
// now we can write the data to disk
EXTfile file; file.create( str255("FileName"), qtrue );
file.write( buffer, 0, dataLen);
file.close();
// delete the buffer
delete [] buffer;
// *** end ***
```

See also EXTcrb::unflatten, EXTcrb::getFlatSize

EXTcrb::getCrbRef()

---

qcrb EXTcrb::getCrbRef(EXTcrb& pTmpCrb, qcrbindex pIndex, qbool pWillAlter)

It is possible to store data collections within data collections. You can do this by calling this function. If required, when calling this function, the data at the given index is converted to an Omnis data collection. If you have several data collections stored in a CRB, you can optimize performance by creating your own temp EXTcrb object for manipulating the nested data collections, which you can specify for the pTmpCrb parameter. If you do not specify your own temp CRB, Omnis will create a CRB instance for each data collection stored in the parent CRB. Specifying your own temp CRB works, because Omnis only stores the data collection as a handle inside another CRB, and not the CRB instance itself which is only used for manipulating the data. If you want to change the contents of the data collection, specify qtrue for pWillAlter.

- **pTmpCrb** - temp EXTcrb object to be used for managing the data collection.
- **pIndex** - index into CRB starting from 1.
- **pWillAlter** - if qtrue, you can change the data collection at the index by assigning new data to index positions of the returned CRB.
- **returns** - pointer to a CRB instance. The CRB instance belongs to the parent CRB and there is no need to destroy it. If you have passed a temp crb in the pTmpCrb parameter, your temp CRB instance is returned instead.

Example:

```
// ** store two data collections in our CRB **
// create our parent CRB and
// temp CRB for manipulating our child data collections
EXTcrb crb; EXTcrb tempCrb;
// fetch our first data collection and set some data in it
// Note: we ignore the return value since it will point to the
// CRB of our temp CRB object
crb.getCrbRef( tempCrb, 1, qtrue );
tempCrb.setLong( childCrb, 1, 15 );
tempCrb.setLong( childCrb, 2, 120 );
tempCrb.setReal( childCrb, 3, 1.5234 );
// fetch our second data collection and set some data in it
// in our first column we will store some text
CRBgetCrbRef( tempCrb, 2, qtrue );
EXTfldval fvalp( tempCrb.getDataRef( 1, qtrue ) );
fvalp.setChar(str15("Hello World"));
tempCrb.setLong( 2, 1024 );
```

Note: You can nest data collections many levels deep.

See also EXTcrb::getDataRef

EXTcrb::getData()

```
void EXTcrb::getData( qcrbindex pIndex,
qshort pFft, qshort pFdp, qfldval pCrbVal )
```

Retrieves a copy of the data stored at the specified index position in the EXTcrb object. You must specify the data type and sub type of the data to be returned as. If the data in the CRB is of a different type, Omnis will convert the data to the specified type.

- **pIndex** - index into CRB starting from 1.
- **pFft** - the data type to return the data as.
- **pFdp** - the sub data type to return the data as.
- **pCrbVal** - pointer to an Omnis data item. You can retrieve this pointer from an EXTfldval by calling EXTfldval::getFldVal.

Example:

```
EXTfldval fval;
crb.getData( 2, fftCharacter, dpDefault, fval.getFldVal() );
```

See also EXTfldval::getFldVal, EXTcrb::setData, EXTcrb::getDataRef

EXTcrb::getDataRef()

```
qfldval EXTcrb::getDataRef( qcrb pCrb,
qcrbindex pIndex, qbool pWillAlter )
```

Returns a reference to the index in the EXTcrb. This is more efficient than calling EXTcrb::getData, since the data is not copied. You can construct a EXTfldval from the returned Omnis data pointer. You can use EXTcrb::getDataRef to change the data at the given index, if you specify qtrue for pWillAlter.

- **pIndex** - index into CRB starting from 1.
- **pWillAlter** - if qtrue, you can change the data at the index by assigning new data to the EXTfldval.
- **returns** - pointer to an Omnis data item.

Example:

```
// change the data at index 2 using EXTcrb::getDataRef
EXTcrb crb;
EXTfldval fvalp( crb.getDataRef( 2, qtrue ) );
fvalp.setChar(str15("Hello World"));
```

See also EXTcrb::getData, EXTcrb::setData, EXTcrb::getCrbRef

EXTcrb::getFlatSize()

```
qlong EXTcrb::getFlatSize()
```

Calculates the flattened size of the data in the EXTcrb object. You will need to allocate a buffer of the returned size before you can flatten the data.

- **returns** - required size of the buffer for flattening the CRB data.

See also EXTcrb::flatten, EXTcrb::unflatten

EXTcrb::getIndexCount()

qshort EXTcrb::getIndexCount()

Returns the number of data items in the EXTcrb object. The index count will usually be in multiples of 10. EXTcrb::getIndexCount does not return a count of the entries which have been used, it returns the count of allocated indexes.

Note: Indexing starts from 1.

- **returns** - the index count.

EXTcrb::getLong()

qlong EXTcrb::getLong(qcrbindex pIndex)

Returns the data stored at specified index as a long integer value. If the data stored at the index is of a different type, the data is converted to a long integer.

- **pIndex** - index into CRB starting from 1.
- **returns** - the data as a long value.

See also EXTcrb::setLong

EXTcrb::getReal()

qreal EXTcrb::getReal(qcrbindex pIndex)

Returns the data stored at specified index as a floating point number. If the data stored at the index is of a different type, the data is converted to a floating point number.

- **pIndex** - index into CRB starting from 1.
- **returns** - the data as a floating point number.

See also EXTcrb::setReal

EXTcrb::makeMine()

void EXTcrb::makeMine()

If the EXTcrb object does not own the CRB instance, calling this function will make a copy of the CRB instance and set the ownership flag to true. If the EXTcrb object already owns the CRB instance, this function does nothing.

EXTcrb::setData()

void EXTcrb::setData(qcrbindex pIndex, qfldval pCrbVal)

Sets the data in the EXTcrb object at the specified index position.

- **pIndex** - index into CRB starting from 1.
- **pCrbVal** - pointer to an Omnis data item. You can retrieve this pointer from an EXTfldval by calling EXTfldval::getFldVal.

Example:

```
// the following example stores text at index 1
// and a list with two rows at index 2
EXTfldval fval;
EXTcrb crb;
fval.setChar(str255("Hello World"));
crb.setData( 1, fval.getFldVal() );
EXTqlist lst(listScol);
lst.insertRow( 0, str255("Row one"), 1 );
lst.insertRow( 0, str255("Row two"), 2 );
fval.setList( &lst, qtrue );
crb.setData( 2, fval.getFldVal() );
```

See also EXTcrb::getData, EXTcrb::getDataRef

EXTcrb::setLong()

void EXTcrb::setLong(qcrbindex pIndex, qlong pLongValue)

Sets the data at the specified index to the given long integer value.

- **pIndex** - index into CRB starting from 1.
- **pLongValue** - the long integer value to store.

See also EXTcrb::getLong

EXTcrb::setReal()

void EXTcrb::setReal(qcrbindex pIndex, qreal pRealValue)

Sets the data at the specified index to the given floating point number.

- **pIndex** - index into CRB starting from 1.
- **pRealValue** - the floating point value to store.

See also EXTcrb::getReal

EXTcrb::unflatten()

qbool EXTcrb::unflatten(qchar* pBuffer, qlong pBufferLen)

Converts flattened block of data back into a form suitable for a CRB instance to manage. The block of data must have been flattened previously by a call to EXTcrb::flatten.

- **pBuffer** - pointer to the flattened data.
- **pBufferLen** - length of the flattened data.
- **returns** - if the data was successfully converted, qtrue is returned.

Example:

```
// *** load some data we previously written to disk. ***
// *** See EXTcrb::flatten example ***
// read the data from disk into our buffer
EXTfile file(); file.open( str255("fileName"), qtrue, qtrue );
qlong dataLen = file.getLength();
qchar* buffer = new qchar[ dataLen ];
file.read( buffer, 0, dataLen );
file.close();
// create our CRB instance and unflatten the data
EXTcrb crb;
if ( crb.unflatten( crb, buffer, dataLen ) )
  // success
else
  // failure
// delete the buffer
delete [] buffer;
// *** end ***
```

See also EXTcrb::flatten

Chapter 9—EXTqlist Reference

Introduction

The EXTqlist class gives your external components access to the Omnis list data object. The list object handles all of the memory management for columns and rows of data. You can create lists in your components, or you can receive/send the list to/from Omnis. The list object is a very powerful object in Omnis, but gives you even more power when included in external components. General purpose lists do not need to be defined when writing a component as they do within Omnis using the *Define list* command. The list object adjusts column information as you add it. Normal list objects have to store the same type of data in each column. Omnis lists support this, but can also support different data types in the same column in every row added.

Each Omnis list has its own current row, maximum number of rows and a set of selected rows, that can all be inspected or altered using various member functions.

EXTqlist Memory Issues

The EXTqlist class can be constructed in two ways, one as a reference to another EXTqlist, the other as an individual list. You should think of the class as a container, which has some data representing list rows and columns, or as a pointer to another EXTqlist object. Depending on the type of list you create and what you do with it during its life, what you do during destruction of the EXTlist object is **very** important.

Creating a standalone list

If you want your component to store some private items in a list (such as the calendar example), the component first needs to declare an EXTqlist* member. At some point in your component, you need to create a new instance of a EXTqlist object. Use the 'new' operator and specify the lists data type during construction. For example:

```
// Example using a EXTqlist* as a class member
class sampleClass
{
  private:
    EXTqlist* mMyList;
  public:
    sampleClass()
    ~ sampleClass();
    void doSomething();
};
sampleClass::sampleClass()
{
  mMyList = new EXTqlist( listScol );
}
```

Now you have an EXTqlist object, you can use the various member function to add rows, delete rows or manipulate column data.

```
void sampleClass::doSomething()
{
  for ( qlong i =1; i<=10; i++ )
  {
    str255 textForRow;
    qlongToString( i, textForRow );
    textForRow.insert( str80("This is row "), i );
    mMyList->insertRow(0, &textForRow, i );
  }
}
```

When you have finished with the list, you must delete the contents of the list, then the list container. To delete the contents of the EXTqlist, you can assign the list object **qnil**. When the contents of the list have been emptied, you can delete the EXTqlist object.

```
void sampleClass::~sampleClass()
{
  // first clear the contents of the list object
  *mMyList = qnil;
  delete mMyList;
}
```

REMEMBER: When the EXTqlist object is destructed, the contents are not automatically deleted. You must at some point clear the contents by assigning the object **qnil**.

Creating a reference to another list

Sometimes you will need or be given a reference to another EXTqlist. Maybe Omnis is calling you to paint a line in a derived list control, or a parameter is being passed to you which is a list variable. In both cases, the EXTqlist object you have will be a reference to another list object. This is very important, especially during destruction of the EXTlist object.

Above you created a standalone list. Here you create a reference to a list object and add a new row.

```
// Example if you used 'new' during construction
void sampleClass::makeAReference()
{
  EXTqlist* myRef = 0;
```

```
  myRef = new EXTqlist( mMyList );
  str255 textForRow( "Added by the reference" );
  myRef->insertRow(0, &textForRow, 999 );
  delete myRef;
}
```

or

```
void sampleClass::makeAReference()
{
  EXTqlist* myRef( mMyList );
  str255 textForRow( "Added by the reference" );
  myRef->insertRow(0, &textForRow, 999 );
}
```

In the above examples, the EXTqlist reference was deleted by using 'delete' operator or the scope ending. As the EXTqlist was only used as a reference, that is all you need to do.

If you had done:

```
void sampleClass::makeAReference()
{
  EXTqlist* myRef( mMyList );
  str255 textForRow( "Added by the reference" );
  myRef->insertRow(0, &textForRow, 999 );
  *myRef = qnil;
}
```

the original list 'mMyList' would no longer have any contents as you have deleted it.

Note: If you intend to use the EXTfldval class with your EXTqlist objects, see the EXTfldval section on memory issues.

Structures and Enumerations

listtype

An enum defining the data storage method used by the list object.

- listVlen Variable length data will be stored.
- listScol Simple text list storage with support for a qlong value 'mark' on each row. This list ONLY supports one column.

EXTsortItem

A structure that defines sorting information for a single column. It has the following members.

```
typedef struct
{
  qshort  mSortColumn;
  qbool   mUpperCase;
  qbool   mDescending;
} EXTsortItem;
```

- **mSortColumn -** The column number to be sorted. Columns number start from 1.
- **mUpperCase -** qtrue if the column values should be treated as uppercase during sort.
- **mDescending -** qtrue if the column values should be sorted in descending order.

EXTsortStruct

A structure that defines a group of sort fields (a group of EXTsortItem objects)

```
typedef struct
{
  qshort          mSortCount;
  EXTsortItem     mSortLines[cMaxSortItems];
} EXTsortStruct;
```

- **mSortCount -** The number of sort items to use from this structure.
- **mSortLines[cMaxSortItems] -** An array of sort items.

cMaxSortItems is the maximum number of columns that can be used on a sort.

EXTqlist Class Reference

EXTqlist::EXTqlist()

EXTqlist::EXTqlist()

The constructor for an empty EXTqlist object. The EXTqlist will not contain any valid data and cannot be used until EXTqlist::clear has been called.

EXTqlist::EXTqlist()

EXTqlist::EXTqlist(listtype pListType)

The constructor for an empty EXTqlist object.

- **pListType** - The type of list to initialize the new list object as.

EXTqlist::EXTqlist()

EXTqlist::EXTqlist(lsttype* pList)

The constructor for an Omnis list. This constructor does not make a copy of the list data, so there is no need to destroy the list data by assigning qnil.

- **pList** - Points to the internal Omnis list.

See also EXTqlist::getLstPtr

EXTqlist::EXTqlist()

EXTqlist::EXTqlist(EXTqlist* pListData)

The constructor for a new EXTqlist object based on existing list data. EXTqlist * information can be retrieved from EXTfldval objects.

- **pListData** - The list data to build a list object from.

EXTqlist::EXTqlist() (v3.0)

```
EXTqlist::EXTqlist( qbyte* pAdd, qlong pLen, qret* pErr = NULL)
```

The constructor for a new EXTqlist object based on existing list data in disk format. The disk format list data must have been created previously by calling EXTqlist::getBinLen and EXTqlist::getBinary.

- **pAdd** – Address of the list data in binary form.
- **pLen** – Length in bytes of the list data.
- **pErr** – Optional error return. Returns e_ok if the list was constructed successfully.

See also EXTqlist::getBinLen, EXTqlist::getBinary

EXTqlist::~EXTqlist()

```
EXTqlist::~EXTqlist()
```

The destructor for an EXTqlist object.

EXTqlist::addCol()

```
qshort EXTqlist::addCol( qshort pCol, ffttype
pFft, qshort pFdp, qlong pFldLen, strxxx*
pClassname = NULL, strxxx* pColumnname
= NULL )
```

Adds a new column to the list.

- **pCol -** The column number to insert at.
- **pFft -** The data type for the new column.
- **pFdp -** The sub-data type for the new column - see EXTfldval.
- **pFldLen -** The data length for the new column.
- **pClassname** - Specifies the optional class name.
- **pColumnname** - Specifies the optional column name.

Note: If you do not specify column types you may encounter problems sorting lists columns.

EXTqlist::addCol()

```
qshort EXTqlist::addCol( ffttype pFft, qshort
pFdp, qlong pFldLen=0, strxxx* pClassname)
```

Adds a new column to the list.

- **pFft -** The data type for the new column.
- **pFdp -** The sub-data type for the new column - see EXTfldval.
- **pFldLen -** The data length for the new column.
- **pClassname** - Specifies the optional class name.
- **returns** – The new column in the list, zero if unsuccessful.

Note: If you do not specify column types you may encounter problems sorting lists columns.

EXTqlist::addColEx() (v5.0)

```
qshort EXTqlist::addColEx(qshort col, ffttype
fft, qshort fdp, qlong fln, strxxx* classname,
strxxx* columnname, qbool noclear)
```

Adds a new column to the list with support for additional attributes.

- **col –** 1-based column number of the new column.
- **fft –** Omnis data type of the new column.
- **fdp –** Omnis sub-type of the new column.
- **classname –** Stores an optional classname with the column definition.
- **columname –** The name of the new column.
- **noclear –** If qtrue, the contents of the list are left intact after the new column is added, otherwise the list contents are cleared.

EXTqlist::clear()

```
void EXTqlist::clear( listtype pListType )
```

Clears the list's contents, definition, and resets its type.

EXTqlist::clearRow()

```
qret EXTqlist::clearRow( qlong pRow )
```

Clears the contents from a row in the list.

- **pRow** - The row number to be cleared
- **returns** - e_ok if the row was cleared successfully.

EXTqlist::colCnt()

```
qshort EXTqlist::colCnt()
```

Returns the number of columns used in this list.

- **returns** - Returns the column count.

EXTqlist::convertEncoding() (v4.2)

```
void EXTqlist::convertEncoding(qbool
pSrcIsUnicode, qbool pDestIsUnicode)
```

Converts the encoding of character data stored in the list - only suitable for lists with a definition that is allowed for a web service parameter or return value. convertToEncoding(qtrue,qfalse) causes all character data stored in the list to be converted to non-Unicode Omnis character set data. convertToEncoding(qfalse,qtrue) causes character data stored in the list to be converted to Unicode data.

- **pSrcIsUnicode** – If qtrue, indicates that text written to the list is Unicode data.
- **pDestIsUnicode** - If qtrue, indicates that text read from the list should be returned as Unicode data.

EXTqlist::copyDef()

```
qbool EXTqlist::copyDef( EXTqlist pList, qbool pRedefine )
```

Copies the list definition from the passed list object to this list object.

- **pList -** The list to take the definition from.
- **pRedefine -** qtrue if this list is defined from empty, or columns are redefined.
- **returns** - qtrue if the definition copy was successful.

EXTqlist::defineFromSQLClass() (v4.2)

```
qbool EXTqlist::defineFromSQLClass(strxxx
&pSQLClassName, strxxx &pErrorText)
```

Defines the list object from the specified Omnis schema class returning qtrue on success, qfalse otherwise.

- **pSQLClassName** - The name of an Omnis schema class to use.
- **pErrorText** – An error message returned in the event that the list could not be defined.

EXTqlist::deleteRow()

```
qret EXTqlist::deleteRow( qlong pRow )
```

Deletes a row from the list.

- **pRow** - The row number to be deleted.
- **returns** - e_ok if the row was deleted successful.

EXTqlist::dup()

```
qbool EXTqlist::dup(EXTqlist * pList)
```

Duplicates the contents of pList in to this list.

- **pList -** The lists containing the data to be duplicated.
- **returns** - Returns qtrue if the data was duplicated successfully.

EXTqlist::empty()

```
qret EXTqlist::empty()
```

Clears the list's contents, leaving the list definition and column types unchanged.

EXTqlist::findCol()

```
qshort EXTqlist::findCol( strxxx &pColName )
```

Returns the column number matching the specified column name. For use with existing / pre-defined EXTqlists.

- **pColName –** The name of a column in the list.
- **returns –** Returns the ordinal column number, or zero if the column name was not found.

EXTqlist::getBinary()

```
void EXTqlist::getBinary( qchar* pDiskAddress )
```

Copies the contents of the list object to the address supplied, storing it as a simple flat buffer. The list can be reconstructed with the correct EXTqlist constructor.

- **pDiskAddress -** The address to save the list contents to.

EXTqlist::getBinLen()

```
qlong EXTqlist::getBinLen()
```

Returns the size needed to store the contents of the list object as a simple flat buffer.

- **returns -** The length needed to store to disk.

EXTqlist::getCol()

```
void EXTqlist::getCol(qshort pCol, qbool
pInclfilename, strxxx& pName)
```

Retrieves the column name for the column specified.

- **pCol -** The column number to retrieve the name.
- **pInclfilename -** qtrue if the filename should be included.
- **pName -** The string variable populated with the column name after the call.

Note: The name of a column corresponds to the name used when the list was defined using the 'Define list' Omnis command, or using Omnis list notation.

EXTqlist::getCol()

```
void EXTqlist::getCol(qshort pCol, strxxx& pName)
```

Retrieves the column name for the column specified.

- **pCol -** The column number to retrieve the name.
- **pName -** The string variable populated with the column name after the call.

Note: The name of a column corresponds to the name used when the list was defined using the 'Define list' Omnis command, or using Omnis list notation.

EXTqlist::getColType()

```
void EXTqlist::getColType( qshort pCol,
ffttype& pFft, qshort & pFdp ) void
EXTqlist::getColType( qshort pCol, ffttype&
pFft, qshort & pFdp, qlong & pLen )
```

Retrieves data type information from a column number.

- **pCol -** The column number for which to retrieve data type information.
- **pFft -** The data type for the column is returned here.
- **pFdp -** The sub-data type for the column is returned here.
- **pLen -** The maximum data length of the column.

EXTqlist::getColVal()

```
void EXTqlist::getColVal( qlong pRow, qshort
pCol, ffttype pFft, qshort pFdp, EXTfldval&
pFval )
```

Returns the contents from a row and column in the form of a EXTfldval object. The data can be optionally converted.

- **pRow -** The list row to access.
- **pCol -** The list column to access.
- **pFft -** The type the returning data should be converted to.
- **pFdp -** The type the returning data should be converted to (sub type) see EXTfldval.
- **pFval** - The EXTfldval object modified to hold the contents of the row/column.

Note: The '**pFval**' parameter is a copy of the columns contents. The memory associated with the copy is deleted when the '**pFval**' parameter is deleted.

EXTqlist::getColVal()

```
qbool EXTqlist::getColVal( qlong pRow,
qshort pCol, EXTfldval& pFvalp)
```

Populates a read-only EXTfldval object with the data for row/column.

- **pRow -** The list row to access.
- **pCol -** The list column to access.
- **pFvalp -** An EXTfldval modified to allow access to row/column.

Returns qtrue if successful, qfalse otherwise. Developers should note that if qfalse is returned (e.g. when requesting a row greater than the row count) the contents of pFvalp remain unchanged.

Note: As the EXTfldval object is marked as read-only, any calls to modify the EXTfldvals' data (e.g. via setChar, setLong etc..) will fail. If you wish to modify the lists data you must use EXTqlist::getColValRef with pWillAlter set to qtrue.

EXTqlist::getColValRef()

qbool EXTqlist::getColVal(qlong pRow, qshort pCol, EXTfldval& pFvalp, qbool pWillAlter)

Populates a EXTfldval object with the data for row/column.

- **pRow -** The list row to access.
- **pCol -** The list column to access.
- **pFvalp -** An EXTfldval modified to allow access to row/column.
- **pWillAlter -** qtrue if you want to modify the contents of **pFvalp.**

Returns qtrue if successful, qfalse otherwise. Developers should note that if qfalse is returned (e.g. when requesting a row greater than the row count) the contents of pFvalp remain unchanged.

The EXTfldval populated by getColValRef can be considered to be a pointer to the data in the list line for which getColValRef was called.

At the C++ level, a list only has one set of pointers for its columns, and as the line changes (e.g. when a new call is made to getColValRef) the data addressed by the pointers is updated with the content of the new line.

Note: If pWillAlter is false (i.e. equivalent to calling EXTqlist::getColVal(qlong, qshort, EXTfldval&) then the EXTfldval object is marked as read-only. Consequently any calls to modify the EXTfldvals' data (e.g. via setChar, setLong etc..) will fail.

EXTqlist::getCurRow()

qlong EXTqlist::getCurRow()

Returns the current row number.

- **returns** - Returns the current row number associated with this list.

EXTqlist::getLstPtr()

lsttype* EXTqlist::getLstPtr()

Returns the pointer to the Omnis list data.

EXTqlist::getColFlags() (v4.1)

qlong EXTqlist::getColFlags(qshort col)

Returns the flags describing an individual column of the list. Flag values are listed in EXTDAM.HE and include the following: cTABflagIsPrimaryKey, cTABflagExcludeFromInsert, cTABflagExcludeFromUpdate, cTABflagCalculated, cTABflagSequenceType & cTABflagExcludeFromWhere

- **col** – 1 based column number to inspect.

EXTqlist::getColNullInfo() (v4.1)

qbool EXTqlist::getColNullInfo(qshort col)

Returns qtrue if the list column supports NULL values, qfalse otherwise.

- **col** – 1 based column number to inspect.

EXTqlist::getRow()

void EXTqlist::getRow(qlong pRow, str255* pString)

Retrieves the string value added with either ::**insertRow** or ::**putRow** function.

- **pRow** - The row to extract the string from.
- **pString** - The string to copy the list sting into.

EXTqlist::getRow()

void EXTqlist::getRow(qlong pRow, qlong& pMark)

Retrieve the long 'mark' value added with either ::**insertRow** or ::**putRow** function.

- **pRow** - The row to extract the 'mark' value from.
- **pMark** - The mark value stored in the list is returned here.

EXTqlist::getRowCrb()

qcrb EXTqlist::getRowCrb(qlong pRow, qbool pWillAlter = qfalse)

Returns a pointer to the Omnis data collection of a list.

- **pRow** - specifies the row of the list the data collection will reference.
- **pWillAlter** - specify qtrue if you want to make changes to the data of the row.

Returns the pointer to the list's data collection.

Warning: Once you have retrieved the pointer to the data collection, changing the current row of the list will change the data in the collection to that of the new current row. But the new row will not be marked as changed until you execute another getRowCrb(rowNumber, qtrue). If you do not mark a row as changed, any changes you make to the data will be lost when the current row is changed.

```
// example changing the data of column 3 in row 2 of a list
EXTqlist lst(listVlen);
lst.setFinalRow(5);
qcrb crb = lst.getRowCrb( 2, qtrue );
CRBsetLong( crb, 3, 255 );
// no further action needs to be taken.
// column 3 of row 2 will now contain the value 255
```

EXTqlist::getRowMax()

```
qlong EXTqlist::getRowMax()
```

Returns the maximum number of rows this list can have.

- **returns** - Returns the maximum row count.

EXTqlist::insertRow()

```
qlong EXTqlist::insertRow(qlong pBefore = 0,
str255* pText = NULL, qlong pMark = 0 )
```

Inserts a new row into the list.

- **pBefore** - The row number to insert the new row before. 0 indicates the end of the list.
- **pText** - The text to be inserted in to the list for the new row. **è Note: listScol only.**
- **pMark** - A long value that can be added to identify the new row. **è Note: listScol only.**
- **returns** - The new line number is returned if the insert was successful.

EXTqlist::isRowSelected()

```
qret EXTqlist::isRowSelected( qlong pRow, qbool pIsSaved = qfalse )
```

Returns the selected state of a row.

- **pRow** - The row to test.
- **pIsSaved** - qtrue if the check is to be made on the saved selected states.
- **returns** - qtrue if the row is selected, and qfalse if the row is not selected.

EXTqlist::loadRows()

```
void EXTqlist::loadRows( qchar* pRowData )
```

Takes a '+' separated string and converts it into rows in the list, for example, "Row1+Row2+Row3".

- **pRowData -** A pointer to a c-style string to be converted into rows for the list.

The list is redefined as listScol type.

EXTqlist::operator = (qniltype qnil)

```
void EXTqlist::operator =(qniltype qnil1)
```

Frees the memory used by the list.

EXTqlist::putColVal()

```
void EXTqlist::putColVal( qlong pRow, qshort
pCol, EXTfldval& pFval )
```

Sets the contents of a column value from the passed EXTfldval object.

- **pRow -** The list row to access.
- **pCol -** The list column to access.
- **pFval -** The EXTfldval object who's data should be stored in the column.

Note: The '**pFval**' parameter's contents are duplicated and stored in the list. The memory for the duplicated contents is owned by the list object, and the memory used by the '**pFval**' parameter is deleted when the parameter is deleted.

EXTqlist::putRow()

```
qret EXTqlist::putRow( qlong pRow, str255*
pText = NULL, qlong pMark = 0 )
```

Replaces the contents for a particular row.

- **pRow** - The row number to be modified.
- **pText** - The text to be stored in the list for the new row. **è Note: listScol only.**
- **pMark** - A long value that can be added to the new row. **è Note: listScol only.**
- **returns** - e_ok if the contents were replaced successfully.

EXTqlist::rowCnt()

```
qlong EXTqlist::rowCnt()
```

Returns the number of rows in this list.

- **returns** - Returns the row count.

EXTqlist::selectRow()

```
qret EXTqlist::selectRow(qlong pRow, qbool
pSelect, qbool pIsSaved = qfalse )
```

Selects or deselects a row in the list.

- **pRow** - The row to select or deselect.
- **pSelect** - qtrue if the row is to be selected.
- **pIsSaved** - qtrue if the change of state is to happen to the lists saved selection buffer.
- **returns** - e_ok if the lines state changed.

EXTqlist::setCol() (v5.0)

```
void EXTqlist::setCol(qshort col, strxxx* name)
```

Changes the name of an existing list column.

- **col** - 1-based column number.
- **name** - New column name.

EXTqlist::setCurRow()

```
qret EXTqlist::setCurRow( qlong pCurrentRow )
```

Sets the current row number for this list object.

- **pCurrentRow** - The new current row for this list.
- **returns** - e_ok if the current row changed.

EXTqlist::setFinalRow()

```
qret EXTqlist::setFinalRow( qlong pLastRow )
```

Modifies the list to contain the number of rows as specified by **pLastRow**. If necessary rows are deleted or empty rows are added.

- **pLastRow** - The new final row number of the list.
- **returns** - e_ok if the final row was set.

EXTqlist::setRowMax()

```
qret EXTqlist::setRowMax( qlong pLastMaxValue )
```

Prevents the list from extending beyond the value passed.

- **pLastMaxValue** - The new last row for the list.
- **returns** - e_ok if the lists max line is changed.

Note: If '**pLastMaxValue**' is less than the number of rows the list already has, the number of rows the list currently has becomes the new maximum, not '**pLastMaxRow**'.

EXTqlist::sort()

qbool EXTqlist::sort(EXTsortStruct* pSortItems)

Sorts the list object according to the sort options set in the supplied sorting structure.

- **pSortItems -** The sorting options for columns in the list.
- **returns** - Returns qtrue if the list was sorted, and qfalse if the sort failed.

Example:

```
EXTqlist* paramlist = new EXTqlist ( listVlen );
for ( qshort rows = 1; rows<=10; rows++ )
{
  qlong paramrow = paramlist->insertRow();
  // Parameter name
  paramlist->getColValRef(paramrow, 1, cval, qtrue);
  cval.setChar( newCharValue );
  // fft Data type
  paramlist->getColValRef(paramrow, 2, cval, qtrue);
  cval.setLong( newLongValue2 );
  // EXTD_ flags
  paramlist->getColValRef(paramrow, 3, cval, qtrue);
  cval.setLong( newLongValue3 );
}
paramlist = qnil;
delete paramlist;
```

Chapter 10—EXTfldval Reference

Introduction

The EXTfldval class gives your external components a generic data storage object. All data passed to and from Omnis and your component is in the form of a EXTfldval object. This object can store a variety of data-types, offering some basic conversion between various data formats. For example, you can put a long numeric value into the EXTfldval class and retrieve it in string or data form.

EXTfldval Memory Issues

The EXTfldval class can be constructed in two ways, one as a reference to a known Omnis field such as #S1, or as an individual object. You should think of the class as a container, which either has some data stored within it, or as a reference to another data value. The container can store a range of data-types from pictures and lists to simple data types such as numbers and strings.

The memory associated with the EXTfldval class is *always* owned by Omnis, with the possible exception when the container is storing a list (see below). If the EXTfldval object is being used to store data as opposed to being used as a reference, the memory is deleted when the object is deleted, either by the 'delete' operator or as a result of the EXTfldval object going out of scope.

Some API calls such as ECOaddParam cause the memory used by the EXTfldval object to be disassociated, thus Omnis takes ownership of the memory from the EXTfldval and uses it elsewhere. When this happens, the EXTfldval object does not delete the memory on destruction. These APIs will be marked in this document.

```
// Storing a string and getting a number
void myMethod()
{
```

```
  EXTfldval myFldval;
  str255 stringWithNumber("100");
  myFldval.setChar(stringWithNumber );
  qlong number = myFldval.getLong();
  if ( number==100 )
  {
    // string was converted to a number ok.
  }
}
```

In the above example, an EXTfldval object is storing a string. When the object goes out of scope, Omnis will delete the memory used to store the string. Below is another example where the data being stored is unknown (binary), but again Omnis will delete the memory when the object goes out of scope or is deleted. All the example has to do is take care of deleting the memory it used to assign the EXTfldval object.

```
// Storing some binary information in an EXTfldval
void myMethod()
{
  EXTfldval myFldval;
  HGLOBAL someBinary = NULL;
  // Allocates some memory and returns it.
  someBinary = getSomeBinaryData();
  if ( someBinary )
  {
    myFldval.setHandle( someBinary, fftBinary );
    MEMglobalFree( someBinary );
  }
}
```

EXTfldvals and EXTqlists

Generally, all EXTfldval objects have their own memory to store the data contents. The one exception to this rule can be the storage of the list object, EXTqlist. EXTqlist objects can have their own data storage (see EXTqlist object). Some lists can become very large, such as lists of rows received from a SQL database. When Omnis needs to pass lists objects around, it uses the EXTfldval object. For the sake of memory and speed, the EXTfldval object can carry a reference to a list object, rather than a copy of the object.

When you assign a list to an EXTfldval object, you have to specify if the data to be stored will be a reference to a list object, or if it will store the contents of a list object. If you want to store a reference, the reference is *only* valid while the EXTqlist is valid. That is, if you delete the EXTqlist, the reference stored is no longer valid, and when used will cause a crash.

If you decide to store the contents of a list object, the EXTfldval takes ownership of the memory used to store the content of the list from EXTqlist object, and as such the EXTqlist contents should not be cleared by using the **qnil** assignment as it no longer owns the memory.

Here is an example of using an EXTfldval object to store the contents of a EXTqlist.

```
// Using a EXTfldval to store an EXTqlist
void myMethod( EXTfldval& pFldval )
{
  EXTqlist* tempList = new EXTqlist( listVlen );
  for ( qshort i = 0; i<10; i++ )
  {
    EXTfldval cval;
    qlong newRow = tempList->insertRow();
    tempList->getColValRef(newRow, 1, cval, qtrue );
    cval.setLong( i );
  }
  pFldval.setList( &tempList, qtrue );
  delete tempList;
}
```

In the above example, an EXTfldval object passed in to the function will be given a list to store. The temporary EXTqlist object is first filled with some new rows, then the contents of the list are transferred to the EXTfldval object. At the end of the function, the list is deleted.

Getting a list from an EXTfldval

Once you have been given an extfldval object with a list in it, you can retrieve it in two ways, as a complete list object, or as a reference. Remember in the EXTqlist section you specified the EXTqlist object can be a reference to a list, or an individual object. When you ask the EXTfldval object for a list, you can choose what sort of list is returned. If you choose a complete object, a new EXTqlist is created with a duplicate of the list contents associated with the EXTfldval object. The memory for this object needs to be emptied using the **qnil** assignment operator before the object is deleted. If you choose the reference, a new EXTqlist is created, but as a reference to another EXTqlist object. This also needs to be deleted.

Here is an example building on from the previous example. It uses the EXTfldval object that was passed in to the function above to extract a list from.

```
// Using a EXTfldval to store an EXTqlist
void myMethod( EXTfldval& pFldval )
{
  …( see above )
}
// Takes a complete copy of the list, adds a row and frees the list
void alterList()
{
  EXTfldval myFldval;
  EXTqlist* myQlist;
  // call to fill a list
  myMethod( myFldval );
  // now get a duplicate of the list stored
  myQlist = myFldval.getList(qtrue);
  // add another row
  EXTfldval cval;
  qlong newRow = myQlist.insertRow();
  myQlist->getColValRefPtr(newRow, 1, cval, qtrue );
  cval.setLong( 999 );
  // do something else
  callAnotherFunction( myQlist );
  // free duplicated list
  *myQlist = qnil;
  delete myQlist;
  // real list stored in myFldval is deleted as scope ends.
}
```

Here is another example, but this does not take a copy and operates on a reference to the list.

Note: If you get a EXTqlist as a reference from an EXTfldval, you MUST delete the object, but do not use the qnil assignment operator because that clears the original.

```
// Gets a reference to a list, adds a row and frees reference.
void alterAnotherList()
{
  EXTfldval myFldval;
  EXTqlist* myQlist;
  // call to fill a list
  myMethod( myFldval );
  // now get a reference to the list
  myQlist = myFldval.getList(qfalse);
  // add another row
  EXTfldval cval;
  qlong newRow = myQlist.insertRow();
```

```
  myQlist->getColValRef(newRow, 1, cval, qtrue );
  cval.setLong( 999 );
  // do something else
  callAnotherFunction( myQlist );
  // free reference list
  delete myQlist;
}
```

The next example demonstrates a crash, as an EXTqlist reference is used after the EXTfldval has been deleted.

```
// Gets a reference to a list, and uses it after the original
// has been deleted.
void doNotDoThis()
{
  EXTqlist* myQlist;
  // extra scope added for example
  {
    EXTfldval myFldval;
    // call to fill a list
    myMethod( myFldval );
    // now get a reference to the list
    myQlist = myFldval.getList(qfalse);
  }
  // at this point, the EXTfldval has been deleted, so the list
  // reference no longer points to a good list
  // Any call below that uses 'myQlist' causes a crash.
  // add another row
  EXTfldval cval;
  qlong newRow = myQlist.insertRow();
  myQlist.getColValRef(newRow, 1, cval, qtrue );
  cval.setLong( 999 );
  // do something else
  callAnotherFunction( myQlist );
  // free reference list
  delete myQlist;
}
```

If the above example had taken a copy of the list as shown below, the crash would not occur.

```
// extra scope added for example
{
  EXTfldval myFldval;
  // call to fill a list
  myMethod( myFldval );
  // take a copy
  myQlist = myFldval.getList(qtrue);
}
```

As it is a complete copy, deleting the EXTqlist* at the end of the function also needs to delete the contents like this.

```
// free reference list
*myQlist = qnil;
delete myQlist;
```

Enumerations and Structures

fftype

An enum defining the data storage types that the EXTfldval supports.

- fftNone
  No valid object is stored
- fftCharacter
  Character or national character storage
- fftBoolean
  Simple Boolean storage.
- fftDate
  Date, Time and DateTime storage
- fftNumber
  Real number storage
- fftInteger
  4 or 2 byte integer storage
- fftPicture
  Picture image storage
- fftBinary
  Binary storage
- fftList
  List storage
- fftRow
  Row storage
- fftObject
  Object storage
- fftCrb
  Omnis data collection (see EXTqcrb)
- fftCalc
  Tokenised calculation
- fftConstant
  Omnis constant

In addition to the major data types, some data types such as **fftDate**, **fftNumber** need to know exactly what sort of date or number to store. This is accomplished using a **subtype**.

The subtypes for **fftCharacter** are:

- dpFcharacter
  Character data storage
- dpFnational
  National character data storage

The subtypes for **fftDate** are:

- dpFdate1900
  Short date field 1900-1999
- dpFdate1980
  Short date field 1980-2079
- dpFdate2000
  Short date field 2000-2099
- dpFdtime1900
  Date and time as above

- dpFdtime1980
  Date and time as above
- dpFdtime2000
  Date and time as above
- dpFtime
  Short time field
- dpFdtimeC
  Date and time including century

The subtypes for **fftNumber** are:

- dpFmask
  a mask for accessing the number field decimal places.
- dpFsnumber
  Short number fields. Allows 0 and 2 decimal places.
- dpFloat
  Floating number

The subtypes for **fftInteger** are:

- 0
  4 byte integer
- dpFsinteger
  2 byte integer

The subtype for **fftList** are:

- 0
  Normal list
- dpFrow
  Row variable

Subtype that can be used for all ffttypes for default settings is

Default subtype. This varies depending on the fft.

fftCharacter - dpDefault results in a **dpFcharacter** subtype.

fftBoolean - dpDefault is ignored.

fftDate - dpDefault results in a **dpFdtime1980** subtype.

fftNumber - dpDefault results in a zero decimal place number.

fftInteger - dpDefault results in a **short integer**.

For all other types dpDefault is ignored.

crbFieldInfo (V2.2)

This structure is used with ECOgetCrbFieldInfo to get format information of an Omnis variable. The members are:

```
struct crbFieldInfo
{
  ffttype fft;
  qshort fdp;
  qlong fln;
  qbool fdx;
  qshort iln;
};
```

- Fft
  the data type
- Fdp
  the data sub type. See ffttype description for more information
- Fln
  for character data it specifies the maximum length of the field
- Fdx
  if true, the field is indexed.
- Iln
  if the field is indexed, it specifies the index length

datestamptype

Structure used for passing date and time information in and out of the EXTfldval object. The members are:

```
typedef struct
{
  qshort mYear;
  qchar mMonth;
  qchar mDay;
  qchar mHour;
  qchar mMin;
  qchar mSec;
  qchar mHun;
  qchar mDateOk;
  qchar mTimeOk;
  qchar mSecOk;
  qchar mHunOk;
} datestamptype;
```

- mYear
  Year values. e.g. 1900.
- mMonth
  Month values. 1-12
- mDay
  Day values. 1-31
- mHour
  Hour values. 1-12
- mMin
  Minute values. 0-59
- mSec
  Second values. 0-59
- mHun
  Hundredth of second values
- mDateOk
  qtrue if the date is valid
- mTimeOk
  qtrue if the time is valid
- mSecOk
  qtrue if the seconds are valid
- mHunOk
  qtrue if the hundredth of seconds are valid

EXTfldval Class Reference

EXTfldval::EXTfldval()

EXTfldval::EXTfldval()

The constructor for an empty EXTfldval container.

EXTfldval::EXTfldval()

EXTfldval::EXTfldval(qfldval pData=0)

The constructor for a EXTfldval container which will refer to the defined pData.

EXTfldval::EXTfldval()

EXTfldval::EXTfldval(strxxx& pVariableName, qbool pWillAlter =qfalse, locptype* pLocp = NULL)

The constructor for an EXTfldval container that sets itself up to refer to a pre-defined named field. e.g. #S1

- **pVariableName -** The field to associate the new EXTfldval object with.
- **pWillAlter -** qtrue if you want to alter the data.
- **pLocp** - points to the context. The EXTCompInfo structure which is passed to external components contains two context pointers. The context pointer mInstLocp points to the context of the class instance which contains the component. The context pointer mLocLocp points to the context of the calling method.

EXTfldval::~EXTfldval()

EXTfldval::~EXTfldval()

EXTfldval::operator =()

void EXTfldval::operator =(EXTfldval& pFval)

Assigns (copies) the contents of pFval to the object.

Example:

```
EXTfval myFldVal = pFldVal;
```

EXTfldval::compare()

qshort EXTfldval::compare(EXTfldval& pFldval)

Compares the contents of two EXTfldval objects.

- **pFldVal** - The EXTfldval object to compare against.
- **returns** - Returns 0 if both objects match.
  Returns -1 if pFldval is less than this.
  Returns 1 if pFldval is greater than this.

EXTfldval::compare()

qshort EXTfldval::compare(EXTfldval& pFldval, qbool pIgnoreCase)

Compares the character contents of two EXTfldval objects.

- **pFldVal** - The EXTfldval object to compare against.
- **pIgnoreCase** - qtrue if the case of the characters is ignored.
- **returns** - Returns 0 if both objects match.
  Returns -1 if pFldval is less than this.
  Returns 1 if pFldval is greater than this.

EXTfldval::compare()

qshort EXTfldval::compare(strxxxx& pString, qbool pIgnoreCase)

Compares the character contents of the EXTfldval object and pString.

- **pString** – A string object to compare against.
- **pIgnoreCase** - qtrue if the case of the characters is ignored.
- **returns** - Returns 0 if both the string in the EXTfldval and pString match.
  Returns -1 if pString is less than this.
  Returns 1 if pString is greater than this.

EXTfldval::concat()

void EXTfldval::concat(qchar pChar)

Concatenates a character on to the end of the existing stored data. If the data is not in character form, it is converted first.

- **pChar** - The character to concatenate.

EXTfldval::concat()

void EXTfldval::concat(qchar* pAddress, qlong pDataLen)

Concatenates a range of characters on to the end of the existing stored data. If the data is not in character form, it is converted first.

- **pAddress** - A buffer to some data.
- **pDataLen** - The number of characters to concatenate.

EXTfldval::concat()

void EXTfldval::concat(EXTfldval* pFldval)

Concatenates characters from another EXTfldval object on to the end of the existing stored data. If the data is not in character form, it is converted first.

- **pFldval** - The EXTfldval objects who's data is concatenated.

EXTfldval::concat()

```
void EXTfldval::concat( strxxx& pString )
```

Concatenates a string on to the end of the existing stored data. If the data is not in character form, it is converted first.

- **pString** - The string to be concatenated.

EXTfldval::conv()

```
qbool EXTfldval::conv( ffttype pDataType, qshort pSubDataType )
```

Tries to convert to another data type.

- **pDataType** - The data type to try to convert to.
- **pSubDataType** - The sub data type to try to convert to.
- **returns -** Returns qtrue if the conversion was successful.

EXTfldval::evalCalculation() (Studio 2.0)

```
qbool EXTfldval::evalCalculation( EXTfldval&
pResult, locptype* pLocp, EXTqlist* pList =
NULL, qbool pUseCache = qtrue)
```

Evaluates the calculation stored in the EXTfldval.

- **pResult** - The result of the calculation is returned in this parameter.
- **pLocp -** The EXTCompInfo structure which is passed to external components contains two context pointers. The context pointer mInstLocp points to the context of the class instance which contains the component. The context pointer mLocLocp points to the context of the calling method.
- **pList** - If a list is specified, the calculation is evaluated against the list. If the calculation refers to field names which exist as columns within the list, the data in the current row of that column is used.
- **pUseCache** - If true, Omnis will use a global cache for storing the result. This increases efficiency when dealing with large amounts of data, but it is potentially dangerous, since the there is only one cache, which is reused when another calculation is evaluated. Unless performance is an issue, always pass qfalse.

See also EXTfldval::getCalculation, EXTfldval::setCalculation

EXTfldval::getBinary()

```
void EXTfldval::getBinary(qlong pBufferLen,
qchar* pBuffer, qlong& pRealLen )
```

Retrieves the object's data as binary.

- **pBufferLen -** The maximum size of the buffer.
- **pBuffer** - The buffer to receive a copy of the data.
- **pRealLen** - Returned is the real length of the data copied.

EXTfldval::getBinLen()

```
qlong EXTfldval::getBinLen()
```

Returns the size of the object stored, in bytes.

See also EXTfldval::getCharLen()

EXTfldval::getBool()

```
qshort EXTfldval::getBool( qbool* pBool = 0 )
```

Retrieves a boolean value.

- **pBool -** Returns qtrue if the EXTfldval object result can be used.
- **returns** - If supplied, returns 0, 1 or 2.

Note: Boolean values have the following values:

Return (0) - value is not set (fldval is empty or null).
Return (1) - value is qfalse.
Return (2) - value is qtrue.

EXTfldval::getCalculation() (Studio 2.0)

```
void EXTfldval::getCalculation( locptype*
pLocp, qshort &pCalculationType, EXTfldval
&pText)
```

Returns the calculation type and text representation of a tokenized calculation.

- **pLocp -** The EXTCompInfo structure which is passed to external components contains two context pointers. The context pointer mInstLocp points to the context of the class instance which contains the component. The context pointer mLocLocp points to the context of the calling method.
- **pCalculationType** - The calculation type ctySquare or ctyCalculation is returned in this parameter.
- **pText** - The textual representation of the calculation is returned in the given EXTfldval.

See also EXTfldval::setCalculation, EXTfldval::evalCalculation

EXTfldval::getChar()

```
void EXTfldval::getChar(strxxx& pString,
qbool pZeroEmpty = qfalse )
```

Returns a string version of the data stored.

- **pString** - The string to copy the data into.
- **pZeroEmpty** - if true and EXTfldval stores a number, the pString will be empty if that number is zero.

EXTfldval::getChar()

```
strxxx& EXTfldval::getChar(qbool pZeroEmpty = qfalse )
```

Returns a reference to the string version of the data stored.

- **pZeroEmpty** - if true and EXTfldval stores a number, the returned string reference will be empty if that number is zero.
- **Returns** – A string reference to the string version of the data stored.

EXTfldval::getChar()

```
strxxx& EXTfldval::getchar(std::string& pString)
```

Returns a std::string version of the data stored.

- **pString** – the std::string to copy the data into.
- **Returns** – A std::string version of the data stored.

EXTfldval::getChar()

```
void EXTfldval::getChar( qlong pMaxLen,
qchar* pAddress, qlong& pRealLen, qbool
pZeroEmpty = qfalse )
```

Returns a string version of the data stored.

- **pMaxLen** - The maximum number of bytes allowed to copy into **pAddress.**
- **pAddress** - The buffer to copy the string into.
- **pRealLen** - The returned real length that was copied into pAddress.
- **pZeroEmpty** - if true and EXTfldval stores a number, the pString will be empty if that number is zero.

EXTfldval::getCharLen() (v4.1)

```
qlong EXTfldval::getCharLen()
```

When the fldval contains character data, getCharLen() returns the length of the data in characters.

See also EXTfldval::getBinLen()

EXTfldval::getConstant()

preconst EXTfldval::getConstant(preconst pMin, preconst pMax, qbool *pRet=0)

Returns the constant ID of the value stored with the fldval. The constant must be in the range specified by pMin and pMax.

- **pMin** – The constant range start ID. Please see the source file dmconst.he for valid values.
- **pMax** – The constant range end ID. Please see the source file dmconst.he for valid values.
- **pRet** – Optional pointer to a boolean result. If qfalse is returned, the value of the fldval did not conform to the given constant range.
- **Returns** – the constant ID of the value.

Example:

```
qbool ok;
preconst cid = fval.getConstant(preButtmodeF,preButtmodeL,&ok);
if (ok)
{
  // cid will be in the range preButtmodeF to preButtmodeL
}
```

See also EXTfldval::setConstant, EXTfldval::getLong(preconst,preconst,qbool*)

EXTfldval::getCrbRef()

qcrb EXTfldval::getCrbRef(qbool pDuplicate)

Returns an Omnis data collection.

- **pDuplicate** - If true, a copy is returned, which must be disposed of by calling CRBdestroy

EXTfldval::getDate()

void EXTfldval::getDate (datestamptype& pDateTime, qshort pSubDataType = dpDefault, qbool* pError = 0)

Retrieves the data stored as datetime information.

- **pDateTime -** The datetime structure to modify.
- **pSubDataType** - Defines what type of datetime is retrieved. See **dpDefault** above.
- **pError** - If supplied, returns qtrue of the date could be retrieved.

EXTfldval::getFldVal()

qfldval EXTfldval::getFldVal()

Returns the pointer to the Omnis data. Some print manager functions and structures require these data pointers instead of EXTfldval pointers.

- **returns -** An Omnis data pointer.

See also EXTfldval::setFldVal, EXTfldval::setOmnisData

EXTfldval::getHandle()

HGLOBAL EXTfldval::getHandle()

Returns a moveable block of memory which is a copy of the data stored in the EXTfldval.

- **returns -** A moveable block of memory.

EXTfldval::getHandle()

qHandle EXTfldval::getHandle(qbool pMakeCopy)

Returns an Omnis handle containing the data.

- **pMakeCopy**- If true, getHandle will make a copy of the data. You will be responsible for disposing of the handle by calling HANglobalFree()
- **returns -** An Omnis handle.

EXTfldval::getList()

EXTqlist* EXTfldval::getList(qbool pDuplicate)

Retrieves a list value.

- **pDuplicate -** qtrue if the returned object is a duplicate of the list in the EXTfldval object.
- **returns** - A new EXTqlist object. This must be deleted. NULL is returned if the EXTfldval object cannot return an EXTqlist object.

EXTfldval::getList()

void EXTfldval::getList(EXTqlist* pList, qbool pDuplicate)

Populates the supplied list with the list in the EXTfldval object.

- **pList** – A EXTqlist object which will be populated upon return. This value cannot be NULL.
- **pDuplicate -** qtrue if the EXTqlist object is a duplicate of the list in the EXTfldval object.

EXTfldval::getLong()

qlong EXTfldval::getLong(preconst pMin, preconst pMax, qbool *pRet=0)

Returns the long value of the constant stored with the fldval. The value must be in the range of values specified by the pMin and pMax constant IDs.

- **pMin** – The constant range start ID. Please see the source file dmconst.he for valid values.
- **pMax** – The constant range end ID. Please see the source file dmconst.he for valid values.
- **pRet** – Optional pointer to a boolean result. If qfalse is returned, the value of the fldval did not conform to the given constant range.
- **Returns** – the long value of the constant.

Example:

```
qbool ok;
qlong value = fval.getLong(preButtmodeF,preButtmodeL,&ok);
if (ok)
{
  // value will be in the range of values as specified by
  // constants preButtmodeF to preButtmodeL
}
```

See also EXTfldval::getConstant(), EXTfldval::setConstant

EXTfldval::getLong()

qlong EXTfldval::getLong()

Retrieves the value in the EXTfldval object as a qlong value.

- **returns -** A qlong value.

EXTfldval::getNum()

void EXTfldval::getNum(qreal& pNumValue, qshort& pSubDataType, qbool* pError=0)

Returns a number value.

- **pNumValue** - Variable to receive the numeric value.
- **pSubDataType**- The required decimal places. dpDefault does not convert.
- **pError** - If an error parameter is supplied, qtrue if the number could be converted.

EXTfldval::getObjInst()

qobjinst EXTfldval::getObjInst(qbool pDuplicate)

Retrieves a qobjinst pointer.

- **pDuplicate -** qtrue if the returned object is a duplicate of the qobjinst in the EXTfldval object.
- **returns** – A qobjinst pointer. NULL is returned if the EXTfldval object cannot return an qobjinst object.

EXTfldval::getOmnisField()

qbool EXTfldval::getOmnisField(strxxx& pVariableName, qbool pWillAlter)

Sets the EXTfldval object to refer to a pre-defined Omnis variable. e.g. #S1.

- **pVariableName -** The field to associate the new EXTfldval object to.
- **pWillAlter -** qtrue if you want to alter the data.
- **returns** - qtrue if the variable name was found and the EXTfldval object can be used.

EXTfldval::getType()

void EXTfldval::getType(ffttype& pDataType, qshort* pSubDataType = 0)

Retrieves the data type information from the EXTfldval object

- **pDataType**- Receives the data type.
- **pSubDataType**- Receives the sub data type if supplied.

EXTfldval::insertStr

void EXTfldval::insertStr(qlong pPos, const strxxx& pString)

Inserts a sub-string at a given index position.

- **pPos**- The index location at which to insert at. Index starts from 1.
- **pString**- The string to insert.

EXTfldval::isEmpty()

qbool EXTfldval::isEmpty()

Tests if the EXTfldval object contains no data.

- **returns** - Returns qtrue if the object is empty and qfalse if the object contains data.

EXTfldval::isList()

qbool EXTfldval::isList()

Tests if the EXTfldval object contains list data.

- **returns** - Returns qtrue if the object contains a list (**fftList**) as its data.

EXTfldval::isLongChar()

qbool EXTfldval::isLongChar()

Tests if the EXTfldval object contains character data and that the length is less than 256 characters.

- **returns** - Returns qtrue if the object contains character (**fftCharacter**) as its data and that the length is less than 256 characters.

EXTfldval::isNull()

qbool EXTfldval::isNull()

Tests if the EXTfldval object contains null data.

- **returns** - Returns qtrue if the object contains null (#NULL) data.

EXTfldval::isOmnisData()

qbool EXTfldval::isOmnisData()

Tests if the Omnis data pointer in the EXTfldval belongs to the EXTfldval.

- **returns** - Returns qtrue if the object contains null (#NULL) data.

See also EXTfldval::setOmnisData

EXTfldval::isReadOnly()

qbool EXTfldval::isReadOnly()

- **returns** - Returns qtrue if the data cannot be altered.

EXTfldval::pos()

qlong EXTfldval::pos(EXTfldval& pFldval)

Returns the position of a sub-string from **pFldval** in this EXTfldval object.

- **pFldval** - The EXTfldval to search for. (in character form)
- **returns** - 0 if the string in pFldval could not be found in this object. Returns a positive value to indicate the sub-string index location.

EXTfldval::pos()

```
qlong EXTfldval::pos( qchar* pAddreess, qlong pDataLen )
```

Returns the position of the sub-string pAddress in this EXTfldval object.

- **pAddress** - The address of a sub-string to search for.
- **pDataLen** - The length of the sub-string in bytes.
- **returns** - 0 if the data from **pAddress** could not be found in this object. Returns a positive value to indicate the sub-string index location.

EXTfldval::pos()

```
qlong EXTfldval::pos( qchar pChar )
```

Returns the position of a character in this EXTfldval object.

- **pChar**- The character to search for.
- **returns** - 0 if the character could not be found in this object. Returns a positive value to indicate the character's index location.

EXTfldval::pos()

```
qlong EXTfldval::pos( const strxxx& pString )
```

Returns the position of a string in this EXTfldval object.

- **pString**- The string to search for.
- **returns** - 0 if the string could not be found in this object. Returns a positive value to indicate the string's index location.

EXTfldval::replaceStr()

```
qbool EXTfldval::replaceStr( strxxx& pFindStr,
const strxxx& pReplaceStr, qbool
pIgnoreCase = qfalse)
```

Searches for a sub-string and if found, replaces with another string.

- **pFindStr**- The string to search for.
- **pReplaceStr** - The replacement string.
- **pIgnoreCase** - qtrue if the case during find can be ignored.
- **returns -** qtrue if the string was found and replaced successfully.

EXTfldval::replaceStr()

```
void EXTfldval::replaceStr(EXTfldval &
pFindObject, EXTfldval& pReplaceObject,
qbool pAll )
```

Searches for a sub-string extracted from pFindObject and if found, replaces with another string extracted from pReplaceObject.

- **pFindObject**- The EXTfldval object containing the string to search for.
- **pReplaceObject**- The EXTfldval object containing the string to replace with.
- **pAll -** qtrue if the call replaces all occurrences of the find string.

EXTfldval::setBinary()

```
void EXTfldval::setBinary( ffttype pDataType,
qchar* pAddress, qlong pDataLen, qshort
pSubDataType = dpDefault )
```

Stores data in binary form.

- **pDataType -** The type of data being stored.
- **pAddress** - The buffer to read data from and store.
- **pDataLen** - The length of the data to store.
- **pSubDataType** - The sub data type. This depends on the pDataType parameter.

EXTfldval::setBool()

```
void setBool(qshort pValue)
```

- **pValue -** The boolean value to be stored.

Note: Boolean values has the following values:
pValue(0) - value is not set (to store empty or NULL)
pValue(1) - value is qfalse
pValue(2) - value is qtrue

EXTfldval::setCalculation() (Studio 2.0)

```
qret EXTfldval::setCalculation( locptype*
pLocp, qshort pCalculationType, qchar*
pBuffer, qlong pLen, qlong* pError1=NULL,
qlong* pError2=NULL)
```

Tokenizes the specified calculation and stores it in the EXTfldval. If the calculation was not valid, the function returns an error code, and the starting and end positions of the offending part of the calculation.

- **pLocp -** The EXTCompInfo structure which is passed to external components contains two context pointers. The context pointer mInstLocp points to the context of the class instance which contains the component. The context pointer mLocLocp points to the context of the calling method.

- **pCalculationType** - The calculation type ctySquare or ctyCalculation.
- **pBuffer** - Address of the calculation in text form.
- **pLen** - The length of the calculation in text form.
- **pError1** - First character of offending text in calculation.
- **pError2** - Last character of offending text in calculation.

See also EXTfldval::getCalculation, EXTfldval::evalCalculation

EXTfldval::setChar()

void EXTfldval::setChar(const strxxx& pString,
qshort pSubDataType = dpDefault)

Stores a string in the EXTfldval object.

- **pString** - The string to be stored.
- **pSubDataType** - The sub data type to convert to. See **dpDefault** above.

EXTfldval::setChar()

void EXTfldval::setChar(qchar* pAddress, qlong pLen)

Stores a string in the EXTfldval object.

- **pAddress** - The address of some data to be stored as a string value.
- **pLen** - The number of bytes to copy from pAddress.

EXTfldval::setConstant()

void EXTfldval::setConstant(preconst pX)

Sets the value of the fldval the specified constant.

- **pX** – The ID of the constant. Please see the source file dmconst.he for valid values.

Example:

```
fval.setConstant(preButtmodeOk);
```

See also EXTfldval::getConstant, EXTfldval::getLong(preconst,preconst,qbool*)

EXTfldval::setConstant()

void EXTfldval::setConstant(preconst pMin, preconst pMax, qlong pVal)

Sets the value of the fldval to the constant ID of the given value. Omnis will find the constant ID, using the range of Ids specified by pMin and pMax.

- **pMin** – The constant range start ID. Please see the source file dmconst.he for valid values.
- **pMax** – The constant range end ID. Please see the source file dmconst.he for valid values.
- **pVal** – The value to search for.

Example:

```
fval.setConstant(preButtmodeF,preButtmodeL,1);
// this will set the fldval to preButtmodeOk
```

See also EXTfldval::getConstant, EXTfldval::getLong(preconst,preconst,qbool*)

EXTfldval::setConstant() (v4.1)

qbool EXTfldval::setConstant(strxxx& pX)

Sets the fldval to a constant value directly from the supplied string.

- **pX** – String containing the constant value.

Example:

```
EXTfldval fldval;
str255 colorStr(QTEXT("kDarkMagenta"));
fldval.setConstant(colorStr);
```

EXTfldval::setCrbRef()

void EXTfldval::setCrbRef(qcrb pCrb, qbool pTransferOwnership)

Stores an Omnis data collection.

- **pCrb** - Points to the data collection to be stored.
- **pTransferOwnership** - If true, the data collection will belong to the EXTfldval, and must NOT be destroyed. If false, Omnis will store a copy of the data.

EXTfldval::setDate()

```
void EXTfldval::setDate ( const
datestamptype& pDateTime, qshort
pSubDataType = dpDefault )
```

Stores a datetime value.

- **pDateTime -** The datetime structure to store.
- **pSubDataType** - Defines what type of datetime is stored.

EXTfldval::setEmpty() (v3.1)

```
void EXTfldval::setEmpty( ffttype fft1, qshort fdp1 )
```

Sets the data to empty and sets it to the specified data type.

- **fft1 -** The data type
- **fdp1** - The sub data type. This depends on the fft1 parameter.

See also EXTfldval::setNull

EXTfldval::setFldVal()

```
void EXTfldval::setFldVal( qfldval pData )
```

Sets the Omnis data pointer in the EXTfldval to the given pointer. Any data belonging to the EXTfldval is destroyed prior to the pointer being changed.

Note: The data is not duplicated, and will not belong to the EXTfldval.

- **qfldval -** The new Omnis data pointer.

See also EXTfldval::getFldVal, EXTfldval::setOmnisData

EXTfldval::setHandle()

```
void EXTfldval::setHandle (ffttype pDataType,
HGLOBAL pHandle, pSubDataType =
dpDefault )
```

Stores data in binary form.

- **pDataType -** The type of data that is assumed to have been stored.
- **pHandle** - The buffer to read the data from and store.
- **pSubDataType** - The sub data type. This depends on the pDataType parameter.

EXTfldval::setHandle()

```
void EXTfldval::setHandle (ffttype pDataType,
qHandle pHandle, qbool pTakeACopy,
pSubDataType = dpDefault )
```

Stores data in binary form.

- **pDataType -** The type of data that is assumed to have been stored.
- **pHandle** - The buffer to read the data from and store.
- **pTakeACopy**- Should Omnis take a copy of the given qHandle
- **pSubDataType** - The sub data type. This depends on the pDataType parameter.

EXTfldval::setList()

```
void EXTfldval::setList( EXTqlist* pList, qbool
pTransferOwnership )
```

Stores a list in the EXTfldval object

- **pList -** The list to store.
- **pTransferOwnership** - qtrue if the EXTfldval should take ownership of the list's contents. If this is qtrue, you should NOT assign **qnil** to the EXTqlist object as it causes a crash. If this parameter is qfalse, the EXTfldval contains a reference to the EXTqlist being passed in, and as such will only be valid while the EXTqlist is valid.

EXTfldval::setList()

```
void EXTfldval::setList( EXTqlist* pList, qbool
pTransferOwnership, qbool pIsRow )
```

Stores a list or row in the EXTfldval object

- **pList -** The list to store.
- **pTransferOwnership** - qtrue if the EXTfldval should take ownership of the list's contents. If this is qtrue, you should NOT assign **qnil** to the EXTqlist object as it causes a crash. If this parameter is qfalse, the EXTfldval contains a reference to the EXTqlist being passed in, and as such will only be valid while the EXTqlist is valid.
- **pIsRow** – if qtrue, the list subtype is set to dpFrow, meaning that only the first row will be stored.

EXTfldval::setLong()

```
void EXTfldval::setLong( qlong pValue )
```

Stores a qlong numeric value.

- **pValue -** The value to store.

EXTfldval::setNull() (v3.1)

void EXTfldval::setNull(ffttype fft1, qshort fdp1=(qshort)dpDefault)

Sets the data to NULL and sets it to the specified data type.

- **fft1 -** The data type
- **fdp1** - The sub data type. This depends on the fft1 parameter.

See also EXTfldval::setEmpty

EXTfldval::setNum()

void EXTfldval::setNum(qreal pNumValue, qshort& pSubDataType = dpDefault)

Stores a number value.

- **pNumValue** - The numeric value to be stored.
- **pSubDataType**- The decimal places to store the number as. dpDefault does not convert.

Example:

```
// sending an event parameter
EXTfldval evParam;
evParam.setLong( 10 );
ECOsendEvent( mHWnd, myEvent, &evParam, 1 );
// converting a number to a string.
EXTfldval general;
str255 s;
general.setLong( 10 );
general.getChar( s );
s.concat( str255(" errors were found" );
// s now contains ’10 errors were found’
```

EXTfldval::setObjInst()

void EXTfldval::setObjInst(qobjinst pObjInst, qbool pTransferOwnership)

Stores an objinst in the EXTfldval object.

- **pObjInst -** The object to store.
- **pTransferOwnership** - qtrue if the EXTfldval should take ownership of the object instance.

EXTfldval::setOmnisData()

void EXTfldval::setOmnisData(qbool pIsOmnisData)

Sets the ownership of the Omnis data pointer in the EXTfldval.

- **pIsOmnisData -** If true, the Omnis data pointer will belong to the EXTfldval, and will be destroyed when the EXTfldval is destroyed.

See also EXTfldval::isOmnisData, EXTfldval::setFldVal, EXTfldval::getFldVal

EXTfldval::setReadOnly()

Internal use only.

Chapter 11—HWND Reference

This chapter describes the public interface of the HWND module, which is the Omnis cross-platform window manager. This chapter includes a description of the Structures, Data types, and Defines required by some HWND functions, Style flags for the Omnis window, the Messages sent to a window's message procedure, and HWND Functions.

The HWND

The HWND is a child window, the graphical container for an Omnis window control. It is split into two areas, the *client area* and the *non-client area*. The non-client area contains the border (there are a number of border styles available) and scrollbars of the window. The client area (the area which remains after subtracting the border and scrollbars) can be used to display the control's interface. The client area can also contain further child windows which are part of the control's interface, or are complete controls in themselves.

The Z-order

The *Z-order* is the order in which windows appear on the screen. When thinking in terms of a chain of sibling windows, or child windows belonging to the same parent window, the Z-order is like a deck of cards. The top most card can always be seen in its entirety, and how much can be seen of all remaining cards, depends on how they are laid out on the table. When thinking in terms of parent and child windows, the Z-order becomes more complex. Parent windows can be thought of as boxes with a rectangular opening in the lid, through which the child windows can be viewed. The size and location of the opening depends on the window's coordinates. How much of a child can be seen through the opening depends on the child's coordinates in relation to that opening. Child windows are always considered to be below their parent window in terms of Z-order, but are considered to be above all sibling windows of their parent if these sibling windows are positioned below that parent (just as if you were to stack a number of parent boxes containing child cards). Changing the parent of a child (see WNDsetParent) alters its position in the Z-order.

When enumerating windows (see WNDenumChildWindows) it is the Z-order which determines the order of the enumerated windows, their child windows, and the children's children, and so on.

When system updates occur, windows are painted starting at the top of the Z-order.

HWND Components

Given that the client area of a window can contain any number of child windows, these child windows normally have a location and size within their parent window which is specified at the time they are created, and altered later on. If the parent window has been given scrollbars, the child window can be moved by scrolling the parent's client area.

Child windows can also be created as specific components in a parent's client area (in this case the parent should have no scrollbars). A component window has a fixed location within its parent, and usually only the width or height of a component can be specified, if at all. When the parent's client area height or width changes, all components resize accordingly.

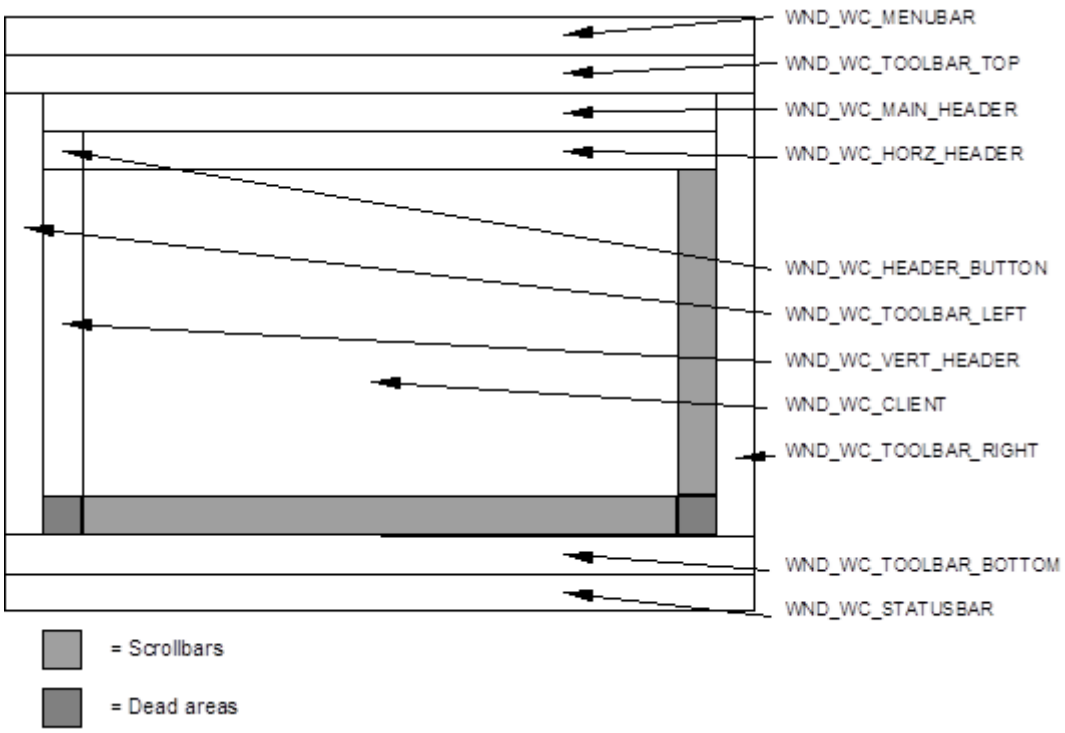

Figure 1:

Any window can contain one of each component type. Any component in turn can contain a further full set of components. There is no specific limit to the number of nested windows or component windows. The following diagram shows all component types in their correct position.

In this diagram only the client component is displayed with scrollbars, but any other component could have scrollbars, if appropriate.

Note: The names of the components bear no relationship to objects generally described by these names; a component's name gives you an idea of its position in the window. For example, Menubar tells you that the component is at the top of the window where you would expect to find a menu bar.

The following is a listing of all the components and their special function.

Name	Special function	Sizeability
WND_WC_FRAME	This is the default component ID of a window. A frame window has no special functionality.	all
WND_WC_MENUBAR	Width is dependent on parent's width	height
WND_WC_TOOLBAR_TOP	Width is dependent on parent's width	height
WND_WC_TOOLBAR_LEFT	Height is dependent on parent's height minus the height of MENUBAR, TOOLBAR_TOP, TOOLBAR_BOTTOM and STATUSBAR components	width
WND_WC_TOOLBAR_RIGHT	Height is dependent on parent's height minus the height of the MENUBAR, TOOLBAR_TOP, TOOLBAR_BOTTOM and STATUSBAR components	width
WND_WC_TOOLBAR_BOTTOM	Width is dependent on parent's width	height
WND_WC_STATUSBAR	Width is dependent on parent's width	height
WND_WC_MAIN_HEADER	Width is dependent on parent's width minus the width of the left and right toolbar components	height
WND_WC_HORZ_HEADER	Width is dependent on parent's width minus the width of the left toolbar, right toolbar, and vertical header components. This component's horizontal scroll range and position is linked to that of the client component's horizontal scroll range and position. When the client component is scrolled horizontally, this component receives a duplicate of all scroll messages.	height

Name	Special function	Sizeability
WND_WC_VERT_HEADER	Height is dependent on parent's height minus the height of the MENUBAR, TOOLBAR_TOP, MAIN_HEADER, HORZ_HEADER, TOOLBAR_BOTTOM and STATUSBAR components.This component's vertical scroll range and position is linked to that of the client component's vertical scroll range and position. When the client component is scrolled vertically, this component receives a duplicate of all scroll messages.	width
WND_WC_HEADER_BUTTON	Height and width are dependent on the horizontal and vertical headers' height and width.	none
WND_WC_CLIENT	Height and width are dependent on the remainder of the parent's client area after subtracting all other components.	none

Structures, Data types, and Defines

GW_xxx

These flags are used with the function WNDgetWindow:

- GW_**CHILD**
  Identifies the window's first child window.
- GW_**HWNDFIRST**
  Returns the first sibling window for a child window.
- GW_**HWNDLAST**
  Returns the last sibling window for a child window.
- GW_**HWNDNEXT**
  Returns the sibling window that follows the given window in the window manager's list.
- GW_**HWNDPREV**
  Returns the previous sibling window in the window manager's list.

GWL_xxx

These flags are used with the functions WNDgetWindowLong and WNDsetWindowLong:

- **GWL_STYLE**
  Returns the window's basic window style.
- **GWL_EXSTYLE**
  Returns the window's extended window styles. (Omnis additional window styles)
- **GWL_EXCOMPONENTID**
  Returns the window's component id (one of the WND_WC_xxx styles). This flag **cannot** be used with WNDsetWindowLong.
- **GWL_BKTHEME**
  Stores the window theme background. See WNDdrawThemeBackground for full details. Setting this value will invalidate the HWND area.
- **GWL_BKTHEME_NOINVAL**
  Same as GWL_BKTHEME, but does not invalidate the HWND area when setting this value.
- **GWL_INFLATE_ALL** (Mac OSX only)
  Allows you to set an area around the HWNDs visual area for drawing by this HWND. In other words during painting to this HWND, you may paint outside the HWNDS bounding area. This is useful if you need to paint drop shadows around your control. To specify the inflate values you can set-up a qrect and use the function WNDmakeLong to convert the qrect to a long value.

- **Example:**

```
// inflate the paint area on the left and right by 2 pixels, and 4
// pixels to the bottom
qrect inflateRect( -2, 0, 2, 4 );
WNDsetWindowLong(theHwnd, GWL_INFLATE_ALL, WNDmakeLong(&inflateRect));
```

- **GWL_INFLATE_FRAME** (Mac OSX only)
  Same as GWL_INFLATE_ALL but only effects the non-client painting.

HDC

The HDC is a graphical device context and is fully documented in the *GDI Reference* chapter.

HTxxx

These defines are used by some mouse related message, for example, WM_SETCURSOR, to specify the part of a window, the mouse is currently over.

- **HTNOWHERE**
  The mouse is not over the window.
- **HTCLIENT**
  The mouse is over the client area.
- **HTHSCROLL**
  The mouse is over the horizontal scroll bar.
- **HTVSCROLL**
  The mouse is over the vertical scroll bar.
- **HTGROWBOX**
  The mouse is over the grow box (WIN95 and MacOS only).
- **HTBORDER**
  The mouse is over the border of the window.

HWND

A handle or reference to a child window.

HWND_xxx

You can use the following defines instead of a valid HWND with some of the functions in the API:

- **HWND_DESKTOP**
  Refers to the desktop window. HWND_DESKTOP can be used with various functions including WNDstartDraw and WNDend-Draw, if unclipped drawing to anywhere on the screen is required. Under MacOS it is the sum of all connected monitors.
- **HWND_MAINWINDOW**
  Under Windows it refers to the Omnis Program window. Under MacOS it is the main monitor (the one with the menu bar).
- **HWND_TOP**
  Can be used with WNDsetWindowPos to move the window to the top of the z-order (Top of its sibling chain).
- **HWND_BOTTOM**
  Can be used with WNDsetWindowPos to move the window to the bottom of the z-order (Bottom of its sibling chain).

LPARAM

LPARAM is a typedef and is of type unsigned long. The lParam and uParam parameters of the WndProc function are of this type.

SW_xxx

These flags are used with the function WNDshowWindow:

- **SW_HIDE**
  Hides the window.
- **SW_SHOW**
  Shows the window.

SWP_xxx

These flags are used with the functions WNDsetWindowPos and WNDsetWindowPosEx:

- **SWP_NOSIZE**
  If specified no sizing of the window takes place.
- **SWP_NOMOVE**
  If specified no moving of the window takes place.
- **SWP_NOZORDER**
  If specified the position in the z-order of the window is not altered.
- **SWP_NOREDRAW**
  If specified no invalidating takes place. Any changes to the visibility, z-order or position and size is not reflected on screen.
- **SWP_SHOWWINDOW**
  If specified the window is made visible.
- **SWP_HIDEWINDOW**
  If specified the window is hidden.

UINT

The UINT is defined as an unsigned integer. The message parameter of the WndProc function is of this type.

WM_EXUSER

WM_EXUSER is the base define for all user defined messages for the external components. External components which use the HWND module can create their own message by defining a WM_your_constant as WM_EXUSER + n, where n can be in the range HEX 0 to HEX 1FFF.

Example:

```
#define WM_MY_MESSAGE1 WM_EXUSER+1
#define WM_MY_MESSAGE2 WM_EXUSER+2
```

WND_CAPTURE_xxx

These flags are used with the functions WNDsetCapture and WNDreleaseCapture to specify the capture for mouse or key events:

- **WND_CAPTURE_KEY**
  Captures all keyboard events. It is not necessary for external components to capture the key events. Omnis sets the key capture for a window when it receives the input focus.
- **WND_CAPTURE_MOUSE**
  Captures all mouse events.

WND_BORD_xxx

These are the flags for the various border styles of a window. They are used to set the *mBorderStyle* member of the WNDborderStruct. WNDborderStruct is used with the functions WNDcreateWindow, WNDgetBorderSpec, WNDsetBorderSpec, WNDinsetBorderRect, WNDinflateBorderRect, WNDaddWindowComponent and WNDpaintBorder.

- **WND_BORD_NONE**
  No border.
- **WND_BORD_PLAIN**
  Draws a plain border using the qpen specified by the *mLineStyle* member of the WNDborderStruct. For a complete description of a qpen refer to the GDI documentation.
- **WND_BORD_INSET**
  Draws a 3D inset border. Standard 3D system colors are used to draw the effect.
- **WND_BORD_EMBOSSED**
  Draws a 3D embossed border. Standard 3D system colors are used to draw the effect.
- **WND_BORD_BEVEL**
  Combination of the embossed and inset border styles, with the inset frame being drawn inside the embossed frame and a flat area in between. The *mSize1*, *mSize2* and *mSize3* members of the WNDborderStruct specify the sizes of the three bevel parts (embossed, flat and inset). Standard 3D system colors are used to draw the effect.
- **WND_BORD_INSETBEVEL**
  Same as WND_BORD_BEVEL, except that the three bevel parts are reversed, making the bevel appear inset.
- **WND_BORD_CHISEL**
  Draws a two pixel wide chiseled border. Standard 3D system colors are used to draw the effect.
- **WND_BORD_EMBOSSEDCHISEL**
  Same as WND_BORD_CHISEL, except that it appears embossed.
- **WND_BORD_SHADOW**
  Gives client area the appearance of having a shadow. The two members *mSize1* and *mSize2* are used to specify the horizontal and vertical shadow size (offset) and the *mColor* specifies the shadows color. *mLineStyle* is used to give the client area an additional simple frame. A shadow border has two areas called dead area. These are areas that are not covered by the border effect itself, but nevertheless need to be erased. The HWND module queries the erase colors by sending a WM_GETERASEINFO message to the WndProc function.
- **WND_BORD_SINGLE_INSET**
  Draws a 3D single pixel width inset border. Standard 3D system colors are used to draw the effect.
- **WND_BORD_SINGLE_EMBOSSED**
  Draws a 3D single pixel width embossed border. Standard 3D system colors are used to draw the effect.
- **WND_BORD_3DFACE**
  Same as WND_BORD_INSET, but uses 3DFACE color and no black.
- **WND_BORD_3DHILITE**
  Same as WND_BORD_INSET, but uses 3DHILITE color and no black.
- **WND_BORD_CTRL_EDIT (v3.1)**
  Draws the correct border for a edit control. Platform dependent.
- **WND_BORD_CTRL_LIST (v3.1)**
  Draws the correct border for a list control. Platform dependent.
- **WND_BORD_CTRL_LISTCELL (v3.1)**
  Draws the correct border for a list cell control. Platform dependent.
- **WND_BORD_CTRL_TABPANE (v3.1)**
  Draws the correct border for a tab pane control. Platform dependent. Generates a WM_GETSHADOWRECT message to allow caller to manipulate the border rect prior to drawing.

- **WND_BORD_CTRL_SHADOW (v3.1)**
  Draws a system shadow border. Platform dependent.

- **WND_BORD_CTRL_SHADOW_EX (v3.1)**
  Same as WND_BORD_CTRL_SHADOW, but generates a WM_GETSHADOWRECT message to allow caller to manipulate the border rect prior to drawing.

- **WND_BORD_CUSTOM**
  When specified, custom borders can be drawn in the windows non-client area (frame). The messages WM_BORDCALCRECT and WM_BORDPAINT are send to the WndProc function when the non-client area needs to be calculated or the border needs painting.

WND_CURS_xxx

These flags are used with the WNDsetWindowCursor and WNDgetWindowCursor functions to specify the cursor type associated with a window, and the functions WNDsetCursor and WNDgetCursor, to instantly change the cursor on screen.

- **WND_CURS_DEFAULT**
  Cursor does not change when moving over the window. Control over the cursor is passed to the parent window.

- **WND_CURS_ARROW**
  Standard cursor.

- **WND_CURS_IBEAM**
  Standard edit text cursor.

- **WND_CURS_WATCH**
  Standard time/watch cursor.

- **WND_CURS_LOCK**
  Record locked cursor.

- **WND_CURS_MOVE**
  Cursor for moving objects.

- **WND_CURS_SIZE_VERT**
  Cursor for sizing object vertically only. (Center top/bottom size knobs)

- **WND_CURS_SIZE_HORZ**
  Cursor for sizing objects horizontally only. (Center left/right size knobs)

- **WND_CURS_SIZE_LTRB**
  Cursor for sizing objects diagonally left.top to right.bottom. (Left.Top and Right.Bottom size knobs)

- **WND_CURS_SIZE_LBRT**
  Cursor for sizing objects diagonally left.bottom to right.top. (Left.Bottom and Right.Top size knobs)

- **WND_CURS_INSERT**
  Cursor for inserting rows between rows etc.

- **WND_CURS_COPY_SINGLE**
  Cursor for copying a single object or data item.

- **WND_CURS_COPY_MULTI**
  Cursor for copying multiple objects or data items.

- **WND_CURS_DRAG_OBJECT**
  Cursor for dragging objects.

- **WND_CURS_DRAG_DATA**
  Cursor for dragging data.

- **WND_CURS_SPLITTER_VERT**
  Cursor for moving vertical splitter bars.

- **WND_CURS_SPLITTER_HORZ**
  Cursor for moving horizontal splitter bars.
- **WND_CURS_NOGO**
  Nogo cursor used for letting user know that you cannot put something here. (Drag or copy)
- **WND_CURS_HELP**
  Help cursor, when used to click on an objects, should bring up context sensitive help.
- **WND_CURS_EXAMINE**
  Examine cursor used for expanding data etc.
- **WND_CURS_TRASH**
  Trash cursor.
- **WND_CURS_ARROW_WATCH**
  Cursor displaying an arrow and watch.
- **WND_CURS_CROSS**
  Area selection tool.
- **WND_CURS_DROPPER**
  Color suction tool.
- **WND_CURS_BUCKET**
  Area fill tool.
- **WND_CURS_PENCIL**
  Drawing tool.

WND_RW_xxx

Used with WNDredrawWindow to redraw/invalidate or update the non-client and/or client area of a window:

- **WND_RW_NCPAINT**
  Redraws all of the non-client area.
- **WND_RW_PAINT**
  Redraws the specified area within the client area of the window.
- **WND_RW_ERASE**
  If specified, erase background messages are generated.
- **WND_RW_ALLCHILDREN**
  If specified, all children are included in the redraw operation.
- **WND_RW_INVALIDATE**
  If specified, the specified area is invalidated only, and repainted during the normal update process.
- **WND_RW_UPDATE**
  If specified, any invalid area of the window is immediately updated. The qrgn and qrect parameters are ignored. Only the WND_RW_ALLCHILDREN and WND_RW_ERASE flags can be specified in combination with this flag.

WND_SCROLLBAR_WIDTH (v3.1)

Returns the width in pixels of a standard scrollbar.

WND_TIMER_xxx

These constants are used with the WNDsetTimer and WNDkillTimer functions:

- **WND_TIMER_NULL**
  Internal use only.
- **WND_TIMER_TOOLTIP**
  Internal use only.
- **WND_TIMER_FIRST**
  Base constant for all timer ids which are used by external components.

WNDborderStruct

The border struct contains information for the border style of a window. It is used with the functions WNDcreateWindow, WNDaddWindowComponent, WNDsetBorderSpec, and WNDgetBorderSpec.

```
struct WNDborderStruct
{
  qshort mBorderStyle;
  qpen mLineStyle;
  qdim mSize1;
  qdim mSize2;
  qdim mSize3;
  qcol mColor;
  WNDborderStruct();
  WNDborderStruct( qshort pBorderStyle );
  WNDborderStruct( qshort pBorderStyle, qpen pLineStyle );
  WNDborderStruct( qshort pBorderStyle, qdim pSize1, qdim pSize2, qdim pSize3 );
  WNDborderStruct( qshort pBorderStyle, qpen pLineStyle, qdim pSize1, qdim pSize2, qcol pColor );
};
```

- **mBorderStyle** specifies one of the WND_BORD_xxx constants.
- **mLineStyle** is used by WND_BORD_PLAIN border styles.
- **mSize1** is used by WND_BORD_BEVEL, WND_BORD_INSETBEVEL and WND_BORD_SHADOW.
- **mSize2** is used by WND_BORD_BEVEL, WND_BORD_INSETBEVEL and WND_BORD_SHADOW.
- **mSize3** is used by WND_BORD_BEVEL and WND_BORD_INSETBEVEL.
- **mColor** is used by WND_BORD_SHADOW.

WNDenumProc

typedef qbool (*WNDenumProc)(HWND hwnd, LPARAM lParam);

WNDeraseInfoStruct

```
struct WNDeraseInfoStruct
{
  qcol   mBackColor;
  qcol   mForeColor;
  qcol   mFillPat;
  qulong mBKTheme;
}
```

- **mBackColor** - specifies the color for all clear pixels in the fill pattern.

- **mForeColor** - specifies the color for all set pixels in the fill pattern.
- **mFillPat** - specifies the fill pattern.
- **mBKTheme** – specifies the background theme (see GWL_BKTHEME)

WNDminMaxInfo

```
struct WNDminMaxInfo
{
  qpoint  ptReserved;
  qpoint  ptMaxSize;
  qpoint  ptMaxPosition;
  qpoint  ptMinTrackSize;
  qpoint  ptMaxTrackSize;
};
```

- **ptReserved** - Reserved for internal use.
- **ptMaxSize** - Currently NOT used for child windows.
- **ptMaxPosition** - Currently NOT used for child windows.
- **ptMinTrackSize** - Specifies the minimum tracking width (point.h) and the minimum tracking height (point.v) of the window.
- **ptMaxTrackSize** - Specifies the maximum tracking width (point.h) and the maximum tracking height (point.v) of the window.

WNDmultiKey (v4.1)

WNDmultiKey is a class used to pass multiple keypress information via event parameters and is used in conjunction with WM_MULTIKEYxxx events. WNDmultiKey is defined in hwnd.he and contains the following public members:

- **WNDmultiKey()** – Default constructor.
- **WNDmultiKey(qchar *pData, qlong pLen)** – Initializes the class using the supplied keys (one per qchar position). The number of keys is specified via pLen.
- **WNDmultiKey(WNDmultiKey &pMultiKey)** – Initializes the class copying data from an existing WNDmultiKey instance.
- **~WNDmultiKey()** – Default destructor.
- **void set(qchar *pData, qlong pLen)** – Assigns the key-combination held in pData to the class. pLen contains the number of keys being pressed.
- **qlong len()** – Returns the number of key presses stored by the class.
- **qchar *dataPtr()** – Returns a pointer to the keys stored by the class.

WNDpaintStruct

```
struct WNDpaintStruct
{
  HDC     hdc;
  qbool  fErase;
  qrect  rcPaint;
  HDC    fRestore;
};
```

- **hdc** - Identifies the display context to be used for painting.
- **fErase** - Specifies whether the background needs to be redrawn.
- **rcPaint** - Specifies the upper-left and lower-right corners of the rectangle in which the painting is requested.
- **fRestore** - Reserved; internal use.

WNDprocClass

```
class WNDprocClass
{
  public:
  virtual qlong WndProc(HWND hWnd, UINT message, WPARAM wParam,LPARAM lParam, LPARAM uParam ) = 0;
};
```

WNDwindowPosStruct

```
struct WNDwindowPosStruct
{
  HWND   hwnd;
  HWND   hwndInsertAfter;
  qdim   x;
  qdim   y;
  qdim   cx;
  qdim   cy;
  qulong flags;
};
```

- **hwnd** - Identifies the window.
- **hwndInsertAfter** - Identifies the window behind which this window is placed.
- **x** - Specifies the position of the left edge of the window.
- **y** - Specifies the position of the right edge of the window.
- **cx** - Specifies the window width.
- **cy** - Specifies the window height.
- **flags** - Specifies window positioning options. This member is one or more of the SWP_xxx flags.

WPARAM

Styles

WND_DRAGBORDER (extended style)

If this extended style is specified for a component other than WND_WC_FRAME and WND_WC_CLIENT, the user can size the window at runtime. When the cursor moves over the correct border edge (which edge can be dragged depends on the component type) the cursor changes to WND_CURS_SPLITTER_VERT or WND_CURS_SPLITTER_HORZ. Sizing a component effects the size of other sibling components.

WND_FLOAT_xxx (extended style)

These are the floating styles for a window. Floating takes place if the parent of a floating window is sized. The window is sized or moved, horizontally or vertically by the same amount the parent has grown or shrunk by. The floating style can only be specified for windows of type WND_WC_FRAME. The following styles are defined:

- **WND_FLOAT_NONE**
  No floating.
- **WND_FLOAT_LEFT**
  If set, the window grows or shrinks horizontally by floating the left edge of the window.
- **WND_FLOAT_RIGHT**
  If set, the window grows or shrinks horizontally by floating the right edge of the window.

- **WND_FLOAT_LEFT_RIGHT**
  If set, the window moves horizontally. (WND_FLOAT_LEFT | WND_FLOAT_RIGHT)
- **WND_FLOAT_TOP**
  If set, the window grows or shrinks vertically by floating the top edge of the window.
- **WND_FLOAT_BOTTOM**
  If set, the window grows or shrinks vertically by floating the bottom edge of the window.
- **WND_FLOAT_TOP_BOTTOM**
  If set, the window moves vertically.
- **WND_FLOAT_MASK**
  Masking bits for masking out floating styles in the extended style window long.

WND_KEYPREVIEW (extended style)

If specified, WM_KEYDOWNPREVIEW and WM_KEYUPPREVIEW messages are generated and sent to all parents of the window for which the actual key message was intended. The parameters are identical to WM_KEYDOWN and WM_KEYUP. If a parent deals with a key and returns 0, no further messages are sent relating to the key.

WND_NOADJUSTCOMPONENTS (extended style)

If this style is specified, the component windows of the window are not sized when the window's size changes. The window component type of all child windows is ignored.

WND_NOFLOATCHILDREN (extended style)

If this style is specified, no child windows of the window are floated when the window size changes. All floating styles of all child windows are ignored.

WND_OSMESSAGES (extended style)

If this style is specified, the window can receive additional non-cross-platform messages, messages which are not normally supported by the hwnd module. Under Windows 95 platform this may be messages like WM_DROPFILES, etc. This style is currently *not* supported under MacOS.

WND_REDRAWONSIZE (extended style)

If this style is specified, all of the client area of the window is invalidated when the width or height of the window changes. By default only the uncovered areas are invalidated.

WND_TRANSPARENT (extended style)

If this style is specified, the window becomes transparent. Transparent windows do not receive erase background messages and are painted last (after all non-transparent windows have been painted) and in reverse order. They do not clip the visual region of sibling windows which they cover, nor do they clip their parent. If areas within a transparent window are invalidated, all intersecting sibling windows are effected as well as the area within the parent.

Note: Transparent windows are less efficient, and may not always yield the desired results. Omnis uses transparent windows only for background objects in window and report classes.

WND_WC_xxx (extended style)

The WND_WC_xxx styles are used with the functions WNDaddWindowComponent, WNDremoveWindowComponent, WNDgetWindowComponent, and WNDnextWindowComponent. These flags specify the component type of a window.

- **WND_WC_FRAME**
  the default type for all windows created by WNDcreateWindow. It is ignored by the component functions.
- **WND_WC_MENUBAR**
  the menu bar component. A window of this type is always positioned in the topmost area of the parent window.
- **WND_WC_TOOLBAR_TOP**
  the top toolbar component. A window of this type is always positioned in the topmost area of the parent window just below the menu bar component.
- **WND_WC_TOOLBAR_BOTTOM**
  the bottom toolbar component. A window of this type is always positioned in the bottom-most area of the parent window just above the status bar component.
- **WND_WC_TOOLBAR_LEFT**
  the left toolbar component. A window of this type is always positioned in the leftmost area of the parent window.
- **WND_WC_TOOLBAR_RIGHT**
  the right toolbar component. A window of this type is always positioned in the rightmost area of the parent window.
- **WND_WC_HEADER_BUTTON**
  can be created in conjunction with the WND_WC_HORZ_HEADER and WND_WC_VERT_HEADER components both of which must exist. Its position and size depends entirely on these two components being positioned in the top-left corner between the two header components.
- **WND_WC_MAIN_HEADER**
  the main header component. A window of this type is always positioned in the topmost area of the parent window just below the top toolbar.
- **WND_WC_HORZ_HEADER**
  horizontally scrolling header component which works in conjunction with the WND_WC_CLIENT component. It sits just above the client component, and scrolls horizontally in the same direction by the same amount, when the client component is scrolled horizontally.
- **WND_WC_VERT_HEADER**
  vertically scrolling header component which works in conjunction with the WND_WC_CLIENT component. It sits just to the left of the client component, and scrolls vertically in the same direction by the same amount, when the client component is scrolled vertically.
- **WND_WC_CLIENT**
  the client component. It is positioned to fill in any space not used by any of the other components within the same parent window. If components are used, this is the component which should receive the scrollbars, if scrollbars are required.
- **WND_WC_STATUSBAR**
  the status bar component. A window of this type is always positioned in the bottom-most area of the parent window.
- **WND_WC_MASK**
  define which can be used to specify all component types.

WS_xxx

These are the windows basic styles.

- **WS_CHILD**
  Must always be specified.

- **WS_CLIPSIBLINGS**
  Clips child windows relative to each other; that is, when a particular child window receives a WM_PAINT message, this style clips all other top-level child windows out of the region of the child window to be updated. (If the WS_CLIPSIBLINGS style is not given and child windows overlap, it is possible, when drawing in the client area of a child window, to draw in the client area of a neighboring child window.)

- **WS_CLIPCHILDREN**
  Excludes the area occupied by child windows when drawing within the parent window. Used when creating the parent window.

- **WS_DISABLED**
  If specified, the window is disabled and receives no mouse events. All mouse events which would normally be received by this window, are sent to the parent window.

- **WS_HSCROLL** and **WS_VSCROLL**
  Creates a window that has a horizontal or vertical scroll bar.

- **WS_VISIBLE**
  Creates a window that is initially visible.

Messages

When users interact with a window, or a window's properties change, appropriate messages are generated. These messages can be received by sub-classing the **WNDprocClass** and overloading the virtual function **WndProc**. An instance of that class must be specified when creating a window. More than one window can be associated with the same instance in this way. The correct HWND is sent to the WndProc function along with its message.

Example of a WndProc function:

```
qlong MyWndProc::WndProc( HWND hWnd, UINT message, WPARAM wParam, lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_GETMINMAXINFO:
    {
      // calculate the minimum tracking size
      WNDminMaxInfo* info = (WNDminMaxInfo*)lParam;
      // first get minimum tracking size for child windows from HWND
      WNDgetMinMaxInfo( hWnd, info );
      if ( info->ptMinTrackSize.h < 300 )
      {
        // do not allow size less than 300 pixels horizontally
        info->ptMinTrackSize.h = 300;
      }
      return 0L;
    }
    case WM_WINDOWPOSCHANGED:
    {
      // reset the scroll range
      qdim hRange = 200;
      qdim vRange = 400;
      qrect cRect;
      WNDgetClientRect( hWnd, &cRect );
      WNDsetScrollRange( hWnd, SB_HORZ, 1, hRange, cRect.width(), qtrue );
      WNDsetScrollRange( hWnd, SB_VERT, 1, vRange, cRect.height(), qtrue );
      break;
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

Note: All message examples are assumed to have the following pieces of code surrounding them:

```
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_xxx:
    {
      // example
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

WM_BORDCALCRECT

The WM_BORDCALCRECT message is sent when the client area of window needs to be calculated and the border style of the window is WND_BORD_CUSTOM.

Parameters:

- **inflate** - Boolean value of wParam. If qtrue, the supplied rectangle is to be inflated, otherwise it is to be inset.
- **rect** - Value of lParam. Points to the qrect to be inflated or inset.

Returns:

Always return 1.

The amount by which the rectangle is inflated or inset depends on how much space the custom border requires. It directly effects the size of the client area of the window.

Example:

```
// this example custom border has a single pixel line at the top and bottom
qrect* pRect = (qrect*)lParam;
if ( wParam )
{
  // inflate
  pRect->top++;
  pRect->bottom++;
}
else
{
  // inset
  pRect->top--;
  pRect->bottom--;
}
return 1L;
```

WM_BORDERCHANGED (v3.1)

The WM_BORDERCHANGED message is send when WNDsetBorderSpec is called, and after the border has been changed in the HWND.

Parameters:

- **borderSpec** - Value of lParam. Points to the border spec structure.

Returns:

Always return 1L.

See also WM_BORDERCHANGING, WNDsetBorderSpec

WM_BORDERCHANGING (v3.1)

The WM_BORDERCHANGING message is send when WNDsetBorderSpec is called, prior to the border being changed.

Parameters:

- **redraw** - Value of wParam. If 1, the caller called WNDsetBorderSpec with the redraw flag set.
- **borderSpec** - Value of lParam. Points to the border spec structure.

Returns:

Return 1 if WNDsetBorderSpec is to go ahead. Return 0 to prevent the border from changing.

Example:

```
// this example makes a copy of the border spec and prevents WNDsetBorderSpec from changing
// the border in the HWND (the control draws and manages the border it self)
qbool redraw = (qbool)wParam;
WNDborderStruct* borderSpec = (WNDborderStruct*)lParam;
mBorderSpec = *borderSpec;
if ( redraw )
{
  // invalidate our control
}
```

See also WM_BORDERCHANGED, WNDsetBorderSpec

WM_BORDERASEBACKGROUND (v4.0)

The WM_BORDERERASEBACKGROUND message is sent when painting a WND_BORD_CTRL_GROUPBOX on Windows XP. The message informs the control that two 3-pixel strips immediately above and below the control should be erased.

- **dc** - Value of wParam. Points to the device in which the border is to be painted.
- **rect** - Value of lParam. Points to the qrect which forms the outside edge of the border rect. The coordinates are local to the device.

Example: (excerpt from WndProc)

```
//..
case WM_BORDERASEBACKGROUND:
{
  if (isSetup())
    eraseBorderBackground(hWnd, (HDC) wParam, (qrect *) lParam);
  return 1L;
}
```

WM_BORDPAINT

The WM_BORDPAINT message is sent when the border of a window needs painting and the border style of the window is WND_BORD_CUSTOM.

- **dc** - Value of wParam. Points to the device in which the border is to be painted.
- **rect** - Value of lParam. Points to the qrect which forms the outside edge of the border rect. The coordinates are local to the device.

Returns:

Always return 1.

Example:

```
// this example custom border has a single pixel line at the top and bottom
HDC dc = (HDC)wParam;
qrect* pRect = (qrect*)lParam;
HPEN oldPen = GDIselectObject( dc, GDIgetStockPen( BLACK_PEN ) );
GDIsetTextColor( GDI_COLOR_QBLACK );
GDImoveTo( dc, pRect->left, pRect->top );
GDIlineTo( dc, pRect->right, pRect->top );
GDImoveTo( dc, pRect->left, pRect->bottom );
GDIlineTo( dc, pRect->right, pRect->bottom );
GDIselectObject( dc, oldPen );
return 1L;
```

WM_CAPTUREABORT

This message is sent by WNDabortMouseCapture() before it releases capture of the mouse pointer (WNDreleaseCapture()). Return type is void.

Parameters: None.

Example:

```
//will be followed immediately by WM_CAPTURECHANGED
case WM_CAPTUREABORT:
{
  tqfFishEyeObject *object = (tqfFishEyeObject*) ECOfindObject(eci, hwnd);
  if (object)
  {
    object->trackingEnabled(qtrue);
    object->mDestroyOnCaptureChange = qtrue;
  }
  break;
}
```

WM_CHILDPAINT

The WM_CHILDPAINT message is sent for every child window when a window requests its children to be painted by calling the WNDredrawChildren function.

- **hWnd** - is the window handle of the child window.
- **flags** - value of lParam. This is WND_RW_NCPAINT or WND_RW_NCPAINT and WND_RW_PAINT. If WND_RW_PAINT is specified, the client area needs painting. If WND_RW_NCPAINT is specified the non-client area needs painting.

Returns:

An external component should return zero if it processes this message. If non-zero is returned, a WM_PAINT message is sent to the child window.

Example:

See WNDredrawChildren.

WM_COREPATTERNGRADIENTSUPPORT (v5.0)

This message returns qtrue if the object wants gradients to be shown in the pattern palette in the property inspector, i.e. for $backpattern. Return qfalse to disable gradient back patterns.

Parameters: None.

WM_CREATE

The WM_CREATE message is sent when an external component requests that a window be created by calling the WNDcreateWindow or WNDaddWindowComponent function. The WndProc function for the new window receives this message after the window is created but before the window becomes visible. The message is sent to the window before the WNDcreateWindow or WNDaddWindowComponent function returns.

Parameters:

None.

Returns:

Must always return 0.

WM_DESTROY

The WM_DESTROY message is sent when a window is being destroyed. It is sent to the WndProc function of the window being destroyed after the window is removed from the screen.

This message is sent first to the window being destroyed and then to the child windows as they are destroyed. During the processing of the WM_DESTROY message, you can assume that all child windows still exist.

Parameters:

None.

Returns:

Must always return 0.

WM_DRAGBORDER

WM_DRAGBORDER is sent during border dragging every time the size of the component is changed, after all children and sibling components have been sized.

Parameters:

isVert - value of wParam. If qtrue we are dragging a vertical border.

Returns:

Must always return 0.

WM_DRAGDROP

WM_DRAGDROP message is sent during drag & drop operations. Care should be taken not to process these messages during design mode. Most of these messages can be ignored and simply passed into WNDdefWindowProc or returned with a status of -1 for WebClient.

Parameters:

- **lParam** - Value of lParam depends on wDragDropCode, refer to the individual messages.
- **wDragDropCode** - Value of wParam. Specifies a drag drop code that indicates the request. This parameter can be one of the following values:

DD_STARTDRAG
Indicates that the drag process is starting. Normally this message is ignored. LParam will contain a pointer to the FLDdragDrop structure.

DD_ENDDRAG
Indicates that the drag process is finishing. Normally this message is ignored. LParam will contain a pointer to the FLDdragDrop structure.

DD_CHILD_STARTDRAG
Indicates that the drag process is starting. Sent to the parent of the dragging window. LParam will contain a pointer to the FLDdrag-Drop structure.

DD_CHILD_ENDDRAG
Indicates that the drag process is finishing. Sent to the parent of the dragging window. LParam will contain a pointer to the FLDdrag-Drop structure.

DD_CANDRAG_ON_DOWN
Enquiry on whether dragging can be started by a mouse button down action. Return true or false, or simply ignore the message. LParam will contain a pointer to a qpoint structure which will contain the mouse position. The point is local to the client area of the window which receives these messages.

DD_CANDRAG_ON_MOVE
Enquiry on whether dragging can be started by a mouse move action. Return true or false, or simply ignore the message. LParam will contain a pointer to a qpoint structure which will contain the mouse position. The point is local to the client area of the window which receives these messages.

DD_CANDROP
Sent to the drop control and it can return qtrue if drop action is allowed. LParam will contain a pointer to the FLDdragDrop structure and member mDropPoint may be used to establish drop position.

DD_CANDROP_OVER
Sent to the drop control and it can return qtrue if dropping is allowed. LParam will contain a pointer to the FLDdragDrop structure and member mDropPoint may be used to establish mouse position.

DD_CANDROPPARENT
Sent to the parent of the drop control and it can return qtrue if dropping is allowed. LParam will contain a pointer to the FLDdragDrop structure and member mDropPoint may be used to establish mouse position.

DD_HILITE
Request to the current dropping control to hilite its acceptance to allow dropping. LParam will contain a pointer to the FLDdragDrop structure.

DD_UNHILITE
Request to the current dropping control to unhilite itself. LParam will contain a pointer to the FLDdragDrop structure.

DD_ALWAYS_HILITE
Request to the current dropping control to establish whether highlighting is required. Return qtrue or qfalse. LParam will contain a pointer to the FLDdragDrop structure

DD_SHOWDRAGSHAPE
Message to show the drag shape. Normally this is ignored. LParam will contain a pointer to the FLDdragDrop structure.

DD_HIDEDRAGSHAPE
Message to hide the drag shape. Normally this is ignored. LParam will contain a pointer to the FLDdragDrop structure.

DD_MOVEDRAGSHAPE
Message to move the drag shape. Normally this is ignored. LParam will contain a pointer to the FLDdragDrop structure.

DD_CANSCROLL
Request to the current dropping control to establish whether scroll is required. Return qtrue or qfalse. If qtrue is returned then DD_DRAGDROPSCROLL will be sent. LParam will contain a pointer to the FLDdragDrop structure.

DD_GETSCROLLRECT
Request to the current dropping control for it to adjust the scrolling rectangle, if required. Return qtrue if processed. lParam will contain a pointer to the qrect which can be adjusted.

DD_DRAGDROPSCROLL
Request to the current dropping control for it to scroll, if required. Return qtrue if processed. lParam will contain a pointer to the qpoint which can be used to ensure that the point is inside the control.

DD_SETDRAGVALUE
Request for control to set the drag value and can be used, for example, to set the drag value to a selection of text. LParam will contain a pointer to the FLDdragDrop structure.

DD_GETDRAGCONTAINER
Request for control to set the drag source HWND (FLDdragDrop member mDragSourceHwnd). Normally this is ignored but can be useful for complex controls that allow dragging of multiple HWNDs. LParam will contain a pointer to the FLDdragDrop structure.

DD_BUTTONDOWN
Message that the button is down during drag move. Normally this is ignored but it can be used to change drop tabs on a tabbed pane control, for example. LParam will contain a pointer to the FLDdragDrop structure.

DD_BUTTONUP
Message that the button is up during drag move. Normally this is ignored but it can be used to change drop tabs on a tabbed pane control, for example. LParam will contain a pointer to the FLDdragDrop structure.

Returns:

Depends on the value of wDragDropCode, but most of these messages can be ignored and simply passed into WNDdefWindowProc or returned with a status of -1 for WebClient.

Example:

```
// #1: this example is taken from the list control
case WM_DRAGDROP:
{
  switch ( wParam )
  {
    case DD_CANDRAG_ON_DOWN: return qfalse;
    case DD_CANDRAG_ON_MOVE: {
      qlong lineNumber = lineNumberFromPoint( (qpoint*)lParam );
      return lineNumber!=0; } // Start drag if the mouse is over a line
    case DD_SETDRAGVALUE: {
      FLDdragDrop* dragDrop = (FLDdragDrop*)lParam;
      EXTfldval dtype( dragDrop->mDragType );
      dtype.setLong( cFLDdragDrop_dragData ); // We are dragging data
      EXTfldval dval( dragDrop->mDragValue );
      dval.setList( mLocalList, qfalse); // Set data to our list
      return qtrue; }
    default: // Not processed
      #ifdef isRCC
        return 0xffffffffL; // Web component
      #endif
      return DefWindowProc(hWnd,pMsg,wParam,lParam);
    }
  }

  //#2: this example is taken from the droplist control
qbool tqfDroplist::dragDrop( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, qbool pDoEdwc)
{
  switch( message ) {
    case WM_DRAGDROP:
    {
      switch ( wParam ) {
        case DD_CANDRAG_ON_DOWN:
        case DD_CANDRAG_ON_MOVE:  {
          if ( mIsDropped ) return qfalse;
          if ( !isDesign() && !readOnly() && udDragMode()==cFLDdragDrop_dragData ) {
            qpoint pt = *((qpoint*)lParam);
            qrect r; WNDgetClientRect( wnd(), &r );
            r.left = r.right-mButtonWidth;
            if ( GDIptInRect( &r, &pt ) ) return qfalse; //don't allow data dragging from the button
          }
          break;
        }
        case DD_SETDRAGVALUE:
        {
          if ( !isDesign() && dDragMode()==cFLDdragDrop_dragData && !!mList) {
            FLDdragDrop *dragDrop = (FLDdragDrop *) lParam;
            dragDrop->setDragType( cFLDdragDrop_dragData );
```

```
            fldval* dval = dragDrop->getDragValue( qtrue );
            dval->setlist( mList.getlstptr(), qfalse ); //don't make copy of list
            return qtrue;
          }
          break;
        }
      }
    }
  }
  return tqfld::dragDrop( hWnd, message, wParam, lParam, pDoEdwc );
}
```

See also FLDdragDrop

WM_ERASEBKGND

The WM_ERASEBKGND message is sent when the window background needs to be erased (for example, when a window is sized). It is sent to prepare an invalidated region for painting. From v3.1 onwards you should always call WNDdrawThemeBackground and only erase the area manually if the function returns qfalse. See example below.

Parameters:

- **hdc** - Value of wParam. Identifies the device context.

Returns:

An external component should return non-zero if it erases the background; otherwise, it should return zero.

Example:

```
HDC dc = (HDC)wParam;
qrect cRect; WNDgetClientRect( hWnd, &cRect );
if ( !WNDdrawThemeBackground(hWnd,dc,&cRect,WND_BK_DEFAULT) )
{
  GDIsetTextColor( dc, GDI_COLOR_WINDOW );
  GDIfillRect( dc, &cRect, GDIgetStockBrush( BLACK_BRUSH ) );
}
return 1L;
```

WM_FOCUSCHANGED

This message is generated when the input focus has been changed by Omnis. If a window displays an input caret, this is the time when the caret should be created or destroyed.

Parameters:

- focus - Value of wParam. If zero, the window looses the input focus. If 1 the window receives the input focus.

Returns:

An external component should return non-zero if it creates or destroys the caret; otherwise, it should return zero.

Example:

```
// this example creates a text input cursor at character position 7
// of some text the control is displaying
switch ( message )
{
  case WM_FOCUSCHANGED:
  {
```

```
    if ( wParam )
    {
      // create the caret, if ovrTypeOn is qtrue, we in over type mode
      // and the caret is displayed as a block, otherwise we are in
      // insert mode, and the caret is displayed as a vertical line.
      GDItextSpecStruct tSpec(fntEdit,styPlain,GDI_COLOR_WINDOWTEXT,jstLeft);
      str255 text( "This is an example for WNDcreateCaret" )
      qdim caretHeight = GDIfontPart( &tSpec.mFnt, tSpec.mSty, eFontHeight );
      qdim caretWidth = ( ovrTypeOn ? GDIcharWidth( text[8], &tSpec ) : 1 );
      qdim caretLeft = GDItextWidth( &text[1], 7, &tSpec );
      qrect cRect;
      qpoint pt;
      WNDgetClientRect( hWnd, &cRect );
      pt.h = cRect.left + caretLeft;
      pt.v = cRect.top + 1;
      WNDcreateCaret( hWnd, caretWidth, caretHeight );
      WNDsetCaretPos( &pt );
      WNDshowCaret();
    }
    else
    {
      // destroy the caret
      WNDdestroyCaret( hWnd );
    }
    return 1L;
  }
}
```

WM_GETERASEINFO

This message is generated during non-client painting of a window, when the non-client area contains dead areas which need to be erased. Dead areas occur when a window has both horizontal and vertical scrollbars, or has a shadow border style. In response to this message the WNDeraseInfoStruct must be filled in. A pointer to this structure is supplied in lParam.

Parameters:

- **eraseInfo** - Value of lParam. Pointer to the WNDeraseInfoStruct .

Returns:

Always return 0.

Example:

```
WNDeraseInfoStruct *eraseInfo = (WNDeraseInfoStruct*)lParam;
eraseInfo->mBackColor = colWhite;
eraseInfo->mForeColor = colBlack;
eraseInfo->mFillPattern = patFill;
return 0L;
```

WM_GETMINMAXINFO

The WM_GETMINMAXINFO message is sent to a window whenever Omnis needs the maximum or minimum tracking size of the window. The maximum tracking size of a window is the largest window size that can be achieved by using the borders to size the window. The minimum tracking size of a window is the smallest window size that can be achieved by using the borders to size the window. This message is not usually generated by the HWND module (except during border dragging, see WM_DRAGBORDER), but by other parts of Omnis, for example, window design. However, the function WNDgetMinMaxInfo can be used to assist in calculating the minMaxInfo of a component window when this message is received. When components are used this function should always be called. It is recursive in that it generates further WM_GETMINMAXINFO messages for all child windows. After calling this function, any further restrictions can be applied to the minMaxInfo it calculated.

Parameters:

- **minMaxInfo** - Value of lParam. Pointer to the WNDminMaxInfo struct .

Returns:

Always return 1.

Example:

```
WNDminMaxInfo* minMaxInfo = (WNDminMaxInfo*)lParam;
WNDgetMinMaxInfo( hWnd, minMaxInfo );
if ( minMaxInfo->ptMinTrackSize < 100 ) minMaxInfo->ptMinTrackSize = 100;
if ( minMaxInfo->ptMaxTrackSize > 400 ) minMaxInfo->ptMinTrackSize = 400;
return 1L;
```

WM_GETSHADOWRECT (Mac OSX only)

The WM_GETSHADOWRECT message is send when the HWND manager needs to paint a WND_BORD_CTRL_TABPANE or WND_BORD_CTRL_SHADOW_EX border on Mac OSX. This allows the control to manipulate the rect prior to it being drawn. If the border is to be drawn as is, you do not need to respond to this message.

Parameters:

- **theRectPtr** - Value of lParam. Contains pointer to the qrect at which the border will be painted. The rect will be local to the client area of the given HWND.

Example:

```
qrect* theRectPtr = (qrect*)lParam;
theRectPtr->top += 10;
return 1L;
```

WM_HSCROLL and WM_VSCROLL

The WM_HSCROLL message is sent to a window when the user clicks the window's horizontal scroll bar (WM_VSCROLL if the user clicks the vertical scrollbar).

Parameters:

- **wScrollCode** - Value of wParam. Specifies a scroll bar code that indicates the user's scrolling request. This parameter can be one of the following values:

SB_ENDSCROLL	End scroll.
SB_LEFT or SB_TOP	Scroll to far left or top.
SB_LINELEFT or **SB_LINEUP**	Scroll one line left or up.
SB_LINERIGHT or **SB_LINEDOWN**	Scroll one line right or down.
SB_PAGELEFT or **SB_PAGEUP**	Scroll one page left or up.
SB_PAGERIGHT or **SB_PAGEDOWN**	Scroll one page right or down.
SB_RIGHT or **SB_BOTTOM**	Scroll to far right or bottom.

SB_THUMBPOSITION	Scroll to absolute position. The current position is specified by the LOWORD of lParam.
SB_THUMBTRACK	Drag scroll box (thumb) to specified position. The current position is specified by the LOWORD of lParam.

- **nPos** - Value of the low-order word of lParam. Specifies the current position of the scroll box if the wScrollCode parameter is SB_THUMBPOSITION or SB_THUMBTRACK; otherwise, the nPos parameter is not used.

Returns:

An external component should return zero if it processes this message.

The SB_THUMBTRACK scroll bar code is typically used by external components that give some feedback while the scroll box is being dragged.

If an external component scrolls the contents of the window (see WNDscrollWindow), it must also reset the position of the scroll box by using the WNDsetScrollPos function.

Example:

```
qdim min, max, page, oldPos, newPos;
qshort sbar = ( message == WM_HSCROLL ? SB_HORZ : SB_VERT );
// get current scrollbar settings
WNDgetScrollPos( hWnd, sbar, &oldPos );
WNDgetScrollRange( hWnd, sbar, &min, &max, &page );
// calculate newPos appropriately
switch ( wParam )
{
  // in this example 1 scroll unit equals 8 pixels
  case SB_LINEDOWN: newPos = oldPos + 8; break;
  case SB_LIENUP: newPos = oldPos - 8; break;
  case SB_PAGEDOWN: newPos = oldPos + page; break;
  case SB_PAGEUP: newPos = oldPos - page; break;
  case SB_TOP: newPos = min; break;
  case SB_BOTTOM: newPos = max; break;
  case SB_THUMBPOSITION:
  case SB_THUMBTRACK:
  {
    // handle sign extension correctly
    qshort shortNewPos = LOWORD( lParam );
    newPos = shortNewPos;
    break;
  }
  case SB_ENDSCROLL: return 1L;
  default: newPos = oldPos; break;
}
if ( newPos != oldPos )
{
  qdim hOff = ( sbar == SB_HORZ ? oldPos - newPos : 0 );
  qdim vOff = ( sbar == SB_VERT ? oldPos - newPos : 0 );
  WNDsetScrollPos( hWnd, sbar, newPos, qtrue );
  WNDscrollWindow( hWnd, hOff, vOff );
}
```

WM_IPHONE_ROUNDRECT_TEXTFIELDSTYLE

This message should return qtrue if the iPhone rounded rectangle border is the same as the UITextField border, qfalse otherwise. This message applies only to iPhone client component development (Studio v5.0).

Parameters:

None.

WM_KEYxxx

The WM_KEYDOWN message is sent when a key is pressed and the window has the key capture (See function WNDsetCapture). If a parent window has the WND_KEYPREVIEW style, the WM_KEYDOWNPREVIEW message is sent to that parent prior to the child receiving the WM_KEYDOWN message. WM_KEYUP and WM_KEYUPPREVIEW messages are generated when the key is released.

- **WM_KEYDOWN**
  sent to the window who has the key capture.
- **WM_KEYUP**
  sent to the window who has the key capture.
- **WM_KEYDOWNPREVIEW**
  sent to all parents of the window who has the key capture, and the parents specify WND_KEYPREVIEW in their extended styles.
- **WM_KEYUPPREVIEW**
  sent to all parents of the window who has the key capture, and the parents specify WND_KEYPREVIEW in their extended styles.

Parameters:

- key - Value of lParam. Specifies a pointer to a qkey.

Returns:

An external component should return zero if it processes this message. Otherwise it must return 1, so Omnis can continue processing the key.

Example:

```
// in this example we are only interested in movement keys to scroll
// the window vertically on a WM_KEYDOWN message
qkey* key = (qkey*)lParam;
vchar vch = key->getVChar();
if ( vch ) switch ( vch )
{
  case vcUp:    wParam = SB_LINEUP;   break;
  case vcDown:  wParam = SB_LIENDOWN; break;
  case vcPup:   wParam = SB_PAGEUP;   break;
  case vcDown:  wParam = SB_PAGEDOWN; break;
  case vcHome:  wParam = SB_TOP;      break;
  case vcEnd:   wParam = SB_BOTTOM;   break;
  default:   return 1L;
}
else
{
  return 1L;
}
WNDsendMessage( hWnd, WM_VSCROLL, wParam, 0 );
return 0L;
```

WM_LBUTTONxxx and WM_RBUTTONxxx

The WM_LBUTTONxxx and WM_RBUTTONxxx messages are generated when the user operates the left or right mouse button.

- **WM_LBUTTONDOWN** or **WM_RBUTTONDOWN**
  The WM_LBUTTONDOWN or WM_RBUTTONDOWN message is sent when the user presses the left or right mouse button.
- **WM_LBUTTONUP** or **WM_RBUTTONUP**
  The WM_LBUTTONUP or WM_RBUTTONUP message is sent when the user releases the left or right mouse button.
- **WM_LBUTTONDBLCLK** or **WM_RBUTTONDBLCLK**
  The WM_LBUTTONDBLCLK or WM_RBUTTONDBLCLK message is sent when the user double clicks the left or right mouse button.

Note: Under MacOS right button clicks are generated by holding down the option key while operating the mouse button.

Parameters:

- **point** - Value of lParam. Specifies the point as a long value (use WNDmakePoint to retrieve the point). The point is local to the client area of the window which receives these messages.

Returns:

An external component should return zero if it processes this message.

Example:

```
// This is an example for a simple pushbutton dealing with mouse tracking
// when the user clicks on the button.
switch ( message )
{
  case WM_LBUTTONDOWN:
  {
    if ( ! WNDhasCapture( hWnd, WND_CAPTURE_MOUSE ) )
    {
      // set the capture for mouse tracking
      WNDsetCapture( hWnd, WND_CAPTURE_MOUSE );
      // paint the button in the down position
      // remember the position in a member
      mDown = qtrue;
    }
    return 0L;
  }
  case WM_MOUSEMOVE:
  {
    if ( WNDhasCapture( hWnd, WND_CAPTURE_MOUSE ) )
    {
      qpoint pt; WNDmakePoint( lParam, &pt );
      qrect cRect; WNDgetClientRect( hWnd, &cRect );
      if ( mDown != GDIptInRect( &cRect, &pt ) )
      {
        // the user has moved the mouse out of the button,
        // or into the button.
        // paint the button up or down
        mDown = !mDown;
      }
    }
    return 0L;
  }
  case WM_LBUTTONUP:
  {
```

```
    if ( WNDhasCapture( hWnd, WND_CAPTURE_MOUSE ) )
    {
      // tracking has finished, release the mouse capture
      WNDreleaseCapture( hWnd, WND_CAPTURE_MOUSE );
      // was the mouse button released inside the client area
      if ( mDown )
      {
        // do something
        // paint the button in the up position
        mDown = qfalse;
      }
    }
  return 0L;
  }
}
```

WM_MOUSEMOVE

The WM_MOUSEMOVE message is sent to a window when the mouse cursor moves. If the mouse is not captured, the message goes to the window beneath the cursor. Otherwise, the message goes to the window that has captured the mouse.

Parameters:

- **point** - Value of lParam. Specifies the point as a long (use WNDmakePoint to retrieve the point). The point is local to the client area of the window which receives these messages.

Returns:

An external component should return zero if it processes this message.

Example:

See WM_LBUTTONDOWN

WM_MOUSEWHEEL

The WM_MOUSEWHEEL message is sent to a window when the mouse wheel is rotated.

If the mouse is not captured, the message goes to the window beneath the cursor. Otherwise, the message goes to the window that has captured the mouse.

Parameters:

direction – Value of wParam. The high-order word indicates the distance the wheel is rotated. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

Returns:

An external component should return zero if it processes this message.

Notes:

On macOS

1) The direction value will be either 0, 1 or -1. No other values are supported.

2) Controls that have no scroll bars added will need to respond to WM_FLD_NEEDSWM_MOUSEWHEEL to receive a WM_MOUSEWHEEL message (see example)

Example:

```
// This is an example handing WM_MOUSEWHEEL
// Processing to be added to WNDPROC..
switch ( message )
{
```

```
// controls with no scroll bars need to return 1L for the WM_FLD_NEEDSWM_MOUSEWHEEL message to receive WM_MO
case WM_FLD_NEEDSWM_MOUSEWHEEL: return 1L;
//
case WM_MOUSEWHEEL:
{
  // Mouse wheel moved
  if ( wParam )
  {
    qbool lineUp = ((qshort)HIWORD(wParam)) >= 0;
  }
  // return 0 to indicate this control has processed this message.
  return 0L;
}
```

WM_MULTIKEYDOWNPREVIEW

Sent to parent window on a WM_MULTIKEYDOWN event if parent has WND_KEYPREVIEW set (currently WM_MULTIKEYDOWN only applies to Mac OSX).

Parameters:

- **Keys** – Value of lParam. This is a pointer to WNDmultiKey class instance (defined in hwnd.he) and contains the key combination being held down.

Returns:

An external component should return zero if it processes this message.

WM_NCACTIVATE

The WM_NCACTIVATE message is sent when a window is activated or deactivated. A window becomes active when it becomes the topmost window (ignoring all floating palette windows).

Parameters:

- **active** - Value of wParam. This is a boolean value. If qtrue, the window has become active, otherwise the window has become inactive.

Returns:

Always return zero.

Example:

```
// in this example the control needs to draw its control items disabled
// when a window is not active.
qbool active = (qbool)wParam;
if ( active )
{
  // paint control enabled
}
else
{
  // paint control disabled
}
```

WM_NCLBUTTONDOWN

The WM_NCLBUTTONDOWN message is sent when the left mouse button has been held down over the non-client area of a window.

Parameters:

- **hittest** - Value of wParam. Specifies the hit-test area code. This parameter is one of the Htxxx defines.
- **point** - Value of lParam. Specifies the point as a long (use WNDmakePoint to retrieve the point). The point is local to the desktop.

Returns:

An external component should return zero if it processes this message.

Example:

```
// this example lets the user drag the window via the top area of the
// border restricting the movements to the client area of the parent window
if ( wParam == HTBORDER )
{
  // test if the mouse is in the top part of the border
  qpoint pt; WNDmakePoint( lParam, &pt );
  qrect cRect; WNDgetClientRect( hWnd, &cRect );
  // map client rect to desktop
  WNDmapWindowRect( hWnd, HWND_DESKTOP, &cRect );
  if ( pt->v <= cRect.top )
  {
    // restrict movement of mouse to parent's client area
    HWND parentHwnd = WNDgetParent( hWnd );
    WNDgetClientRect( parentHwnd, &cRect );
    WNDmapWindowRect( parentHwnd, HWND_DESKTOP, &cRect );
    WNDclipCursor( &cRect );
    // loop following the mouse movements while the button is held down
    qpoint lastPt = pt;
    qpoint curPt;
    while ( WNDmouseLeftButtonDown() )
    {
      WNDgetCursorPos( &curPt );
      qdim hDiff = curPt.h - lastPt.h;
      qdim vDiff = curPt.v - lastPt.v;
      if ( hDiff || vDiff )
      {
        // move the window
        qrect wRect; WNDgetWindowRect( hWnd, &wRect );
        // WNDmoveWindow expects coordinates for the window local to
        // the parent's client area
        WNDmapWindowRect( HWND_DESKTOP, parentHwnd, &wRect );
        WNDmoveWindow( hWnd, wRect.left + hDiff, wRect.top + vDiff, wRect.width(), wRect.height(), qtrue );
        // update parent and all children to give immediate feedback
        // to the user
        WNDredrawWindow( parentHwnd, NULL, NULL, WND_RW_UPDATE | WND_RW_ALLCHILDREN | WND_RW_ERASE );
        lastPt = curPt;
      }
    }
    // must clear cursor clipping before returning
    WNDclipCursor( NULL );
    return 0L;
  }
}
return 1L;
```

WM_NULL

This message is generated while the computer is idle, that is, no other messages are pending, and the mouse capture is set (see WNDsetCapture).

Parameters:

None.

Returns:

Always return 1.

WM_PAINT

The WM_PAINT message is sent when a portion of a window needs repainting. To repaint a window the function WNDbeginPaint must be called, followed by a call to WNDendPaint after all painting has been done.

Note: Nested calls to WNDbeginPaint or WNDstartDraw are not supported and result in a runtime error.

Parameters:

None.

Returns:

An external component should return zero if it processes this message.

Example:

```
WNDpaintStruct paintInfo;
WNDbeginPaint( hWnd, &paintInfo );
// paint the control
WNDendPaint( hWnd, &paintInfo );
return 0L;
```

WM_PRI_INSTALL

The WM_PRI_INSTALL message is sent when the print manager opens the page preview or screen report, and an alternative hwnd has been specified in PRIdestParmStruct when calling PRIstartJob or PRIredirectJob. For more information about printing refer to the print manager documentation.

It is possible to write external components which will display screen or preview reports. When a report is about to be displayed in the given HWND, WM_PRI_INSTALL is sent to the HWND. When the report is closed, WM_PRI_REMOVE is sent.

The component will be responsible for killing an active print job when a new job is about to use the HWND. The component must store the pointer to the job which currently occupies the HWND.

Parameters:

- **job** - Value of lParam. Specifies the pointer to the print job, PRIjob.
- **device** - Value of uParam. Specifies the pointer to the output device.

Returns:

An external component should return 1L if it processes this message.

Example:

```
// if there is an existing job occupying the hwnd, kill the job
if ( mJob ) PRIdefOutputProc( mJob, mOutput, PM_OUT_KILL, 0, 0, 0 );
// remember the new job and device
mJob = (PRIjob)lParam;
mOutput = (void*)uParam;
return 1L;
```

WM_PRI_REMOVE

The WM_PRI_REMOVE message is sent when the print manager closes the page preview or screen report, and an alternative hwnd has been specified in PRIdestParmStruct when calling PRIstartJob or PRIredirectJob. See WM_PRI_INSTALL.

The component will be responsible for killing an active print job when a new job is about to use the HWND. The component must store the pointer to the job which currently occupies the HWND.

Parameters:

None.

Returns:

An external component should return 1L if it processes this message.

Example:

```
// clear the job and device
mJob = 0;
mOutput = 0;
return 1L;
```

WM_RBUTTONxxx

See WM_LBUTTONxxx.

WM_OSXREPAINTPLUGIN (v5.0)

This message informs an OSX browser plugin that it needs to repaint. It is issued by WNDabortMouseCapture()

Parameters:

None.

Example:

```
case WM_OSXREPAINTPLUGIN:
{
  mThis->mNeedPaint = qtrue;
  break;
}
```

WM_SETCURSOR

WM_SETCURSOR is generated every time the mouse moves across a window and the mouse capture has not been set. If WM_SETCURSOR is passed on to the DefWindowProc the cursor is set to the arrow cursor if the mouse is over the non-client area of the window. If the function WNDcheckCursor is called in response to this message, the HWND module sets the cursor to the appropriate cursor depending on the cursor associated with the window or the windows parents (see WNDsetWindowCursor).

Parameters:

- **hwndCursor** - Value of wParam. Specifies the HWND that contains the cursor.
- **hittest** - Value of the low-order word of lParam. Specifies the hit-test area code. This parameter can be one of the Htxxx defines.
- **mouseMsg** - Value of the high-order word of lParam. Specifies the number of the mouse message.

If nHittest is set to HTCLIENT, the window procedure should call WNDcheckCursor.

Note: While the mouse capture is on, no WM_SETCURSOR messages are generated.

Example:

```
// in this example the cursor is set to a drag cursor when the mouse is over
// the top part of the border
qword2 hittest = LOWORD(lParam);
if ( hittest == HTBORDER )
{
  // test if we are in the top part of the windows border
  qpoint pt; WNDgetCursorPos( &pt );
  qrect cRect; WNDgetClientRect( hWnd, &cRect );
  // map client rect to desktop
  WNDmapWindowRect( hWnd, HWND_DESKTOP, &cRect );
  if ( pt.v <= cRect.top )
  {
    WNDsetCursor( WND_CURS_DRAG_OBJECT );
    return 1L;
  }
}
WNDcheckCursor( hWnd, hittest );
return 1L;
```

WM_SHOWSIZEGRIP

The WM_SHOWSIZEGRIP message is sent when the HWND module needs to query the window as regards to properties of the grow box within the client area of the window. If a window has both horizontal and vertical scrollbars, the grow box is displayed in the non-client area, and no WM_SHOWSIZEGRIP message is generated.

Parameters:

- **submerge** - value of wParam. This parameter is one of the following values:

WND_GRIP_ALLOWED
Is the grow box allowed to be in the client area. Return one of the following:

WND_GRIP_ALLOW_NO No grow box is allowed.
WND_GRIP_ALLOW_YES The grow box is allowed.
WND_GRIP_ALLOW_STOP The grow box is allowed but not visible.

Note: Under MacOS, the return value is ignored. A grow box is always enforced, if the MacOS window has been given the grow box property.

WND_GRIP_GET_RECT
Position the supplied grow box rect. lParam points to a qrect which is already positioned for the bottom right corner of the client area. The window can adjust this rectangle, so the grow box is painted in the correct location within the client area of the window (DO NOT alter the width or height of the rect). WND_GRIP_GET_RECT should return the same value which was returned by WND_GRIP_ALLOWED.

WND_GRIP_CHANGED
The grow box is removed or added to the window.

- **growboxrect** - value of lParam. A pointer to the grow box's rectangle is supplied in this parameter, if the sub-message is WND_GRIP_GET_RECT.

Returns:

For WND_GRIP_ALLOWED return one of the WND_GRIP_ALLOW_xxx flags.

For WND_GRIP_GET_RECT return the same value which is returned by WND_GRIP_ALLOWED.

For WND_GRIP_CHANGED always return 0.

Example:

```
// this window has no scrollbars, so it needs to implement the WM_SHOWSIZEGRIP
// messages if it wants to allow a growbox in its client area
switch ( wParam )
{
  case WND_GRIP_ALLOWED:
  {
    return WND_GRIP_ALLOW_YES;
  }
  case WND_GRIP_GET_RECT:
  {
    // rect is already positioned for bottom right corner of the client
    // area, but you want to bring it in an additional 2 pixels to give
    // as room for our special border
    qrect* theRect = (qrect*)lParam;
    GDIoffsetRect( theRect, -2, -2 );
    return WND_GRIP_ALLOW_YES;
  }
  case WND_GRIP_CHANGED:
  {
    // get the growbox rect so we can invalidate it. Note:
    // WNDgetGrowBoxRect generates a WND_GRIP_GET_RECT message.
    qrect theRect; WNDgetGrowBoxRect( hWnd, &theRect );
    WNDinvalidateRect( hWnd, &theRect );
  }
}
```

WM_SHOWWINDOW

The WM_SHOWWINDOW message is sent when the function WNDshowWindow is called to show or hide a window.

Parameters:

- **show** - Value of wParam. If window is shown, this value is one, otherwise it is zero.

Returns:

An external component should return zero if it processes this message.

Example:

```
// this example only allows the window to be shown if the member mVisible is qtrue.
if ( wParam && !mVisible ) return 0L;
return 1L;
```

WM_TIMER

The WM_TIMER message is sent to the WndProc function of a window after each interval which was specified when the WNDsetTimer function was called to install the timer.

Note: Because WM_TIMER messages are only generated if no other messages are on the message queue, the accuracy of the intervals at which they are generated cannot be guaranteed, that is, while Omnis is busy, no WM_TIMER messages are generated.

Parameters:

- **timerID** - Value of wParam. The timer id which was specified when the timer was installed.

Returns:

Always return zero.

Example:

```
// this example implements a typical about box behavior by destroying the window
// after it has been around for 5 seconds or on a mouse button up. While the mouse
// button is held down, the window stays around.
switch ( message )
{
  case WM_PAINT:
  {
    // leave the window around for 5 seconds after the first paint
    static qbool sTimerSet = qfalse;
    WNDpaintStruct paintInfo;
    WNDbeginPaint( hWnd, &paintInfo );
    // paint the window
    WNDendPaint( hWnd, &paintInfo );
    if ( ! sTimerSet )
    {
      WNDsetTimer( hWnd, 1, 5000 );
      sTimerSet = qtrue;
    }
    return 0L;
  }
  case WM_TIMER:
  {
    // 5 seconds later destroy the window if the mouse button
    // is not held down
    WNDkillTimer( hWnd, 1 );
    if ( !WNDmouseLeftButtonDown() )
    {
      WNDdestroyWindow( hWnd );
    }
    return 0L;
  }
  case WM_LBUTTONDOWN:
  {
    return 0L;
  }
  case WM_LBUTTONUP:
  {
    // always destroy the window on a button up
    WNDdestroyWindow( hWnd );
    return 0L;
  }
}
```

WM_VSCROLL

See WM_HSCROLL.

WM_WINDOWPOSCHANGED

The WM_WINDOWPOSCHANGED message is sent to a window whose size, position, visibility, or z-order has changed as a result of a call to WNDsetWindowPos or another window-management function.

Parameters:

- **wpos** - Value of lParam. Points to a WNDwindowPosStruct data structure that contains information about the window's new size and position.

Returns:

Always return 1.

Example:

```
// this example resets the scroll ranges if the width or height of the window has been changed
WNDwindowPosStruct* windowPosInfo = (WNDwindowPosStruct*)lParam;
if ( ( windowPosInfo->flags & SWP_NOSIZE ) == 0 )
{
  qdim min, max, page;
  WNDgetScrollRange( windowPosInfo->hwnd, SB_HORZ, &min, &max, &page );
  page = windowPosInfo->cx;
  WNDsetScrollRange( windowPosInfo->hwnd, SB_HORZ, min, max, page );
  WNDgetScrollRange( windowPosInfo->hwnd, SB_VERT, &min, &max, &page );
  page = windowPosInfo->cy;
  WNDsetScrollRange( windowPosInfo->hwnd, SB_HORZ, min, max, page );
}
return 1L;
```

WM_WINDOWPOSCHANGING

The WM_WINDOWPOSCHANGING message is sent to a window whose size, position, visibility, or z-order is about to be changed as a result of a call to WNDsetWindowPos or another window-management function.

Parameters:

- **wpos** - Value of lParam. Points to a WNDwindowPosStruct data structure that contains information about the window's new size and position.

Returns:

An external component should return zero if it processes this message.

During this message, modifying any of the values in the WNDwindowPosStruct structure affects the new size, position, or z-order. An external component can prevent changes to the window by setting or clearing the appropriate bits in the flags member of the WNDwindowPosStruct structure.

Example:

```
// this example prevents the window being sized outside a specific size range
WNDwindowPosStruct* windowPosInfo = (WNDwindowPosStruct*)lParam;
if ( ( windowPosInfo->flags & SWP_NOSIZE ) == 0 )
{
  if ( windowPosInfo->cx < 100 )
    windowPosInfo->cx = 100;
  else if ( windowPosInfo->cx > 400 )
    windowPosInfo->cx = 400;
  if ( windowPosInfo->cy < 80 )
    windowPosInfo->cy = 80;
  else if ( windowPosInfo->cy > 200 )
    windowPosInfo->cy = 200;
}
return 1L;
```

Functions

HIWORD()

qword2 HIWORD(qword4 pVal)

Returns the high order word of the given long value.

LOWORD()

```
qword2 LOWORD( qword4 pVal )
```

Returns the low order word of the given long value.

WNDabortMouseCapture()

```
void WNDabortMouseCapture()
```

Aborts mouse capture as a result of some user action elsewhere in the process, e.g. CMND+N to open a new window in a web browser. Sends WM_CAPTUREABORT to the hwnd with the mouse capture, and then releases the mouse capture.

WNDaddWindowComponent()

```
HWND WNDaddWindowComponent(
HWND pHwnd, qulong pComponent,
qulong pStyle, qulong pExStyle,
WNDprocClass* pObject, qdim pSize OR
qrect pRect, WNDborderSpec* pBorderSpec
)
```

Adds a new window component to the specified parent. Adding components may cause the position of other components to be altered, and generates WM_PAINT messages if these components are visible.

- **pHwnd** - identifies the window to which to add the component.
- **pComponent** - specifies one of the following component types:

  WND_WC_MENUBAR
  WND_WC_TOOLBAR_TOP
  WND_WC_TOOLBAR_LEFT
  WND_WC_TOOLBAR_BOTTOM
  WND_WC_TOOLBAR_RIGHT
  WND_WC_HEADER_BUTTON
  WND_WC_MAIN_HEADER
  WND_WC_HORZ_HEADER
  WND_WC_VERT_HEADER
  WND_WC_CLIENT
  WND_WC_STATUSBAR
- **pStyle** - specifies the styles for the window. Same as for WNDcreateWindow.
- **pExStyle** - specifies the extended styles for the window. Same as for WNDcreateWindow.
- **pObject** - specifies the WNDprocClass instance which is to be associated with the new component.
- **pSize or pRect** - pSize specifies the height or width of the component (if pRect is specified only the height or width of the rectangle is used), depending on whether it is a horizontal or vertical component. If zero is passed, the default size applies. pSize is ignored for WND_WC_CLIENT and WND_WC_HEADER_BUTTON components.
- **pBorderSpec** - specifies the border style of the component.
- **return** - returns the new HWND of the component.

Example:

```
// in this example a window adds a menu and client component to it self
// when it is first created
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_CREATE:
    {
      qulong style = WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | WS_VISIBLE;
      WNDborderStruct border( WND_BORD_EMBOSSED );
      mMenuHwnd = WNDaddWindowComponent( hWnd, WND_WC_MENUBAR, style, WND_DRAGBORDER, this, 20, &border );
      style |= WS_HSCROLL | WS_VSCROLL;
      mClientHwnd = WNDaddWindowComponent( hWnd, WND_WC_CLIENT, style, WND_DRAGBORDER, this, 0, &border );
      return 0L;
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

WNDbeginPaint()

WNDprocClass* WNDbeginPaint(HWND pHwnd, WNDpaintStruct* pPaintStruct)

Prepares the given window for painting and fills a WNDpaintStruct structure with information about the painting. The WNDbeginPaint function automatically sets the visual region of the device context to exclude any area outside the update region. The update region is set by the WNDinvalidateRect or WNDinvalidateRgn function and by the HWND module after sizing, moving, creating, scrolling, or any other operation that affects the client area. If the update region is marked for erasing, WNDbeginPaint sends a WM_ERASEBKGND message to the window.

If the caret is in the area to be painted, WNDbeginPaint automatically hides the caret to prevent it from being erased.

Warning: This function must only be called in response to a WM_PAINT or WM_CHILDPAINT message, and must always be followed by a call to WNDendPaint before the next call to WNDbeginPaint (must NOT be nested).

- **pHwnd** - identifies the window to be repainted.
- **pPaintStruct** - points to the WNDpaintStruct structure that receives the painting information.
- **return** - returns a pointer to the WNDprocClass instance which is associated with this window. WARNING: the pointer is NULL if the window has no associated instance.

Example:

See WM_PAINT, WM_TIMER, WNDredrawChildren.

See also WNDendPaint, WNDstartDraw, WNDendDraw, WM_PAINT, WM_NCPAINT, WM_ERASEBKGND, WM_CHILDPAINT, WNDpaintStruct, HDC

WNDbringWindowToTop()

qbool WNDbringWindowToTop(HWND pHwnd)

Brings the given child window to the top of a stack of overlapping windows. The WNDbringWindowToTop function should be used to uncover any window that is partially or completely obscured by any overlapping windows.

Calling this function is similar to calling the WNDsetWindowPos function to change a window's position in the Z-order.

- **pHwnd** - identifies the window to bring to the top.

- **return** - returns qtrue if successful. Otherwise, it is qfalse.

Example:

```
// in this example a window brings itself to the top when it is being clicked on
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_LBUTTONDOWN:
    {
      WNDbringWindowToTop( hWnd );
      return 0L;
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

See also WNDsetWindowPos

WNDchangeComponentId()

qbool WNDchangeComponentId(HWND pHwnd, qulong pComponent)

Changes the component type of the given window. All the usual restrictions apply, that is, only one of each component type can be present in the same parent window at any one time.

- **pHwnd** - identifies the window who's component type is to change.
- **pComponent** - specifies the new component type.
- **return** - returns qtrue if successful.

Example:

```
WNDchangeComponentId( myComponentHwnd, WND_WC_CLIENT );
```

See also WNDaddWindowComponent, WNDremoveWindowComponent

WNDcheckCursor()

void WNDcheckCursor(HWND pHwnd, qword2 pHittest)

Checks the window to see if the cursor needs changing and changes it if necessary. This function should be called when a WM_SETCURSOR message is received.

- **pHwnd** - identifies the window to check.
- **pHittest** - specifies the hit test area code. If pHittest is set to HTCLIENT and the window's cursor is anything other than WND_CURS_DEFAULT, the cursor is changed. If pHittest is set to HTCLIENT and the window's cursor is set to WND_CURS_DEFAULT, the parent's window cursor is applied. If that parent's cursor is also WND_CURS_DEFAULT, the parent's parent is checked, etc. If none of the parents have a cursor set, the cursor is set to WND_CURS_ARROW. If pHittest is anything other than HTCLIENT, the cursor is set to WND_CURS_ARROW.

Example:

See WM_SETCURSOR.

See also WNDsetCursor, WNDgetCursor, WNDsetWindowCursor, WNDgetWindowCursor, WNDclipCursor, WNDgetCursorPos, WNDsetCursorPos, WM_SETCURSOR

WNDchildPaintBegin() (v3.1)

void* WNDchildPaintBegin(void* pChildPaintInfo, HWND pParentHwnd, HDC pParentHdc, HWND pChildHwnd, qrect* pChildRect, qrect* pClipRect)

This function allows you to paint child windows inside the parent DC during the parents paint at a location specified by the parent. This is useful for complex lists which use embedded controls to paint the list data (i.e. Omnis Complex Grid). Such a list will need to paint the same child multiply times, once for every visible row of the list.

You call WNDchildPaintBegin repeatedly for each child which requires painting. When the last child has been painted you must call WNDchildPaintEnd. You would repeat this for every row.

The children must be painted starting with the bottom most child, since visual regions of the children are ignored.

You should use GDIoffscreenPaintBegin and GDIoffscreenPaintEnd when painting each row, to avoid unwanted flicker while painting the children.

- **pChildPaintInfo** – the paint info returned by a previous call to WNDchildPaintBegin.
- **pParentHwnd** – identifies the parents HWND.
- **pParentHdc** – identifies the parents DC.
- **pChildHwnd** – identifies the HWND of the child to be painted.
- **pChildRect** – specifies the coordinates at which to paint the child.
- **pClipRect** – identifies the area in the parent which requires painting.

Example:

```
// start the parent update
WNDpaintStruct ps;
WNDbeginPaint( parentHwnd, &ps );
// prepare painting of rows
// for the benefit of the example we hard code the row height,
// and assume that the top of each child within each row is zero,
// and the left, right and bottom are correct
qlong rowHeight = 50;
qrect clientRect; WNDgetClientRect( parentHwnd, &clientRect );
qlong fstVisRow = 1;
qlong lstVisRow = ( clientRect.height() + rowHeight -1 ) / rowHeight;
qrect rowRect = clientRect; rowRect.bottom = rowHeight - 1;
void* offscreenInfo = 0;
HDC paintDC = ps.hdc;
// paint the rows
for ( qlong row = fstVisRow ; row<=lstVisRow ; row++ )
{
  // prepare the offscreen paint
  qrect paintRect = rowRect;
  qrect updRect = ps.rcPaint;
  void* offscreenInfo2 = GDIoffscreenPaintBegin( offscreenInfo, paintDC, paintRect, updRect );
  // if offscreenInfo2 == NULL this row doesn't intersect the update rect
  // so we don't need to paint anything
  if ( offscreenInfo2 )
  {
    offscreenInfo = offscreenInfo2;
    void* childInfo = 0;
    // erase the background prior to painting the children
```

```
    WNDdrawThemeBackground( parentHwnd, paintDC, &paintRect, WND_BK_CONTAINER );
    // get the bottom most child window
    HWND childHwnd = WNDgetWindow( parentHwnd, GW_CHILD );
    if ( childHwnd ) childHwnd = WNDgetWindow( parentHwnd, GW_HWNDLAST );
    // next through the children and paint them
    while ( childHwnd )
    {
      // calculate the childs rects
      qrect childUpdRect = updRect;
      qrect childRect; WNDgetWindowRect( childHwnd, &childRect );
      WNDmapWindowRect( HWND_DESKTOP, parentHwnd, &childRect );
      GDIoffsetRect( &childRect, -paintRect.left, paintRect.top-childRect.top );
      // prepare painting of child
      void* childInfo2 = WNDchildPaintBegin( childInfo, parentHwnd, paintDC, childHwnd, &childRect, &childUpdR
      // if childInfo2==NULL the child does not intersect the
      // childUpdRect and there is nothing to paint
      if ( childInfo2 )
      {
        childInfo = childInfo2;
        // paint the child
        WNDsendMessage( childHwnd, WM_PAINT, WPARAM(paintDC), 0 );
      }
      // get the next child, making the assumption that we only have
      // one level of children
      childHwnd = WNDgetWindow( childHwnd, GW_HWNDPREV );
    }
    // we have painted all children for this row, so we must finish off
    if (childInfo) WNDchildPaintEnd( childInfo );
  }
  // prepare for next row
  GDIoffsetRect( &rowRect, 0, rowHeight );
}
// we have painted all rows, so finish off offscreen paint
GDIoffscreenPaintEnd( offscreenInfo );
// finish the parent update
WNDendPaint( parentHwnd, &ps );
```

See also WNDchildPaintEnd, GDIoffscreenPaintBegin, GDIoffscreenPaintEnd

WNDchildPaintEnd() (v3.1)

void WNDchildPaintEnd(void* pChildPaintInfo)

This function completes the painting of child windows inside the parents DC. For a full description of WNDchildPaintBegin and WNDchildPaintEnd see WNDchildPaintBegin above.

See also WNDchildPaintBegin

WNDclipCursor()

void WNDclipCursor(qrect* pRect)

Clips the screen cursor to the specified rect, that is, the movement of the cursor is restricted to within the bounds of the rectangle.

- **pRect** - points to the rectangle which must be in screen coordinates. If this parameter is NULL, any clipping previously set by this function is cleared.

Example:

See WM_NCLBUTTONDOWN.

See also WNDsetCursor, WNDgetCursor, WNDsetWindowCursor, WNDgetWindowCursor, WNDgetCursorPos, WNDsetCursorPos

WNDcreateCaret()

```
void WNDcreateCaret( HWND pHwnd, qdim pWidth, qdim pHeight )
```

Creates a new shape for the system caret and assigns ownership of the caret to the given window. The WNDcreateCaret function destroys the previous caret automatically, if any, regardless of which child window owns the caret. Once created, the caret is initially hidden. To show the caret, use the WNDshowCaret function. A child window should create a caret only when it has the input focus (see WM_FOCUSCHANGED).

- **pHwnd** - identifies the window that owns the new caret.
- **pWidth** - specifies the width of the caret in pixels.
- **pHeight** - specifies the height of the caret in pixels.

Example:

See WM_FOCUSCHANGED.

See also WNDdestroyCaret, WNDgetCaretPos, WNDsetCaretPos, WNDhideCaret, WNDshowCaret, WM_FOCUSCHANGED

WNDcreateWindow()

```
HWND WNDcreateWindow( HWND
pParentHwnd, qulong pStyle, qulong
pExStyle, WNDprocClass* pObject, qrect*
pRect, WNDborderStruct* pBorderSpec )
```

Creates a window of type WND_WC_FRAME. The new window becomes the top most window in its parent.

- **pParentHwnd** - identifies the parent of the window being created.
- **pStyle** - specifies the styles for the window. The following styles can be passed in the pStyle parameter:
  WS_CHILD
  WS_CLIPSIBLINGS
  WS_CLIPCHILDREN
  WS_HSCROLL
  WS_VSCROLL
  WS_VISIBLE

Note: For Omnis child windows to work correctly, WS_CHILD, WS_CLIPSIBLINGS, and WS_CLIPCHILDREN must always be specified. If WS_VISIBLE is specified, the window is made visible.

- **pExStyle** - specifies special Omnis extended styles for the window. The following styles can be passed in the pExStyle parameter:
  WND_FLOAT_xxx
  WND_KEYPREVIEW
  WND_REDRAWONSIZE
  WND_TRANSPARENT
  WND_DRAGBORDER
  WND_NOFLOATCHILDREN
  WND_NOADJUSTCOMPONENTS
  WND_OSMESSAGES

- **pObject** - specifies the WNDprocClass instance which is to be associated with the new window.
- **pRect** - specifies the initial window rectangle local to the parent window's client area.
- **pBorderSpec** - specifies the border information for the window.
- **return** - returns the new HWND, or NULL if the module fails to create the window.

Example:

```
// this example subclasses the WNDprocClass for receiving messages for its
// windows and creates a window with an inset border, scrollbars and bottom
// and right floating properties so when the parent sizes, the new window
// sizes by the same amount
// the cMyWndProcClass declaration
class cMyWndProcClass : public WNDprocClass
{
  cMyWndProcClass() {} // default constructor
  ~cMyWndProcClass() {} // default destructor
  virtual qlong WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam );
};
// first instantiate the WNDprocClass
cMyWndProcClass myWndProc = new cMyWndProcClass();
// prepare for window creation
qrect myWRect( 10, 10, 100, 20 );
WNDborderStruct myBorder( WND_BORD_INSET );
// now create the window invisibly
HWND myHwnd = WNDcreateWindow
          (
            myParentHwnd,
            WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | WS_HSCROLL | WS_VSCROLL,
            WND_FLOAT_RIGHT | WND_FLOAT_BOTTOM,
            myWndProc,
            &myWRect,
            &myBorder
);
// WM_CREATE will have been sent by now, make the window visible
WNDshowWindow( myHwnd, SW_SHOW );
```

See also WNDaddWindowComponent, WNDdestroyWindow, WNDgetWindowComponent, WNDnextWindowComponent, WNDremoveWindowComponent, WM_CREATE

WNDdelay()

void WNDdelay(qlong pMilliSecs)

Delays program execution by the specified number of milliseconds.

- pMilliSecs - specifies the delay in milliseconds.

Example:

See WNDgetCursor.

WNDdestroyCaret()

void WNDdestroyCaret(HWND pHwnd)void WNDdestroyCaret()

Destroys the system caret. A child window should destroy the caret if it loses the input focus.

- **pHwnd** - if this parameter is specified, the caret is only destroyed if it belongs to the given window. If the window is NOT specified, the caret is destroyed regardless.

Note: External components should always specify the window parameter, to prevent destroying the caret if it is owned by another window.

Example:

See WM_FOCUSCHANGED.

See also WNDcreateCaret, WNDgetCaretPos, WNDsetCaretPos, WNDhideCaret, WNDshowCaret, WM_FOCUSCHANGED

WNDdestroyWindow()

qbool WNDdestroyWindow(HWND pHwnd)

Destroys the given window and all its children. When a window is destroyed, a WM_DESTROY message is sent to the window and all of its child windows. The window procedure can NOT prevent the windows from being destroyed.

- **pHwnd** - identifies the window to be destroyed.
- **return** - returns qtrue if successful. Otherwise, it is qfalse.

Example:

```
if ( WNDdestroyWindow( myHwnd ) )
{
  // window has been destroyed
}
else
{
  // window has NOT been destroyed
}
```

See WM_TIMER.

See also WNDaddWindowComponent, WNDcreateWindow, WNDgetWindowComponent, WNDnextWindowComponent, WNDremoveWindowComponent, WM_DESTROY

WNDdragAcceptFiles()

void WNDdragAcceptFiles(HWND pHwnd, qbool pAccept)

Registers whether a window accepts dropped files.

- **pHwnd** - identifies the window that is registering whether it will accept dropped files.
- **pAccept** – A value that indicates if the window identified by the hWnd parameter accepts dropped files. This value is qtrue to accept dropped files or qfalse to discontinue accepting dropped files.

WNDdrawThemeBackground() (v3.1)

```
qbool WNDdrawThemeBackground( HWND
pHwnd, HDC pHdc, qrect* pRect, qulong
pBKTheme )
```

Calling this function will erase the rectangle with the specified theme background.

- **pHwnd** - identifies the window to be erased.
- **pHdc** - identifies the device context for drawing.
- **pRect** - specifies the area to be erased.
- **pBKTheme** - specifies the theme for the erase. This can be one of the following:
  WND_BK_TEST - This will simply test if the window has a theme background specified (see GWL_BKTHEME). No drawing takes place. If the window has a theme the function returns qtrue.
  WND_BK_DEFAULT - The function will use the background theme as set by GWL_BKTHEME for drawing. If the window has no theme, no drawing takes place and the function returns qfalse.
  WND_BK_PARENT - Fill the area using the parents theme or erase colors. This will send a WM_GETERASEINFO message to the parent if the parent has no theme. Function returns qtrue.
  WND_BK_HILITE - The area is filled with the standard hilite colors. Function returns qtrue.
  WND_BK_NONE - No painting takes place, function returns qfalse.
  WND_BK_WINDOW - Area is filled with the standard window background theme. Function returns qtrue.
  WND_BK_CONTAINER - Area is filled with the standard container background theme. Function returns qtrue.
  WND_BK_TABPANE - Area is filled with the standard tab pane background theme. Function returns qtrue.
  WND_BK_TABSTRIP - Area is filled with the standard tab strip background theme. Function returns qtrue.
  WND_BK_CONTROL - Area is filled with the standard control background theme. Function returns qtrue.
  WND_BK_MENUBAR - Area is filled with the standard menu bar background theme. Function returns qtrue.
  WND_BK_MENU - Area is filled with the standard menu background theme. Function returns qtrue.

5. **returns** – qtrue if painting has taken place, otherwise returns qfalse.

Example:

See WM_ERASEBKGND

See also GWL_BKTHEME, WM_ERASEBKGND, WNDdrawThemeControl

WNDdrawThemeControl()

```
qbool WNDdrawThemeControl( HWND
hWnd, HDC pHdc, qulong pType, qulong
pFlags, qrect* pRect)
```

Draws the specified control using the systems current theme.

- **pHwnd** - identifies the controls window.
- **pHdc** - identifies the device context for painting.
- **pType** – identifies the control type. Please note that not all control types are supported on all platforms. The function will return false if a control can not be drawn. The control type can be one of the following:
  THEME_PUSHBUTTON - Draws a standard system button. The following flags can be used with this control: THEME_CONTROL_DISABLED THEME_CONTROL_PRESSED, THEME_CONTROL_HOT, THEME_CONTROL_DEFAULT.
  THEME_CHECKBOX - Draws a standard system checkbox. The following flags can be used with this control: THEME_CONTROL_DISABLED THEME_CONTROL_ACTIVE, THEME_CONTROL_PRESSED, THEME_CONTROL_HOT
  THEME_RADIOBUTTON - Draws a standard system radio button. The following flags can be used with this control: see

THEME_CHECKBOX
THEME_TABPANE - Draws a standard system tab pane control. The following flags can be used with this control: THEME_CONTROL_FRAM
THEME_CONTROL_CLIENT, THEME_CONTROL_HOT, THEME_CONTROL_DISABLED, THEME_CONTROL_ACTIVE, THEME_CONTROL_POS
THEME_CONTROL_POS_BOTTOM.
THEME_COMBOBOX - Draws a standard system combo box. The following flags can be used with this control: THEME_CONTROL_PRESSE
THEME_CONTROL_HOT.
THEME_SCROLLBAR - Draws a standard scrollbar.
THEME_HEADER - Draws a standard header. The following flags can be used with this control: THEME_CONTROL_PRESSED, THEME_CONTROL_HOT.
THEME_TOOLBAR - Draws a standard toolbar. The following flags can be used with this control: THEME_CONTROL_POS_TOP, THEME_CONTROL_POS_BOTTOM, THEME_CONTROL_POS_LEFT, THEME_CONTROL_POS_RIGHT.

- **pFlags** – additional drawing flags. See control types for flags which can be used. Please note that some flags may only apply to some platforms. The function will return false if a control can not be drawn using the given flags.
- **pRect** – points to the qrect structure specifying the co-ordinates for drawing the control.
- **returns** – qtrue if painting has taken place, otherwise returns qfalse and the control needs to be painted manually.

See also WNDdrawThemeBackground

WNDendDraw()

```
void WNDendDraw( HWND pHwnd, HDC pHdc )
```

Marks the end of painting in the given window. This function is required for each call to the WNDstartDraw function, but only after painting is complete.

WNDstartdraw and WNDendDraw can be used to paint a window without having received a WM_PAINT message.

- **pHwnd** - identifies the window that has been repainted.
- **pHdc** - identifies the device context to be released.

Example:

See WNDredrawChildren, WNDgetWindowFromPt, WNDpaintBorder.

See also WNDstartDraw, WNDbeginPaint, HDC (GDI document)

WNDendDrawEx()

```
void WNDendDrawEx( HWND pHwnd, HDC pHdc )
```

Identical to WNDendDraw on non-macOS platforms.

On macOS this marks the end of drawing to the port in the same way as WNDendDraw but does not end a drawing block. This should be paired with a previous call to WNDstartDrawEx but this does not have to be in the same scope as that call. This allows drawing blocks to be defined separately. All drawing instructions defined in blocks between this call and the preceding WNDstartDrawEx call will be invoked.

- **pHwnd** - identifies the window that has been repainted.
- **pHdc** - identifies the device context to be released.

Drawing Blocks

On macOS these define drawing instructions in the same scope between a pair of WNDstartDrawEx and WNDendDrawEx calls.

The start of a drawing block is defined using BLOCKSTART and the end of a drawing block using BLOCKEND(HDC), where HDC is the device context returned from the previous WNDstartDrawEx call. These calls are NOPs on non-macOS platforms.

These calls must be defined in the same scope or the compiler will report an error. There can be multiple drawing block pairs after a WNDstartDrawEx call. These are then terminated and the drawing blocks invoked with a call to WNDendDrawEx.

Example: (excerpt amended from Zoom control)

void tqfZoomIn::drawDesktopSelectRect(qpoint* pPoint)

```
omnis {   HDC desktop = WNDstartDrawEx( HWND_DESKTOP );   BLOCKVAR qrect zoom = getZoomRect( pPoint
); // Mark this as a BLOCKVAR as it is passed by reference in the following drawing block.   BLOCKSTART
GDIdrawFocusRect( desktop,&zoom );   BLOCKEND(desktop)   WNDendDrawEx( HWND_DESKTOP, desktop );   mLastMousePo
= *pPoint; }
```

Note that code defining a drawing block can also be used by painting code used with a DC created by WNDbeginPaint and not by WNDstartDrawEx. In this case BLOCKEND will just run the drawing block instructions immediately. The same thing will happen with bitmap/offscreen based DC's.

Example:

See WNDredrawChildren, WNDgetWindowFromPt, WNDpaintBorder.

See also WNDendDraw, WNDstartDraw, WNDstartDrawEx, WNDbeginPaint, HDC (GDI document)

WNDendPaint()

void WNDendPaint(HWND pHwnd, WNDpaintStruct* pPaintStruct)

Marks the end of painting in the given window. This function is required for each call to the WNDbeginPaint function, but only after painting is complete.

Warning: WNDbeginPaint and WNDendPaint must only be ever used in response to a WM_PAINT message.

- **pHwnd** - identifies the window that has been repainted.
- **pPaintStruct** - points to a WNDpaintStruct structure that contains the painting information retrieved by the WNDbeginPaint function.

Example:

See WM_PAINT, WM_TIMER, WNDredrawChildren.

See also WNDbeginPaint, WNDstartDraw, WNDendDraw, WNDpaintStruct

WNDenumChildWindows()

qbool WNDenumChildWindows(HWND pParentHwnd, WNDenumProc pEnumProc, LPARAM lParam)

Enumerates all child windows of the given window and calls the specified WNDenumProc function with the given lParam for each child.

Warning: Destroying the window which is currently being processed causes the system to crash. Changing the parent of a child window during the enumeration process may cause the child window not to be enumerated.

- **pParentHwnd** - identifies the parent window.

- **pEnumProc** - identifies the user WNDenumProc which is to be called for each enumerated child window. If this function wants the enumeration process to continue, qtrue must be returned, otherwise the enumeration process is halted.
- **lParam** - specifies the lParam value to be passed to pEnumProc.
- **return** - returns qtrue if all windows have been successfully enumerated. Returning qfalse from pEnumProc stops enumeration, and returns qfalse to the calling function. If the given window has no child windows, qtrue is returned.

Example:

```
// this example sends a key message to all child windows of a window and
// stops the enumeration process once a window has accepted the key
// enumeration methods which are called for every child window
static qbool sSendKeyDown( HWND hWnd, LPARAM lParam )
{
  return (qbool) ( WNDsendMessage( hWnd, WM_KEYDOWN, 0, lParam != 0 );
  // note: returning qfalse stops enumeration
}
static qbool sSendKeyUp( HWND hWnd, LPARAM lParam )
{
  return (qbool) ( WNDsendMessage( hWnd, WM_KEYUP, 0, lParam ) != 0 );
  // note: returning qfalse stops enumeration
}
// the parent window receiving a key message
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_KEYDOWN:
    case WM_KEYUP:
    {
      qbool result;
      if ( message == WM_KEYDOWN )
        result = WNDenumChildWindows( hWnd, sSendKeyDown, lParam );
      else
        result = WNDenumChildWindows( hWnd, sSendKeyUp, lParam );
      // check if a child accepted the key message
      if ( result )
      {
        // NO child accepted the key
        return 1L;
      }
      else
      {
        // the key was accepted by one of the child windows
        return 0L;
      }
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

See also WNDgetWindow

WNDfloatChildren()

void WNDfloatChildren(HWND pHwnd, qdim pXOffset, qdim pYOffset)

Sizes or moves all floating child windows of the given parent window by the specified amounts. A child window is sized or moved only if it has the appropriate floating styles.

Note: Their should be no need to call this function from outside the HWND module. The HWND module calls this function automatically when a window sizes.

- **pHwnd** - identifies the window whose children are to be floated.
- **pXOffset** - specifies the amount by which the parent window has altered in size horizontally.
- **pYOffset** - specifies the amount by which the parent window has altered in size vertically.

See also WNDsetWindowPos, WNDwindowPosStruct., WND_FLOAT_xxx, WND_NOFLOATCHILDREN, WM_WINDOWPOSCHANGING, WM_WINDOWPOSCHANGED

WNDgetBorderSpec()

void WNDgetBorderSpec(HWND pHwnd, WNDborderSpec* pBorderSpec)

Returns the windows border information.

- **pHwnd** - identifies the window.
- **pBorderSpec** - the windows border information is returned in this structure.

Example:

```
// this example gets the border spec and changes the border color if it has a plain border
WNDborderSpec border;
WNDgetBorderSpec( myHwnd, &border );
if ( border.mBorderStyle == WND_BORD_PLAIN )
{
  border.mLineStyle.setColor( GDI_COLOR_QRED );
  WNDsetBorderSpec( myHwnd, &border, qtrue );
}
```

See also WNDsetBorderSpec, WNDborderSpec

WNDgetCapture()

HWND WNDgetCapture(qulong pFlags)

Returns the window which has the specified capture.

- **pFlags** - specifies the capture for which to return the window. This parameter can be WND_CAPTURE_MOUSE or WND_CAPTURE_KEY. Only one of the two flags must be specified.
- **return** - returns the window that has the specified capture. NULL is returned if no window has the capture.

Example:

```
HWND keyCapture = WNDgetCapture( WND_CAPTURE_KEY );
HWND mouseCapture = WNDgetCapture( WND_CAPTURE_MOUSE );
qbool haveBothCaptures = qbool( myHwnd == keyCapture && myHwnd == mouseCapture );
// is the same as
qbool haveBothCaptures = qbool( WNDhasCapture( myHwnd, WND_CAPTURE_KEY ) && WNDhasCapture( myHwnd, WND_CAPTURE
```

See also WNDsetCapture, WNDhasCapture, WNDreleaseCapture, WND_CAPTURE_xxx

WNDgetCaretPos()

void WNDgetCaretPos(qpoint* pPos)

Retrieves the system caret position in client coordinates of the associated window.

- **pPos** - points to the qpoint structure which is to receive the coordinates.

Example:

```
// this example makes sure the caret position is within the client area of the window
qpoint pt; WNDgetCaretPos( &pt );
qrect cRect; WNDgetClientRect( myHwnd, &cRect );
// the assumed width of the caret is 1.
cRect.right -= 1;
// the assumed height of the caret is 8
cRect.bottom -= 8;
if ( !GDIptInRect( &cRect, &pt ) )
{
  if ( pt.h < cRect.left ) pt.h = cRect.left;
  else if ( pt.h > cRect.right ) pt.h = cRect.right;
  if ( pt.v < cRect.top ) pt.v = cRect.top;
  else if ( pt.v > cRect.bottom ) pt.v = cRect.bottom
  WNDsetCaretPos( &pt );
}
```

See also WNDcreateCaret, WNDdestroyCaret, WNDsetCaretPos, WNDhideCaret, WNDshowCaret, WM_FOCUSCHANGED

WNDgetClientRect()

void WNDgetClientRect(HWND pHwnd, qrect* pRect)

Retrieves the coordinates of the windows client area.

- **pHwnd** - identifies the window for which to return the client rect.
- **pRect** - points to the qrect structure which is to receive the coordinates local to the client area. Left and top are always zero.

Example:

See WM_NCLBUTTONDOWN, WM_SETCURSOR, WNDgetCaretPos, WNDgetWindowFromPt, WNDpaintBorder, WNDsetCursorPos.

See also WNDgetWindowRect

WNDgetCursor()

qshort WNDgetCursor()

Returns the id of the currently displayed screen cursor.

- **return** - returns one of the WND_CURS_xxx cursor ids.

Example:

```
// this example changes the screen cursor for a number of
// seconds and restores it
qshort oldCursor = WNDgetCursor();
WNDsetCursor( WND_CURS_WATCH );
WNDdelay( 5000 ); // wait 5 seconds
WNDsetCursor( oldCursor );
```

See also WNDsetCursor, WNDgetCursorPos, WNDsetCursorPos, WNDclipCursor, WNDgetWindowCursor, WNDsetWindowCursor

WNDgetCursorPos()

void WNDgetCursorPos(qpoint* pPoint)

Returns the location of the cursors Hotpoint in screen coordinates.

- **pPoint** - points to the qpoint structure which is to receive the cursor's screen position.

Example:

See WM_NCLBUTTONDOWN.

See also WNDsetCursorPos, WNDclipCursor, , WNDsetCursor, WNDgetWindowCursor, WNDsetWindowCursor

WNDgetFloat()

qulong WNDgetFloat(HWND pHwnd)

Returns the floating properties of the given window.

- **pHwnd** - identifies the window for which to return the floating properties.
- **return** - returns the WND_FLOAT_xxx flags of the window.

Example:

```
// this example retrieves the floating properties of a window and switches off the bottom edge floating.
qulong float = WNDgetFloat( myHwnd );
float &= ~WND_FLOAT_BOTTOM;
WNDsetFloat( myHwnd, float );
```

See also WNDsetFloat, WNDcreateWindow, WND_FLOAT_xxx

WNDgetGrowBoxRect()

qbool WNDgetGrowBoxRect(HWND pHwnd, qrect* pRect)

Returns the rectangle of the grow box, if the given window owns the grow box, and the grow box is located within the client area of the window. The grow box only appears in the client area of a window if it owns the grow box and the window has no scrollbars. This function is useful if a control wants to take into account the position of a possible grow box within the client area, for example, in the case of a status bar control, the panes sizes are restricted. Calling this function generates a WM_SHOWSIZEGRIP message for the given window.

- **pHwnd** - identifies the window for which to return the coordinates of the grow box.

- **pRect** - points to the qrect which is to receive the grow box's coordinates.
- **return** - returns qtrue if the given window has a grow box in its client area.

Example:

See WM_SHOWSIZEGRIP

See also WM_SHOWSIZEGRIP

WNDgetMainHwnd() (Win & Linux only)

HWND WNDgetMainHwnd()

This core callback method returns the value of mHwnd; a handle to the Omnis root window. This method was added for use with worker objects, and specifically for use in subscribing client delegates to a central timer object. The resulting HWND is used when starting/stopping the timer using the WNDsetTimer() & WNDkillTimer() callbacks. It is unlikely to be of use generally (and is not supported on macOS), since any messages your component will normally respond to are supplied with an HWND parameter.

- **return** - returns a handle to the main Omnis window.

Example:

```
if(!_timerID) //create timer if it does not exist
{
  #if defined(ismacosx)
    _timerID = WNDsetTimer(NULL, 0, TIMER_MS, _omnisTimer);
  #else
    _timerID = WNDsetTimer(WNDgetMainHwnd(), 0, TIMER_MS, _omnisTimer);
  #endif
}
```

WNDgetMinMaxInfo()

void WNDgetMinMaxInfo(HWND pHwnd, WNDminMaxInfo* pMinMaxInfo)

Calculates the basic minimum tracking sizes of the given window by querying all child windows and adding their minimum tracking sizes depending on the child's component type. WM_GETMINMAXINFO messages are generated for all child windows, and if these child windows call WNDgetMinMaxInfo, further WM_GETMINMAXINFO messages are generated for their children, and so on. All windows which are known to possibly contain child windows must implement the WM_GETMINMAXINFO message and must call this function prior to applying any additional restrictions to the minimum or maximum tracking sizes.

- **pHwnd** - identifies the window for which to calculate the minimum tracking sizes.
- **pMinMaxInfo** - points to the WNDminMaxInfo struct which is to receive the results.

Example:

See WM_GETMINMAXINFO.

See also WM_GETMINMAXINFO

WNDgetOS()

qbool WNDgetOS(HWND pHwnd, qlong pSelector, qlong pLngValue)

Returns or manipulates platform specific information about a window. What information is returned depends on the selector. All information is written to the given buffer.

- **pHwnd** - identifies the window for which to return the platform specific information.
- **pSelector** - platform specific selector. Different platforms have different selectors.

This can be one of the following:

GOS_WINDOW (Mac OS only)

Retrieves the MacOS window port of the given HWND.

Example:

```
CGrafPtr macGrafPtr;
WNDgetOS( myHwnd, GOS_WINDOW, (qlong)&macGrafPtr );
```

GOS_EVENT (Mac OS only)

Retrieves the MacOS event record for the currently executing window message.

Example:

```
EventRecord ev;
WNDgetOS( NULL, GOS_EVENT, &ev );
```

GOS_REGION (Mac OS only)

Retrieves the given window's requested visual region. There are additional modifiers which can be added to the selector. These are:

WND_CLIENT - returns the visual region of the client area. WND_FRAME - returns the visual region of the non-client and client area. WND_EXCLUDE_CLIENT - can be used together with WND_FRAME to get the visual region of the non-client area only. WND_EXCLUDE_SIBLINGS - if specified, all overlapping sibling windows are subtracted from the visual region. WND_EXCLUDE_CHILDREN - if specified, all child window regions are subtracted from the visual region. WND_LOCAL - if specified, the region is local to the non-client or client area depending on which was requested. If not specified the region is local to the MacOS window's port. WND_INTERSECT_MAC_VISUAL - if specified, the visual region is intersected with the visual region of the HWND's MacOS window. WND_EXCLUDE_FOCUS (V3.2) – if specified, the visual region does not include the area covered by the Mac OS focus rectangle.

Example:

```
// the region handle must be allocated by the caller
RgnHandle rgn = NewRgn();
// the next line returns the true visual region of the non-client
// and client area as the window can be seen on screen.
// The region is local to the MacOS window's port.
WNDgetOS( myHwnd, GOS_REGION | WND_FRAME | WND_EXCLUDE_SIBLINGS |
WND_EXCLUDE_CHILDREN | WND_INTERSECT_MAC_VISUAL, (qlong)rgn );
```

```
// the next line returns the true visual region of the client
// area as can be seen on screen, but includes all areas occupied
// by the windows children. The region is local to the MacOS
// window's port.
WNDgetOS(myHwnd, GOS_REGION | WND_CLIENT | WND_EXCLUDE_SIBLINGS |
WND_INTERSECT_MAC_VISUAL, (qlong)rgn );
// do NOT forget to dispose of the region when finished
DisposeRgn( rgn );
```

GOS_MACOS8 (Mac OS only)
Returns 1 if system is version 8 or above.

GOS_OFFSETHWNDS (Mac OS only)
This will offset the given HWND and its children by the qpoint pointed to by pLngValue. No painting takes place. This is useful for 3rd part plug-ins or applications which use our GDI and HWND dll to implement HWNDs. This selector should be called when the Macintosh window has been scrolled, and the HWND containers need to be repositioned in the port without causing any invalidation.

Example:

```
qpoint pt(0,20);
WNDgetOS( theTopHwnd, GOS_OFFSETHWNDS, (qlong)&pt );
```

GOS_CLIPHWNDS (Mac OS only)
This will clip the given HWND and its children to the given rectangle which must be local to the Macintosh port to which the HWND belongs. This is useful for 3rd part plug-ins or applications which use our GDI and HWND dll to implement HWNDs. The selector should be called to prevent HWNDs painting over areas in the Mac port which they are not to paint in.

Example:

```
qrect r(0,0,400,300);
WNDgetOS( theTopHwnd, GOS_CLIPHWNDS, (qlong)&r );
```

- **pLngValue** - this should point to the buffer which receives/gives the information. The buffer size and type depends on the selector.
- **return** - returns qfalse if an invalid selector was specified.

WNDgetParent()

HWND WNDgetParent(HWND pHwnd)

Returns the parent window of the given window.

Note: WNDgetParent does NOT return parent windows if their parent window is HWND_MAINWINDOW. These windows are instantiated from Omnis window classes and are private to Omnis, no direct support is given to access these windows (see WNDgetOS). NULL is returned instead.

- **pHwnd** - identifies the window for which to return the parent window.
- **return** - returns the parent window.

Example:

See WM_NCLBUTTONDOWN, WNDupdateWindow.

See also WNDsetParent

WNDgetScrollPos()

```
void WNDgetScrollPos( HWND pHwnd, qshort pWhich, qdim* pPos )
```

Retrieves the current scroll position of the given window and scrollbar.

Note: Querying a vertical or horizontal header component returns the appropriate scroll position from the client component.

- **pHwnd** - identifies the window for which to return the scroll position.
- **pWhich** - identifies the scrollbar SB_VERT or SB_HORZ.
- **pPos** - points to the qdim which is to receive the scroll position.

Example:

See WM_HSCROLL.

See also WNDsetScrollPos, WNDsetScrollRange, WNDgetScrollRange

WNDgetScrollRange()

```
void WNDgetScrollRange( HWND pHwnd,
qshort pWhich, qdim* pMin, qdim* pMax,
qdim* pPage )
```

Retrieves the scroll range and page size of the given window and scrollbar.

Note: Querying a vertical or horizontal header component returns the appropriate scroll range and page size from the client component. The maximum range includes the page size as specified by WNDsetScrollRange. In order to find the true range of scroll positions you must subtract pPage from the pMax value.

- **pHwnd** - identifies the window for which to return the scroll range.
- **pWhich** - identifies the scrollbar SB_VERT or SB_HORZ.
- **pMin** - points to the qdim which is to receive the minimum scroll range.
- **pMax** - points to the qdim which is to receive the maximum scroll range (includes page size).
- **pPage** - points to the qdim which is to receive the page size.

Example:

See WM_HSCROLL, WM_WINDOWPOSCHANGED.

See also WNDsetScrollPos, WNDgetScrollPos, WNDsetScrollRange

WNDgetThemeColor()

```
qcol WNDgetThemeColor(qulong pType, qulong pFlags, qulong pPropId)
```

Returns the color of the specified attribute when using the specified theme and state information. Constants are defined in hwnd.he.

- **pType** – Type constant corresponding to the theme type required.
- **pFlags** – Flags representing state information about the control.
- **pPropId** – Constant representing the type of attribute required.

Example:

```
qcol textColor = textSpec().mTextColor;
if (WND_BORD_CTRL_GROUPBOX == border.mBorderStyle && gmain.isXP() && GDI_COLOR_WINDOWTEXT == textColor && !gma
{
  textColor = WNDgetThemeColor(THEME_GROUPBOX, THEME_CONTROL_NORMAL, THEME_COLOR_TEXTCOLOR);
  if (GDI_COLOR_QDEFAULT == textColor)
    textColor = GDI_COLOR_ACTIVECAPTION;
}
```

WNDgetThemeState()

qulong WNDgetThemeState(HWND hWnd)

Returns flags describing state information about the window's theme. State flags are defined in hwnd.he.

- **hWnd** – identifies the window. If NULL is passed, the main HWND is assumed.

WNDgetThemeControlSize()

qbool WNDgetThemeControlSize(HWND hWnd,HDC pHdc,qulong pType,qulong pFlags,qpoint* pSize)

Returns the size coordinates of the specified themed control. Types and flags are defined in hwnd.he

- **hWnd** – identifies the window on which the control resides. If NULL is passed, the main HWND is assumed.
- **pHdc** - identifies the drawing device.
- **pType** – Type constant corresponding to the theme type required.
- **pFlags** – Flags representing state information about the control.
- **pSize** - (output). A qpoint structure containing the size coordinates.

Example:

```
qpoint size;
if (WNDgetThemeControlSize(0, pHdc, THEME_STATUS, THEME_CONTROL_DEFAULT, &size))
{ //..}
```

WNDgetUpdateRgn()

void WNDgetUpdateRgn(HWND pHwnd, qrgn* pRgn)

Returns the update region of the given window. This function should only be called during WM_PAINT and WM_CHILDPAINT messages prior to calling WNDbeginPaint (calling WNDbeginPaint clears the update region of the window). Calling it at any other time may not return the correct region.

- **pHwnd** - identifies the window who's update region is to be retrieved.
- **pRgn** - points to the qrgn which is to receive the update region.

Example:

See WNDredrawChildren.

See also WNDbeginPaint, WM_PAINT, WM_CHILDPAINT

WNDgetWindow()

HWND WNDgetWindow(HWND pHwnd, UINT pRelationFlag)

Retrieves the related window of the given window.

- **pHwnd** - identifies the window for which to return the related window.
- **pRelationFlag** - identifies the relation. One of the following flags can be specified:
  GW_CHILD - returns the top most child window
  GW_HWNDFIRST - returns the top most sibling window
  GW_HWNDLAST - returns the bottom most sibling window
  GW_HWNDNEXT - returns the sibling window just below the given window
  GW_HWNDPREV - returns the sibling window just above the given window

Example:

```
// this example steps through all immediate children of the window.
HWND curChild = WNDgetWindow( myHwnd, GW_CHILD );
while ( curChild )
{
  curChild = WNDgetWindow( curChild, GW_HWNDNEXT );
}
```

See also WNDenumChildWindows, WNDgetWindowComponent, WNDnextWindowComponent, GW_xxx

WNDgetWindowComponent()

HWND WNDgetWindowComponent(HWND pHwnd, qulong pComponent)

Returns the specified component of the given window.

- **pHwnd** - identifies the window for which to return the component.
- **pComponent** - identifies the component to be returned. One of the WND_WC_xxx flags must be specified here.
- **return** - the HWND of the component.

Example:

```
// this example retrieves the client component of a window
HWND clientComp = WNDgetWindowComponent( myHwnd, WND_WC_CLIENT );
```

See also WNDgetWindow

WNDgetWindowCursor()

qshort WNDgetWindowCursor(HWND pHwnd)

Returns the cursor id which is associated with the given window.

- **pHwnd** - identifies the window for which to return the cursor id.
- **return** - the cursor id.

Example:

```
// this example changes the window's cursor if the current cursor
// is not set, that is, is equal to WND_CURS_DEFAULT.
if ( WNDgetWindowCursor( myHwnd ) == WND_CURS_DEFAULT )
{
  WNDsetWindowCursor( myHwnd, WND_CURS_NOGO );
}
```

See also WNDsetWindowCursor, WNDcheckCursor, WM_SETCURSOR

WNDgetWindowFromPt()

qbool WNDgetWindowFromPt(HWND* pHwnd, qword2* pHitTest, qpoint* pPoint)

Takes a global point local to HWND_DESKTOP and locates the window that is underneath the point.

- **pHwnd** - points to the HWND variable which is to contain the window which was found underneath the point.
- **pHitTest** - points to the variable which is to contain the window part which is underneath the point. The value is one of the HTxxx values.
- **pPoint** - points to the qpoint.
- **return** - returns qtrue if a window was found. Otherwise qfalse is returned.

Example:

```
// this example displays info about the hwnd the mouse is over if the user
// clicks in the client area of this window and than drags around the screen
// while holding down the mouse button
static HWND lastHwndUnder = NULL;
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_LBUTTONDOWN:
    {
      WNDsetCapture( hWnd, WND_CAPTURE_MOUSE );
      return 0L;
    }
    case WM_MOUSEMOVE:
    {
      if ( WNDhasCapture( hWnd, WND_CAPTURE_MOUSE ) )
      {
        qpoint pt; WNDmakePoint( lParam, &pt );
        HWND hwndUnder = NULL;
        qword2 hittest;
        str255 txt;
        WNDmapWindowPoint( hWnd, HWND_DESKTOP, pt );
        if ( WNDgetWindowFromPt( &hwndUnder, &hittest, pt ) )
        {
          qrect wRect; WNDgetWindowRect( hwndUnder, &wRect );
          str15 num;
          txt = str255("Left=$ ; Top=$ ; Right=$ ; Bottom=$");
          stri( wRect.left, num ); txt.insertStr( num );
          stri( wRect.top, num ); txt.insertStr( num );
```

```
          stri( wRect.right, num ); txt.insertStr( num );
          stri( wRect.bottom, num ); txt.insertStr( num );
        }
        if ( hwndUnder != sLastHwndUnder )
        {
          sLastHwndUnder = hwndUnder;
          HDC dc = WNDstartDraw( hWnd );
          qrect cRect; WNDgetClientRect( hWnd, &cRect );
          GDIsetTextColor( dc, GDI_COLOR_WINDOW );
          GDIfillRect( dc, &cRect, GDIgetStockBrush( BLACK_BRUSH ) );
          GDIsetTextColor( dc, GDI_COLOR_WINDOWTEXT );
          GDIdrawText( dc, 0, 0, &txt[1], txt[0], jstLeft );
          WNDendDraw( hWnd, dc );
        }
      }
      return 0L;
    }
    case WM_LBUTTONUP:
    {
      if ( WNDhasCapture( hWnd, WND_CAPTURE_MOUSE ) )
      {
        WNDreleaseCapture( WND_CAPTURE_MOUSE );
      }
      return 0L;
    }
  }
  return DefWindowProc( hwnd, message, wparam, lparam );
}
```

WNDgetWindowLong()

qulong WNDgetWindowLong(HWND pHwnd, qlong pOffset)

Retrieves style and type information about a window.

- **pHwnd** - identifies the window for which to return the information.
- **pOffset** - identifies the information to be returned. One of the following flags can be specified:
- **return** - the requested information.

Example:

```
// this example switches of the scrollbars of the window
qulong style = WNDgetWindowLong( myHwnd, GWL_STYLE );
style &= ~(WS_HSCROLL | WS_VSCROLL);
WNDsetWindowLong( myHwnd, GWL_STYLE, style );
```

See also WNDsetWindowLong, WNDcreateWindow, WS_xxx (styles), WND_xxx (extended styles), WND_WC_xxx (component ids)

WNDgetProcInst()

WNDprocClass* WNDgetProcInst(HWND pHwnd)

Returns a pointer to the WNDprocClass instance which is associated with the given window.

- **pHwnd** - identifies the window for which to return the associated WNDprocClass instance.
- **return** - returns a pointer to the WNDprocClass instance. **WARNING**: returns NULL if the window has no associated WNDprocClass instance.

Example:

```
// this example creates a number of child windows which have their own
// WNDprocClass instance. On a delete key message the parent window
// destroys all child windows.
class cChildWndProcClass: public WNDprocClass
{
  cChildWndProcClass() {}
  ~ cChildWndProcClass() {}
  virtual qlong WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam );
}

qlong cParentWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_CREATE:
    {
      // ** create the child windows **
      // first instantiate the WNDprocClass
      cChildWndProcClass* childWndProc = new cChildWndProcClass();
      // prepare for first child creation
      qrect childWRect( 10, 10, 100, 20 );
      WNDborderStruct childBorder( WND_BORD_INSET );
      // now create the first child window (we do not need to
      // remember the HWND)
      WNDcreateWindow
       (
          hWnd,
          WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN,
          0,
          childWndProc,
          &childWRect,
          &childBorder
       );
       // prepare for second child creation
       childWRect.top += 40;
       // now create the second child window
       WNDcreateWindow
       (
         hWnd,
         WS_CHILD | WS_CLIPSIBLINGS | WS_CLIPCHILDREN,
         0,
         childWndProc,
         &childWRect,
         &childBorder
       );
       return 0L;
      }
      case WM_KEY:
      {
        qkey* key = (qkey*)lParam;
        vchar vch = key->getVChar();
        if ( vch == vcBack || vch == vcClear )
        {
```

```
          // ** delete the child windows **
          HWND curChild = WNDgetWindow( hWnd, GW_CHILD );
          while ( curChild )
          {
            // first get the WNDprocClass of the child
            // first get the WNDprocClass of the child
            cChildWndProcClass* childWndProc = (cChildWndProcClass*) WNDgetProcInst( curChild );
            // first get the WNDprocClass of the child
            // delete the childWndProc, but set the WNDprocClass in
            // the window to NULL first
            WNDsetProcInst( curChild, NULL );
            delete childWndProc;
            // destroy the window
            WNDdestroyWindow( curChild );
            // get the next child, always start at the top again
            curChild = WNDgetWindow( hWnd, GW_CHILD );
          }
          return 0L;
        }
        return 1L;
      }
    }
  }
  return ( DefWindowProc( hWnd, message, wParam, lParam ) );
}
```

See also WNDprocClass, WNDsetProcInst, WNDcreateWindow, WNDaddWindowComponent

WNDgetWindowRect()

```
void WNDgetWindowRect( HWND pHwnd, qrect* pRect )
```

Retrieves the global coordinates (local to HWND_DESKTOP) of the window.

- **pHwnd** - identifies the window for which to return the rect.
- **pRect** - points to the qrect structure which is to receive the coordinates.

Example:

See WNDredrawChildren.

See also WNDgetClientRect

WNDhasCapture()

```
qbool WNDhasCapture( HWND pHwnd, qulong pFlags )
```

Returns qtrue if the given window has the specified capture.

- **pHwnd** - identifies the window to test for the specified capture.
- **pFlags** - specifies the capture for which to test the given window. This parameter can be WND_CAPTURE_MOUSE or WND_CAPTURE_KEY. Only one of the two flags must be specified.
- **return** - returns qtrue if the given window has the specified capture.

Example:

See WM_LBUTTONxxx, WNDgetCapture, WNDgetWindowFromPt.

See also WNDgetCapture, WNDsetCapture, WNDreleaseCapture, WND_CAPTURE_xxx

WNDhideCaret()

void WNDhideCaret()

Hides the system caret if it is currently visible, and increments the caret's hidden count. If this function is called more than once before calling WNDshowCaret, it takes the same number of calls to WNDshowCaret, to make the caret visible again.

Example:

```
WNDhideCaret();
```

See also WNDcreateCaret, WNDdestroyCaret, WNDgetCaretPos, WNDsetCaretPos, WNDshowCaret, WM_FOCUSCHANGED

WNDinflateBorderRect()

void WNDinflateBorderRect(HWND pHwnd, qrect* pRect, WNDborderStruct* pBorderSpec)

This function is the reverse of WNDinsetBorderRect. Inflates the supplied rectangle by the left, top, right, and bottom by the amount which is required for the specified border information (the amount which the border requires to paint). For example, if the border was of type WND_BORD_INSET the rectangle would be inflated by two pixels on all sides.

For custom borders (WND_BORD_CUSTOM) the HWND module sends a WM_BORDCALCRECT message to the WndProc function of the given window.

- **pHwnd** - identifies the window to be called for custom borders.
- **pRect** - points to the qrect to be inflated.
- **pBorderSpec** - points to the border information.

Example:

See WNDpaintBorder.

See also WNDborderStruct, WNDdrawBorder, WNDinsetBorderRect

WNDinsetBorderRect()

void WNDinsetBorderRect(HWND pHwnd, qrect* pRect, WNDborderStruct* pBorderSpec)

Reverse of WNDinflateBorderRect.

Insets the supplied rectangle by the left, top, right, and bottom by the amount which is required for the specified border information (the amount which the border requires to paint). For example, if the border was of type WND_BORD_INSET the rectangle would be inset by two pixels on all sides.

For custom borders (WND_BORD_CUSTOM) the HWND module sends a WM_BORDCALCRECT message to the WndProc function of the given window.

- **pHwnd** - identifies the window to be called for custom borders.
- **pRect** - points to the qrect to be inset.
- **pBorderSpec** - points to the border information.

Example

See WNDpaintBorder.

See also WNDborderStruct, WNDdrawBorder, WNDinflateBorderRect

WNDinvalidateFrame()

void WNDinvalidateFrame(HWND pHwnd)

Adds the non-client area of the given window to the windows update region. WM_NCPAINT messages are generated as a result of this call.

- **pHwnd** - identifies the window to be invalidated.

Example:

```
WNDinvalidateFrame( myHwnd );
```

See also WNDinvalidateRect, WNDinvalidateRgn

WNDinvalidateRect()

void WNDinvalidateRect(HWND pHwnd, qrect *pRect)

Adds the given rectangular area within the client area of the given window to the windows update region. WM_PAINT messages are generated as a result of this call.

- **pHwnd** - identifies the window to be invalidated.
- **pRect** - points to the qrect to be invalidated inside the client area. If this parameter is NULL, the whole client area is invalidated.

Example:

See WM_SHOWSIZEGRIP.

See also WNDinvalidateRgn, WNDinvalidateFrame

WNDinvalidateRgn()

void WNDinvalidateRgn(HWND pHwnd, qrgn *pRgn)

Adds the given region within the client area of the given window to the windows update region. WM_PAINT messages are generated as a result of this call.

- **pHwnd** - identifies the window to be invalidated.
- **pRgn** - points to the qrgn to be invalidated inside the client area. If this parameter is NULL, the whole client area is invalidated.

Example:

```
// this example invalidates to rectangular areas in the client area of the
// window in one call to WNDinvalidateRgn
qrgn rgn1, rgn2;
GDIsetRectRgn( &rgn1, 10, 10, 50, 20 );
GDIsetRectRgn( &rgn2, 10, 50, 80, 60 );
GDIrgnOr( &rgn1, &rgn1, &rgn2 );
WNDinvalidateRgn( myHwnd, &rgn1 );
```

See also WNDinvalidateRect, WNDinvalidateFrame

WNDisBorderExternal() (v3.1)

qbool WNDisBorderExternal(HWND pHwnd, qshort pBorderStyle)

This function checks if the given border style will be drawn outside the HWNDs frame. This is only true for some borders on Mac OSX.

- **pHwnd** - identifies the window.
- **pBorderStyle** - identifies the border style to test
- **returns** - true if the border will be drawn outside the windows frame.

WNDisPaintInProgress()

qbool WNDisPaintInProgress()

Returns qtrue if a paint is currently in progress, qfalse otherwise.

Example:

```
if (mHWnd)
{
  setPicturesScrollRange();
  if (!WNDisPaintInProgress())
  {
    WNDinvalidateRect(mHWnd,NULL);
    WNDupdateWindow(mHWnd);
  }
}
```

WNDisVistaTheme()

qulong WNDisVistaTheme()

Returns THEME_STATE_xxx flags describing the Windows Vista theme being used by the operating system. This depends whether the component is running under Vista and whether Vista is running in Classic mode or not. Currently, if Windows Vista is using themes, WNDisVistaTheme() returns the THEME_STATE_ACTIVE and THEME_STATE_HOTACTIVE flags. Otherwise, THEME_STATE_NOTACTIVE is returned.

Example:

```
mIsVista = (qbool) (WNDisVistaTheme() != 0);
```

WNDisWindowVisible()

qbool WNDisWindowVisible(HWND pHwnd)

Returns the current visibility state of the given window. It only returns qtrue if the window is visible on screen (the WS_VISIBLE flag is set for it self and all of the windows parents) although it may be hidden by overlapping sibling windows. If it is required to test the WS_VISIBLE style of a window, use the WNDgetWindowLong function.

- **pHwnd** - identifies the window to be tested.
- **return** - returns qtrue if the window is visible.

Example:

```
// in this example, the window redraws itself if it is truly visible, that is, all of its
// parents are also visible.
if ( WNDisWindowVisible( myHwnd ) )
{
  WNDredrawWindow( myHwnd, NULL, NULL, WND_RW_NCPAINT | WND_RW_PAINT | WND_RW_ERASE );
}
```

See also WS_VISIBLE, WNDgetWindowLong

WNDkillTimer()

qbool WNDkillTimer(HWND pHwnd, qushort pTimerId)

Removes the timer of the specified id from the given window.

- **pHwnd** - identifies the window who owns the timer.
- **pTimerId** - specifies the id of the timer to be removed.
- **return** - returns qtrue if the timer was removed successfully. If a timer of the given id could not be found, qfalse is returned.

Example:

See WM_TIMER.

See also WNDsetTimer, WM_TIMER

WNDmakeLong()

qlong WNDmakeLong(qpoint* pPoint)

Converts a qpoint and returns it as a long value. This function should be used when it is required to send a point in the lParam parameter of a message.

- **pPoint** - points to the qpoint to be converted.
- **return** - returns the long value of the point.

Example:

```
// this example sends a left mousedown and mouseup to a window
qpoint pt(10,10);
WNDsendMessage( myHwnd, WM_LBUTTONDOWN, 0, WNDmakeLong( &pt ) );
WNDsendMessage( myHwnd, WM_LBUTTONUP, 0, WNDmakeLong( &pt ) );
```

See also WNDmakePoint

WNDmakeLong() (v3.1)

qlong WNDmakeLong(qrect* pRect)

Converts a qrect and returns it as a long value. This function is used when setting the GWL_INFLATE_ALL and GWL_INFLATE_FRAME values of a HWND.

- **pRect** - points to the qrect to be converted.
- **return** - returns the long value of the rect.

See also WNDmakeRect, GWL_INFLATE_ALL, GWL_INFLATE_FRAME

WNDmakeEnumWindowsProc ()

```
FARPROC WNDmakeEnumWin-
dowsProc(WNDenumProc pEnumProc,
HINSTANCE pInstance)
```

Returns a FARPROC which then can be passed to WNDenumChildWindows.

- **pEnumProc** - The enumerate procedure.
- **pInstance** - Instance of the component.

WNDmakePoint()

```
void WNDmakePoint( qlong pLongValue, qpoint* pPoint )
```

Takes a long value and converts it into a point which is returned in the pPoint parameter.

- **pLongValue** - specifies the long value to be converted to a point.
- **pPoint** - points to the qpoint structure which is to store the converted long value.

Example:

See WM_LBUTTONxxx, WM_NCLBUTTONDOWN, WNDgetWindowFromPt.

See also WNDmakeLong

WNDmakeRect() (v3.1)

```
void WNDmakeRect( qlong pLongValue, qrect* pRect )
```

Takes a long value and converts it into a rect which is returned in the pRect parameter. This function is used when getting the GWL_INFLATE_ALL and GWL_INFLATE_FRAME values of a HWND.

- **pLongValue** - specifies the long value to be converted to a rect.
- **pRect** - points to the qrect structure which is to store the converted long value.

See also WNDmakeLong, GWL_INFLATE_ALL, GWL_INFLATE_FRAME

WNDmapWindowPoint()

```
void WNDmapWindowPoint( HWND
pHwndFrom, HWND pHwndTo, qpoint*
pPoint )
```

Converts or maps a point from a coordinate space relative to one window, to a coordinate space relative to another window.

- **pHwndFrom** - identifies the window from which the point is converted. If this parameter is HWND_DESKTOP, the point is assumed to be in screen coordinates (under MacOS: local to the combined union of all monitors). If it is HWND_MAINWINDOW the point is assumed to be in coordinates local to the Omnis program window (under MacOS: in screen coordinates local to the main monitor).

- **pHwndTo** - identifies the window to which the point is converted. If this parameter is HWND_DESKTOP, the point is converted to screen coordinates (under MacOS: local to the combined union of all monitors). If it is HWND_MAINWINDOW the point is converted to the Omnis program window (under MacOS: to screen coordinates local to the main monitor).
- **pPoint** - points to the qpoint to be converted.

Example:

See WNDgetWindowFromPt.

See also WNDmapWindowRect

WNDmapWindowRect()

```
void WNDmapWindowRect( HWND pHwndFrom, HWND pHwndTo, qrect* pRect )
```

The same as WNDmapWindowPoint except that it converts a qrect.

Example:

See WM_NCLBUTTONDOWN, WM_SETCURSOR, WNDredrawChildren, WNDsetCursorPos, WNDsetParent.

See also WNDmapWindowPoint

WNDmouseLeftButtonDown()

```
qbool WNDmouseLeftButtonDown()
```

Returns qtrue if the logical (not physical) left mouse button is held down at the time of the call to this function.

Example:

See WM_NCLBUTTONDOWN, WM_TIMER.

See also WNDmouseTrackLeftButton, WNDmouseRightButtonDown

WNDmouseRightButtonDown()

```
qbool WNDmouseRightButtonDown()
```

Returns qtrue if the logical (not physical) right mouse button is held down at the time of the call to this function.

Example:

```
// this example holds program execution while the right mouse button is held down
while ( WNDmouseRightButtonDown() ) ;
```

See also WNDmouseLeftButtonDown

WNDmouseTrackLeftButton() (v3.1)

qbool WNDmouseTrackLeftButton()

This function should be used to fix tight loop mouse tracking. The traditional way of sitting in a while loop while the left mouse button is down has adverse effects on some platforms like Mac OSX.

The while loop should now call WNDmouseTrackLeftButton which will free processor time while the mouse is not doing anything of interest. The function only returns to the caller, when the mouse has moved, or the button states have changed.

- **returns** – qtrue if the left mouse button is down.

Example:

```
while ( WNDmouseTrackLeftButton() )
{
  // the button is still down and the mouse has moved
  // do your stuff
}
```

See also WNDmouseLeftButtonDown

WNDmoveWindow()

qbool WNDmoveWindow(HWND pHwnd, qdim pLeft, qdim pTop, qdim pWidth, qdim pHeight, qbool pRepaint)

Changes the position and size of the given window (the window's rectangle) inside its parent window. The given coordinates must be local to the parent's client area. WM_WINDOWPOSCHANGING and WM_WINDOWPOSCHANGED messages are sent as a result of this call.

- **pHwnd** - identifies the window to be moved.
- **pLeft** - the new position of the left side of the window.
- **pTop** - the new position of the top side of the window.
- **pWidth** - the new width of the window.
- **pHeight** - the new height of the window.
- **pRepaint** - if this parameter is qtrue, all appropriate areas of the window, the parent window, and overlapped sibling windows, are invalidated. If it is qfalse, no painting occurs as a result of this call.
- **return** - returns qtrue if successful.

Example:

See WM_NCLBUTTONDOWN, WNDsetParent, WNDupdateWindow.

See also WNDgetWindowRect, WNDsetWindowPos

WNDnextWindowComponent()

```
HWND WNDnextWindowComponent(
HWND pHwnd, HWND pComponentHwnd,
qulong pComponent )
```

Returns the next component of the specified component(s) type. It is possible to next on more than one component type by specifying more than one component type in the pComponent flag.

- **pHwnd** - identifies the parent window for which to return the component(s).
- **pComponentHwnd** - identifies the current component. If it is NULL, the first component of the specified component type(s) is returned, otherwise the next matching component is returned.
- **pComponent** - one or more of the WND_WC_xxx flags must be specified here. Any component types which are not specified are ignored.
- **return** - the HWND of the found component. If no more components can be found, NULL is returned.

Example:

```
// this example steps through all components of a window with the exception of the
// client component
HWND curComp = NULL;
qulong ids = WND_WC_MASK - WND_WC_CLIENT;
while ( curComp = WNDnextWindowComponent( myHwnd, curComp, ids ) )
{
  // do something
}
```

See also WNDaddWindowComponent, WNDnextWindowComponent, WNDremoveWindowComponent, WNDgetWindow, WND_WC_xxx

WNDpaintBorder()

```
void WNDpaintBorder( HWND pHwnd, HDC
pHdc, qrect* pRect, WNDborderStruct*
pBorderSpec )
```

Draws the given border style inside the given rectangle.

- **pHwnd** - identifies the HWND which receives any WM_GETERASEINFO or WM_BORDPAINT (custom borders) messages.
- **pHdc** - identifies the drawing device.
- **pRect** - points to the qrect in which the border is painted.
- **pBorderSpec** - points to the border style structure.

Example:

```
// this example paints a border inside the client area of a
// window and fills the remaining area
WNDborderStruct border( WND_BORD_BEVEL, 2, 4, 2 );
qrect cRect; WNDgetClientRect( myHwnd, &cRect );
HDC dc = WNDstartDraw( myHwnd );
WNDpaintBorder( NULL, dc, &cRect, &border );
WNDinsetBorderRect( NULL, &cRect, &border );
GDIsetTextColor( dc, GDI_COLOR_3DFACE );
GDIfillRect( dc, &cRect, GDIgetStockBrush( BLACK_BRUSH ) );
WNDendDraw( myHwnd, dc );
```

See also WNDborderStruct, WNDinflateBorderRect, WNDinsetBorderRect, WNDgetBorderSpec, WNDsetBorderSpec

WNDpostMessage() (v3.1)

qlong WNDpostMessage(HWND pHwnd, UINT message, WPARAM wParam, LPARAM lParam)

Posts the given message and parameters to the WndProc function associated with the given window. The message is placed in the systems event queue, and not executed immediately.

- **pHwnd** - identifies the window to send the message to.
- **message** - specifies the message to be sent (WM_xxx).
- **wParam** - specifies the wParam parameter to be sent to the windows procedure.
- **lParam** - specifies the lParam parameter to be sent to the windows procedure.

See also WNDsendMessage

WNDredrawChildren()

void WNDredrawChildren(HWND pHwnd, qrgn* pRgn)

Adds the intersection of the given region and each child's visual region to the child's update region. It generates a WM_CHILDPAINT message for the given parent window for each child which requires updating. All child windows are included, no matter how deep the child windows are nested.

The WNDredrawChildren function is specifically designed to allow complex controls, to manage the painting of their child controls. A complex control may want to display the same control in more than one location, displaying different data. The complex control can achieve this by always keeping the child windows invisible, and upon receiving a paint message for itself, the update region can be retrieved to paint the children.

- **pHwnd** - identifies the window who's children are to be redrawn.
- **pRgn** - identifies the region in the parent for which intersecting children are to be redrawn.

Example:

```
// in this example a window controls the painting of its child windows:
static HWND sUpdHwnd;
static qrgn sUpdRgn;
qlong cMyWndProcClass::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
  case WM_PAINT:
  {
    WNDpaintStruct paintStruct;
    // retrieve the update region.
    // Note: region is local to hWnd
    WNDgetUpdateRgn( hWnd, &sUpdRgn );
    // clear the update region of this window
    WNDbeginPaint( hWnd, &paintStruct );
    WNDendPaint( hWnd, &paintStruct );
    // now paint the children. Set sUpdHwnd which is used
```

```
    // during WM_CHILDPAINT.
    sUpdHwnd = hWnd;
    WNDredrawChildren( hWnd, &sUpdRgn );
    // now fill in the remaining region with the background
    // and return
    HDC dc = WNDstartDraw( hWnd );
    GDIsetTextColor( dc, GDI_COLOR_3DFACE );
    GDIfillRgn( dc, &sUpdRgn, GDIgetStockBrush( BLACK_BRUSH ) );
    WNDendDraw( hWnd, dc );
    return 0L;
  }
  case WM_CHILDPAINT:
  {
    if ( lParam & WND_RW_PAINT )
    {
      // if lParam contains WND_RW_PAINT the child's client area needs painting
      // Note: hWnd parameter now points to child window.
      // do any work required before painting the child, that is,
      // prepare the child's data
      // make the child visible but do NOT course any further
      // redraws by specifying SWP_NOREDRAW. The same call can be
      // used to also move the child to a different location.
      WNDsetWindowPos( hWnd, NULL, 0, 0, 0, 0,
SWP_SHOWWINOW|SWP_NOREDRAW|SWP_NOSIZE|SWP_NOMOVE|SWP_NOZORDER );
      // now paint the child
      WNDsendMessage( hWnd , WM_PAINT, 0, 0 );
      // hide the child again
      WNDsetWindowPos( hWnd, NULL, 0, 0, 0, 0, SWP_HIDEWINOW|SWP_NOREDRAW|SWP_NOSIZE|SWP_NOMOVE|SWP_NOZORDER )
      // subtract the child's rect from the static update region so
      // you can fill in the remaining area with the background of
      // the complex control when WNDredrawChildren returns.
      qrect wRect; qrgn wRgn;
      WNDgetWindowRect( hWnd, &wRect );
      WNDmapWindowRect( HWND_DESKTOP, sUpdHwnd, &wRect );
      GDIsetRectRgn( &wRgn, &wRect );
      GDIrgnDiff( &sUpdRgn, &sUpdRgn, &wRgn );
      // return zero so the window manager takes no further action
      return 0L;
    }
    else
    {
      // only the non-client area needs painting
      // return 1 to tell window manager to go ahead and paint the child normally
      return 1L;
    }
  }
}
return ( DefWindowProc( hWnd, message, wParam, lParam ) );
```

See also WM_PAINT

WNDredrawWindow()

```
void WNDredrawWindow( HWND pHwnd,
qrect* pRect, qrgn* pRgn, qulong pFlags)
```

Redraws, invalidates or updates the given window.

- **pHwnd** - identifies the window for the redraw action.
- **pRect** - points to the rectangle to be redrawn or invalidated in the client area of the window. This parameter is only used if WND_RW_PAINT or WND_RW_INVALIDATE is specified, and is ignored if WND_RW_UPDATE is specified. If this parameter is NULL and pRgn is NULL, the entire client area is included in the redraw action if WND_RW_PAINT or WND_RW_INVALIDATE is specified.
- **pRgn** - points to the region to be redrawn or invalidated in the client area of the window. This parameter is only used if WND_RW_PAINT or WND_RW_INVALIDATE is specified, and is ignored if WND_RW_UPDATE is specified. If this parameter is NULL and pRect is NULL, the entire client area is included in the redraw action if WND_RW_PAINT or WND_RW_INVALIDATE is specified.
- **pFlags** - One or more redraw flags. This parameter can be a combination of flags:
  WND_RW_NCPAINT -if specified the non-client area is included in the redraw action.
  WND_RW_PAINT -if specified client area is included in the redraw action.
  WND_RW_ERASE -if specified WM_ERASEBKGND messages are generated as part of the redraw action.
  WND_RW_INVALIDATE -if specified, the window is not redrawn immediately, but the specified areas are invalidated instead.
  WND_RW_UPDATE -if specified the window is updated. All above flags are ignored.
  WND_RW_ALLCHILDREN -if specified with any of the above flags, all children of the window are included in the redraw action.

Example:

See WM_NCLBUTTONDOWN, WNDisWindowVisible, WNDsetParent.

See also WNDupdateWindow, WNDinvalidateRect, WNDinvalidateRgn, WNDinvalidateFrame, WNDredrawWindowCO

WNDredrawWindowCO()

void WNDredrawWindowCO(HWND pHwnd, qrect* pRect, qrgn* pRgn, qulong pFlags, qulong pComponents)

Calls WNDredrawWindow for all specified components of the given window.

- **pHwnd** - identifies the parent window of the components to be redrawn.
- **pRect** - see WNDredrawWindow.
- **pRgn** - see WNDredrawWindow.
- **pFlags** - see WNDredrawWindow.
- **pComponents** - one ore more of the WND_WC_xxx flags must be specified here.

Example:

```
// this example redraws all components of a window with the exception of the
// client component
qulong ids = WND_WC_MASK - WND_WC_CLIENT;
qulong flags = WND_RW_NCPAINT | WND_RW_PAINT | WND_RW_ERASE;
WNDredrawWindowCO( myHwnd, NULL, NULL, flags, ids );
```

See also WNDredrawWindow, WND_WC_xxx (component ids)

WNDreleaseCapture()

void WNDreleaseCapture(qulong pFlags)

Releases the specified capture which was set previously by WNDsetCapture.

- **pFlags** - Specifies which capture is to be released. The values WND_CAPTURE_MOUSE or WND_CAPTURE_KEY can be used with this parameter. Both captures can be specified in the same call. It is not necessary for external components to release the key capture. Omnis releases the key capture for a window when it looses the input focus.

Example:

See WNDgetWindowFromPt, WM_LBUTTONxxx.

See also WNDgetCapture, WNDhasCapture, WNDsetCapture

WNDremoveWindowComponents()

void WNDremoveWindowComponents(HWND pHwnd, qulong pComponent)

Removes the specified component(s) from the given window. Removing components may cause the position of other components to be altered, and generates WM_PAINT messages if these components are visible. The removed component windows are destroyed before the function returns.

- **pHwnd** - identifies the parent window from which to remove the components.
- **pComponent** - one ore more of the WND_WC_xxx flags must be specified here.

Example:

```
// this example removes (destroys) all components of a window with the exception
// of the client component
WNDremoveWindowComponents( myHwnd, WND_WC_MASK - WND_WC_CLIENT );
```

See also WNDaddWindowComponent, WNDnextWindowComponent, WNDcreateWindow, WNDdestroyWindow, WND_WC_xxx (component ids)

WNDscrollWindow()

void WNDscrollWindow(HWND pHwnd, qdim pXOffset, qdim pYOffset)

Scrolls the contents of a windows client area. The positions of any child windows are offset by the amount specified by the pXOffset and pYOffset parameters (NO WM_WINDOWPOSCHANGED messages are generated). The HWND module preserves as much of the client area as it can, and only invalidates the uncovered area in the window. WM_PAINT messages are eventually sent to the window and all child windows which intersect the scrolled in area.

- **pHwnd** - identifies the window to be scrolled.
- **pXOffset** - specifies the amount, in device units, of horizontal scrolling. This parameter must be a negative value to scroll to the left.
- **pYOffset** - specifies the amount, in device units, of vertical scrolling. This parameter must be a negative value to scroll up.

Example:

See WM_HSCROLL.

See also WM_HSCROLL, WM_VSCROLL

WNDsendMessage()

qlong WNDsendMessage(HWND pHwnd, UINT message, WPARAM wParam, LPARAM lParam)

Sends the given message and parameters to the WndProc function associated with the given window.

- **pHwnd** - identifies the window to send the message to.
- **message** - specifies the message to be sent (WM_xxx).
- **wParam** - specifies the wParam parameter to be sent to the windows procedure.
- **lParam** - specifies the lParam parameter to be sent to the windows procedure.

Example:

See WM_KEYxxx, WNDenumChildWindows, WNDredrawChildren, WNDmakeLong.

See also WNDpostMessage

WNDsetBorderSpec()

void WNDsetBorderSpec(HWND pHwnd, WNDborderSpec* pBorderSpec, qbool pRedraw)

Changes the border style of the given window. If as a result of this call the client area of the window changes size, all floating children are floated, and all components are sized accordingly. Changes to the border will result in the following messages being send; WM_BORDERCHANGING, WM_WINDOWPOSCHANGING, WM_WINDOWPOSCHANGED and WM_BORDERCHANGED.

- **pHwnd** - identifies the window which is to receive the new border style.
- **pBorderSpec** - The new border information.
- **pRedraw** - if qfalse, the window is not updated and must be redrawn manually.

Example:

See WNDgetBorderSpec.

See also WNDgetBorderSpec, WM_BORDERCHANGING, WM_WINDOWPOSCHANGING, WM_WINDOWPOSCHANGED, WM_BORDERCHANG

WNDsetCanSetCursorProc()

WNDcanSetCursorProc WNDsetCanSetCursorProc(WNDcanSetCursorProc pNewProc)

Assigns the address of a procedure to be called when the window receives a WM_SETCURSOR message. WNDcanSetCursorProc is defined as:

```
typedef qbool (*WNDcanSetCursorProc)();
```

The supplied proc pointer is stored and the existing proc pointer is returned by this call.

- pNewProc – The address of the new procedure that will determine whether the window cursor procedure can be changed.

See also WNDsetCurosr()

WNDsetCapture()

```
void WNDsetCapture( HWND pHwnd, qulong pFlags )
```

Sets the mouse or key capture to the given window. With the mouse or key capture set, all mouse and key input is directed to that window, regardless of whether the cursor is over that window or the window has the focus. Only one window can have the mouse capture or key capture at any one time.

- **pHwnd** - identifies the window that is to receive all mouse or key messages.
- **pFlags** - specifies which capture to set. The values WND_CAPTURE_MOUSE and WND_CAPTURE_KEY can be used with this parameter. Both captures can be specified in the same call. It is not necessary for external components to capture the key events. Omnis sets the key capture for a window when it receives the input focus.

Example:

See WNDgetWindowFromPt, WM_LBUTTONxxx.

See also WNDgetCapture, WNDhasCapture, WNDreleaseCapture, WND_CAPTURE_xxx

WNDsetCaretPos()

```
void WNDsetCaretPos( qpoint* pPos )
```

Moves the system caret to a new position inside the window associated with the caret.

- **pPos** - points to qpoint which specifies the new position for the caret.

Example:

See WM_FOCUSCHANGED, WNDgetCaretPos.

See also WNDcreateCaret, WNDdestroyCaret, WNDgetCaretPos, WNDhideCaret, WNDshowCaret, WM_FOCUSCHANGED

WNDsetCursor()

```
qshort WNDsetCursor( qshort pCursor )
```

Changes the current mouse cursor. This function should only be used to set a cursor during a mouse capture. During some operations it may be required to permanently set the cursor to a watch cursor. To prevent windows changing the cursor when it moves over them, the mouse capture can be set to HWND_MAINWINDOW or any other valid window. This prevents any WM_SETCURSOR messages being generated.

- **pCursor** - Identifies the new screen cursor. Can be any of the WND_CURS_xxx values.
- **return** - returns the previous cursor id.

Example:

See WNDgetCursor.

See also WNDgetCursor, WNDgetCursorPos, WNDsetCursorPos, WNDclipCursor, WNDgetWindowCursor, WNDsetWindowCursor

WNDsetCursorPos()

void WNDsetCursorPos(qpoint* pPoint)

Sets the location of the cursors Hotpoint on screen.

- **pPoint** - points to the qpoint structure containing the cursors new location in screen coordinates (local to HWND_DESKTOP).

Example:

```
// this example moves the screen cursor to the top left corner of a windows client area.
qrect cRect;
WNDgetClientRect( myHwnd, &cRect );
WNDmapWindowRect( myHwnd, HWND_DESKTOP, &cRect );
qpoint pt( cRect.left, cRect.top );
WNDsetCursorPos( &pt );
```

See also WNDgetCursorPos, WNDclipCursor, WNDgetCursor, WNDsetCursor, WNDgetWindowCursor, WNDsetWindowCursor

WNDsetFloat()

void WNDsetFloat(HWND pHwnd, qulong pFloat)

Sets the floating property of the given window.

- **pHwnd** - identifies the window who's floating properties are to be changed.
- **pFloat** - specifies the new floating properties. This value can be one or a combination of the WND_FLOAT_xxx flags.

Example:

See WNDgetFloat.

See also WNDgetFloat, WNDcreateWindow, WND_FLOAT_xxx

WNDsetParent()

HWND WNDsetParent(HWND pHwnd, HWND pNewParent)

Changes the parent window of the given child window. Changing the parent changes the window's position in the Z-order.

- **pHwnd** - identifies the window who's parent is to be changed.
- **pNewParent** - identifies the new parent window.
- **return** - returns the previous parent if the function was successful.

Example:

```
// this example moves the child windows of one window to another,
// maintaining the coordinates local to the parents.
// first disable all redraws for the two parents, we want to redraw
// everything at the end in one go.
WNDsetRedraw( oldParent, qfalse );
WNDsetRedraw( newParent, qfalse );
HWND curChild = WNDgetWindow( oldParent, GW_CHILD );
while ( curChild )
{
  // remember the child's window rect and make it local to its old parent
  qrect wRect;
  WNDgetWindowRect( curChild, &wRect );
  WNDmapWindowRect( HWND_DESKTOP, oldParent, &wRect );
  // disable redraws for the child while we change the parent and position
  WNDsetRedraw( curChild, qfalse );
  WNDsetParent( curChild, newParent );
  WNDmoveWindow( curChild, wRect.left, wRect.top,
wRect.width(), wRect.height(), qfalse );
  WNDsetRedraw( curChild, qtrue );
  // now get the next child. Always start from top again
  curChild = WNDgetWindow( oldParent, GW_CHILD );
}
// now redraw everything
WNDsetRedraw( oldParent, qtrue );
WNDsetRedraw( newParent, qtrue );
WNDredrawWindow( oldParent, NULL, NULL, WND_RW_PAINT | WND_RW_ERASE );
WNDredrawWindow( newParent, NULL, NULL, WND_RW_PAINT | WND_RW_ERASE | WND_RW_ALLCHILDREN );
```

See also WNDgetWindow, WNDsetWindowPos

WNDsetProcInst()

```
void WNDsetProcInst( HWND pHwnd, WNDprocClass* pWndProc )
```

Changes the WNDprocClass instance which is associated with the given window. The new instance receives all messages for the window from then on.

- **pHwnd** - identifies the window for which to change the WNDprocClass instance.
- **pWndProc** - points to the new WNDprocClass instance.

Example:

See WNDgetProcInst.

See also WNDprocClass, WNDgetProcInst, WNDcreateWindow, WNDaddWindowComponent

WNDsetRedraw()

```
void WNDsetRedraw( HWND pHwnd, qbool pRedraw )
```

Enables or disables the painting of the given window. If painting is disabled, any changes made to a window by any of the window management functions do **not** generate any WM_PAINT messages. In order for changes made to a window while the redraw flag is qfalse to be redrawn, call WNDinvalidateRect, WNDinvalidateRgn or WNDredrawWindow after enabling the redraw for the window.

- **pHwnd** - identifies the window for which to set the redraw flag.

- **pRedraw** - specifies the value for the redraw flag. If qtrue redraws for the window are enabled, otherwise they are disabled.

Example:

See WNDsetParent.

See also WNDinvalidateRect, WNDinvalidateRgn, WNDredrawWindow

WNDsetScrollPos()

```
void WNDsetScrollPos( HWND pHwnd,
qshort pWhich, qdim pPos, qbool pRedraw )
```

Sets the position of the scrollbar thumb.

- **pHwnd** - identifies the window whose scrollbar is to be set.
- **pWhich** - specifies which scrollbar is to be set. This parameter can be one of the following:
  SB_HORZ - horizontal scrollbar
  SB_VERT - vertical scrollbar
- **pPos** - specifies the new scroll position.
- **pRedraw** - if qtrue the scrollbar is redrawn to show the new position.

Example:

See WM_HSCROLL.

See also WNDgetScrollPos, WNDsetScrollRange, WNDgetScrollRange

WNDsetScrollRange()

```
void WNDsetScrollRange( HWND pHwnd,
qshort pWhich, qdim pMin, qdim pMax,
qdim pPage, qbool pRedraw )
```

Sets the minimum and maximum scroll range and the page size of the windows scroll bar. The page size should be included in the scroll range. Example: A list box contains 50 lines of data. The list box client area can display 10 complete lines. The minimum range should be specified as 1, the maximum range as 50, and the page size should be specified as 10. The HWND module restricts the maximum scroll range automatically to 40.

- **pHwnd** - identifies the window whose scroll range is to be set.
- **pWhich** - specifies which scrollbar is to be set. This parameter can be one of the following:
- **pMin** - specifies the minimum scroll range.
- **pMax** - specifies the maximum scroll range.
- **pPage** - specifies the page size (this should be the number of visible scroll units).
- **pRedraw** - if qtrue the scrollbar is redrawn to reflect the new range.

Example:

See WM_WINDOWPOSCHANGED.

See also WNDsetScrollPos, WNDgetScrollPos, WNDgetScrollRange

WNDsetTimer()

qbool WNDsetTimer(HWND pHwnd, qushort pTimerId, qushort pMillisecondDuration)

Installs a system timer with a duration of the specified number of milliseconds. Every time the specified duration expires, a WM_TIMER message is sent to the WndProc instance of the given window.

Note: Because WM_TIMER messages are only generated if no other messages are on the message queue, the accuracy of the intervals at which they are generated cannot be guaranteed, that is, while Omnis is busy, no WM_TIMER messages are generated.

Warning: On WIN16 timers are a scarce resource, and once they are used up, no more timers can be installed.

- **pHwnd** - identifies the window which owns the timer and receives the WM_TIMER messages.
- **pTimerId** - these must be unique for timers associated with the same window if an existing timer for the window is to remain. If a new timer has the same id as an existing one, the old timer duration is replaced. The timer id given must be WND_TIMER_FIRST + n, where n is a value in the range 0 to 32000.
- **pMillisecondDuration** - specifies the duration in milliseconds.
- **return** - returns qtrue if the timer was installed successfully.

Example:

See WM_TIMER.

See also WNDkillTimer, WM_TIMER

WNDsetTimerAttributesOSX()

void WNDsetTimerAttributesOSX(HWND pHwnd, UINT pTimerId, qulong pAttributes)

MacOSX only. Sets additional attributes associated with a timer. Currently accepts the value: HWND_TIMER_RUNS_WHEN_EVENTS_ARE_BLOC
defined in hwnd.he.

- **pHwnd –** identifies the window containing the control.
- **pTimerId** – a constant/number which uniquely identifies the timer.
- **pAttributes** – the attribute flags to be assigned.

Example: (excerpt from Fisheye component)

```
if (mTrackingHwnd && !mTimerRunning)
{
  for (qlong eye = 1; eye <= mEyeCount; ++eye)
  {
    if (!GDIequalRect(&mEye[eye - 1].mCurrentRect, mTracking ? &mEye[eye - 1].mTrackingRect : &mEye[eye - 1].m
    {
      mTimerRunning = qtrue;
      WNDsetTimer(mHWnd, FISHEYE_TIMER_ID, FISH_TIMER_INTERVAL);
      WNDsetTimerAttributesOSX(mHWnd, FISHEYE_TIMER_ID, HWND_TIMER_RUNS_WHEN_EVENTS_ARE_BLOCKED);
      break;
    }
  }
  updateWindows(eci, qtrue, mTracking && !mTimerRunning);
}
```

WNDsetWindowCursor()

```
void WNDsetWindowCursor( HWND pHwnd, qshort pCursor )
```

Associates a cursor with the given window. When the mouse moves over the visible client area of that window and the function WNDcheckCursor is called in response to a WM_SETCURSOR message, the cursor is changed to the specified cursor. The default window cursor for all windows is WND_CURS_DEFAULT which sets the screen cursor to WND_CURS_ARROW, if none of the parent windows have a different window cursor.

- **pHwnd** - identifies the window whose cursor is to be set.
- **pCursor** - specifies the cursor id. This parameter can be one of the WND_CURS_xxx defines.

Example:

See WNDgetWindowCursor.

See also WNDgetWindowCursor, WNDcheckCursor, WM_SETCURSOR

WNDsetWindowLong()

```
qulong WNDsetWindowLong( HWND pHwnd, qlong pOffset, qulong pValue )
```

Sets style information for a window. If the style change causes the window to change physical appearance, the window is invalidated appropriately.

- **pHwnd** - identifies the window for which to set the style information.
- **return** - returns the previous styles.

Example:

See WNDgetWindowLong.

See also WNDgetWindowLong, WNDcreateWindow, WS_xxx (styles), WND_xxx (extended styles)

WNDsetWindowPos()

```
qbool WNDsetWindowPos( HWND pHwnd,
HWND pHwndInsertAfter, qdim pLeft, qdim
pTop, qdim pWidth, qdim pHeight, qulong
pFlags)
```

Changes the size, position, and Z-order of a window.

- **pHwnd** - the window to be positioned.
- **pHwndInsertAfter** - the window to precede the positioned window in the Z-order. This parameter must be a valid window or one of the following values:
  HWND_BOTTOM - The bottom of the Z-order. The system places the window at the bottom of all other sibling windows.
  HWND_TOP - The top of the Z-order.
- **pLeft** - the new position of the left side of the window.
- **pTop** - the new position of the top of the window.
- **pWidth** - the new width of the window.

- **pHeight** - the new height of the window.
- **pFlags** - the window sizing and positioning options. This parameter can be a combination of the following values:
  SWP_HIDEWINDOW - Hides the window.
  SWP_NOMOVE - Retains the current position (ignores the pLeft and pTop parameters).
  SWP_NOSIZE - Retains the current size (ignores the pWidth and pHeight parameters).
  SWP_NOREDRAW - Does not redraw changes. If this flag is set, no repainting of any kind occurs. This applies to the client area, the non-client area (including the title and scroll bars), and any part of the parent window uncovered as a result of the moved window. When this flag is set, the application must explicitly invalidate or redraw any parts of the window and parent window that must be redrawn.
  SWP_NOZORDER - Retains the current ordering (ignores the pHwndInsertAfter parameter).
  SWP_SHOWWINDOW - Displays the window.
- **return** - returns qtrue if successful. Otherwise, it is qfalse.

All coordinates for child windows are client coordinates (relative to the upper-left corner of the parent window's client area).

Example:

See WNDredrawChildren.

See also WNDsetWindowPosEx, WNDbringWindowToTop, WNDshowWindow, WNDmoveWindow, WNDgetWindowRect, WM_WINDOWPOSC
WM_WINDOWPOSCHANGED

WNDsetWindowPosEx()

```
qbool WNDsetWindowPosEx( WNDwindowPosStruct* pWPos )
```

Same as WNDsetWindowPos, but takes a WNDwindowPosStruct for its parameter which contains all the positioning information.

- **pWPos** - points to WNDwindowPosStruct.

See also WNDsetWindowPos

WNDshowCaret()

```
void WNDshowCaret()
```

Shows the caret if Omnis is the current task, and a window owns the caret.

Example:

See WM_FOCUSCHANGED.

See also WNDcreateCaret, WNDdestroyCaret, WNDgetCaretPos, WNDsetCaretPos, WNDhideCaret, WM_FOCUSCHANGED

WNDshowWindow()

```
qbool WNDshowWindow( HWND pHwnd, qulong pCmdShow )
```

Sets the specified window's visibility state.

- **pHwnd** - identifies the window to be shown or hidden.
- **pCmdShow** - specifies how the window is to be shown. This parameter can be one of the following values:
  SW_HIDE - Hides the window and passes activation to another window.
  SW_SHOW - Activates a window and displays it in its current size and position.

- **return** - returns qtrue if the window was previously visible. It is qfalse if the window was previously hidden.

Example:

See WNDcreateWindow.

See also WNDsetWindowPos

WNDstartDraw()

HDC WndStartDraw(HWND pHwnd)

Opens and prepares a drawing port for the given window's client area. This function can be used when drawing is required outside the normal WM_PAINT message.

Note: WNDstartDraw must always be followed by a WNDendDraw, they are not allowed to be nested.

- **pHwnd** - identifies the window which is to be painted.
- **return** - returns the device context in which drawing takes place.

Example:

See WNDredrawChildren, WNDgetWindowFromPt, WNDpaintBorder.

See also WNDendDraw, WNDbeginPaint, WNDendPaint, WM_PAINT, WM_CHILDPAINT

macOS

On macOS, this takes an additional optional parameter.

HDC WNDstartDraw(HWND pHwnd, qrect pClipRect)

- **pClipRect** - Sets the clipping region of the created HDC to the specified rectangle, effectively disabling drawing outside the specified rectangle and enabling drawing inside the specified rectangle (only used on macOS).

Note (macOS): With the macOS 10.14 SDK, it is no longer possible to invoke drawing commands directly to an HWND outside of the normal WM_PAINT message using WNDstartDraw/WNDendDraw.

These commands are now used to tell macOS to redraw the HWND (a Cocoa NSView) as soon as possible and provide a block of drawing instructions to apply to the HDC. This will invoke a normal WM_PAINT on the HWND before the additional drawing instructions are applied. This ensures the HWND draws its normal contents (e.g. background) before the additional drawing instructions are applied to it.

On macOS, the WNDstartDraw and WNDendDraw functions define an Objective-C block object (similar to closures or lambdas in other programming languages). Therefore, this requires that the functions are invoked in the same scope or the compiler will report an error. Variables outside the scope of the block are by default read-only. This means that if the contents of the block attempt to alter the value, the compiler will report an error. To be able to change the value of a variable, it must be marked as a block variable by prefixing with the BLOCKVAR definition (a NOP on non-macOS platforms).

Example: (excerpt from Zoom control)

```
void tqfZoomIn::drawDesktopSelectRect( qpoint* pPoint )
{
  BLOCKVAR qrect zoom = getZoomRect( pPoint ); // Mark this as a BLOCKVAR as it is passed by reference in the
  HDC desktop = WNDstartDraw( HWND_DESKTOP );
  GDIdrawFocusRect( desktop,&zoom );
  WNDendDraw( HWND_DESKTOP, desktop );
  mLastMousePos = *pPoint;
}
```

WNDstartDrawEx()

HDC WndStartDrawEx(HWND pHwnd, qrect pClipRect)

Identical to WNDstartDraw on non-macOS platforms.

On macOS this sets up the drawing port in the same way as WNDstartDraw but does not start a drawing block. This allows drawing blocks to be defined in a separate scope.

- **pHwnd** - identifies the window which is to be painted.

See also WNDstartDraw, WNDendDraw, WNDbeginPaint, WNDendPaint, WM_PAINT, WM_CHILDPAINT

WNDupdateLayeredWindow()

void WNDupdateLayeredWindow(HWND pHwnd, qrect &pWindowRect, HBITMAP pBitmap)

Updates the specified part of a layered window with the supplied bitmap.

- **pHwnd** - identifies the window which is to be updated.
- **pWindowRect** – identifies the bounds of the area to be updated.
- **pBitmap** – the bitmap for the area to be updated.

Example: (excerpt from Fisheye control)

```
void tqfFishEyeObject::createLayeredWindow(EXTCompInfo *eci, HWND &pHwnd, qrect &pRect)
{
  WNDborderStruct border(WND_BORD_NONE);
  qrect bounds(pRect);
  pHwnd = WNDcreateWindow(hwnd(), 0, WND_LAYERED, WNDgetProcInst(hwnd()), &bounds, &border);
  if (pHwnd)
  {
    WNDshowWindow(pHwnd, SW_SHOW);
    ECOinsertObject(eci, pHwnd, (void *) this);
  }
}
//..
if (mTextBitmap)
{
  qrect tr; getTextRect(tr);
  if (!mTextHwnd) createLayeredWindow(eci, mTextHwnd, tr);
  WNDupdateLayeredWindow(mTextHwnd, tr, mTextBitmap);
}
```

WNDupdateWindow()

void WNDupdateWindow(HWND pHwnd)

Updates the given window by sending a WM_PAINT message to the window if the update region for the window is not empty. If the update region is empty, no message is sent.

- **pHwnd** - identifies the window to be updated.

Example:

```
// this example moves a window and immediately causes the invalid areas
// to be updated by calling update window for itself and its parent window.
WNDmoveWindow( myHwnd, 40, 40, 100, 20, qtrue );
WNDupdateWindow( myHwnd );
WNDupdateWindow( WNDgetParent( myHwnd ) );
```

See also WNDupdateWindowCO, WNDredrawWindow, WNDredrawWindowCO, WM_PAINT

WNDupdateWindowCO()

void WNDupdateWindowCO(HWND pHwnd, qulong pComponents)

Updates the specified components of the given window by calling WNDupdateWindow for each matching component it finds.

- **pHwnd** - identifies the window who's component(s) are to be updated.
- **pComponents** - specifies the type(s) of the components to be updated. It can be one or a combination of the WND_WC_xxx component type flags.

Example:

```
// this example updates all components of a window
WNDupdateWindowCO( myHwnd, WND_WC_MASK );
```

See also WNDupdateWindow, WNDredrawWindow, WNDredrawWindowCO, WM_PAINT

WNDwindowFromDC() (v3.1)

HWND WNDwindowFromDC(HDC pHdc)

This function returns the HWND associated with the given HDC.

- **returns** – the HWND. This will be NULL if the DC is not a window DC.

See also WNDbeginPaint, WNDstartDraw

Chapter 12—GDI Reference

The Graphic Design Interface or GDI module is the Omnis cross-platform graphic library. This chapter describes the public interface of the GDI module. It lists the structures, data types, defines, and functions available in the GDI module.

An HDC is a handle or reference to a device context. A device context (DC) is a link between an application, a device driver, and an output device, such as a printer or monitor. All drawing, whether it is to the screen or printer, occurs through a device context.

Structures, Data types, and Defines

GDItextSpecStruct

This structure is passed to some of the GDI text functions. It specifies the font, size and extra spacing, the font style, justification and text color.

```
struct GDItextSpecStruct
{
  qfnt  mFnt;
  qsty  mSty;
  qjst  mJst;
  qcol  mTextColor;
  GDItextSpecStruct(){}
  GDItextSpecStruct( qfnt& pFnt, qsty pSty = styPlain,
  qcol pTextColor = colDefault, qjst pJst = jstLeft )
};
```

HBITMAP

The HBITMAP is a handle to a black and white or color bitmap.

HBITMAPMASK

The HBITMAPMASK is a handle to a black and white bitmap.

HBRUSH

The HBRUSH is a handle or pointer to a brush object. It contains the fill pattern for all GDI fill operations. You can create an HBRUSH by calling GDIcreateBrush.

HFONT

The HFONT is a handle or pointer to a font object. It contains the font's name, size, style, and so on for text drawing. You can create an HFONT object by calling GDIcreateFont.

HPEN

The HPEN is a handle or pointer to a pen object. It contains the line style and color for drawing lines, framing rectangles, and so on. You can create an HPEN by calling GDIcreatePen.

HPALETTE

The HPALETTE is a handle to a palette object. Its contains colors for all GDI palette operations. You can create a HPALETTE by calling GDIcreatePalette.

HPIXMAP

The HPIXMAP is a handle do a device independent bitmap.

HPIXMAPinfo

The HPIXMAPinfo structure is used to get information about a particular HPIXMAP by calling GDIgetHPIXMAPinfo.

```
typedef struct
{
  qdim   mWidth;
  qdim   mHeight;
  qdim   mNumColors;
  qdim   mBitCount;
} HPIXMAPinfo;
```

qcol

The qcol is a longint storing an index value to the system color table or the three RGB values in the low three bytes of the long. If qcol stores an index value, the high bit of the long is set, otherwise the high byte is zero.

The following non-system color constants are defined:

- colDefault
  no valid color (appropriate controls system color should be used)
- colNone
  no valid color (appropriate controls system color should be used)
- colBlack
  black
- colWhite
  white
- colRed
  red
- colLightShade
  light gray, same as system color GDI_COLOR_3DFACE
- colMedShade
  medium gray, same as system color GDI_COLOR_3DSHADOW
- colDarkShade
  darker gray, almost a black gray

The following basic system colors are available:

- GDI_COLOR_3DDKSHADOW
  Dark shadow (almost black) for 3D controls.
- GDI_COLOR_3DFACE
  Face color for 3D controls.
- GDI_COLOR_3DHIGHLIGHT
  Highlight color for 3D controls (edge facing the light source).
- GDI_COLOR_3DHILIGHT
  Highlight color for 3D controls (edge facing the light source).
- GDI_COLOR_3DLIGHT
  Light color for 3D controls (edge facing the light source).
- GDI_COLOR_3DSHADOW
  Shadow color for 3D controls (edge facing away from the light source).

- GDI_COLOR_ACTIVEBORDER
  Color of the active window border.
- GDI_COLOR_ACTIVECAPTION
  Color of the active window caption.
- GDI_COLOR_APPWORKSPACE
  Background color of the Omnis application window (always white under MacOS).
- GDI_COLOR_BACKGROUND
  Background color of the desktop (always white under MacOS).
- GDI_COLOR_BTNFACE
  Face color of standard system controls/buttons (always white under MacOS).
- GDI_COLOR_BTNHIGHLIGHT
  Highlight color for pushbuttons (edge facing the light source).
- GDI_COLOR_BTNHILIGHT
  Highlight color for pushbuttons (edge facing the light source).
- GDI_COLOR_BTNSHADOW
  Shadow color for pushbuttons (edge facing away from the light source).
- GDI_COLOR_BTNTEXT
  Text color for pushbuttons.
- GDI_COLOR_CAPTIONTEXT
  Text color for of window caption text.
- GDI_COLOR_DESKTOP
  Background color of the desktop (always white under MacOS).
- GDI_COLOR_GRAYTEXT
  Text color of grayed/disabled text.
- GDI_COLOR_HIGHLIGHT
  Background highlight color for selected items/text in a control.
- GDI_COLOR_HIGHLIGHTTEXT
  Text color for selected text in a control.
- GDI_COLOR_INACTIVEBORDER
  Inactive window border color.
- GDI_COLOR_INACTIVECAPTION
  Window caption color of inactive windows.
- GDI_COLOR_INACTIVECAPTIONTEXT
  Text color for text in window caption of inactive windows.
- GDI_COLOR_INFOBK
  Background color for tooltips.
- GDI_COLOR_INFOTEXT
  Text color for tooltips.
- GDI_COLOR_MENU
  Menu background color.
- GDI_COLOR_MENUTEXT
  Menu text color.
- GDI_COLOR_QBACKFILL
  Standard object fill color for clear pixels in the fill pattern.
- GDI_COLOR_QBLACK
  Standard VGA black.

- GDI_COLOR_QBLUE
  Standard VGA blue.
- GDI_COLOR_QCYAN
  Standard VGA cyan.
- GDI_COLOR_QDEFAULT
  Not a valid color. Use objects default.
- GDI_COLOR_QDKBLUE
  Standard VGA blue.
- GDI_COLOR_QDKCYAN
  Standard VGA dark cyan.
- GDI_COLOR_QDKGRAY
  Standard VGA dark gray.
- GDI_COLOR_QDKGREEN
  Standard VGA green.
- GDI_COLOR_QDKMAGENTA
  Standard VGA dark magenta.
- GDI_COLOR_QDKRED
  Standard VGA dark red.
- GDI_COLOR_QDKYELLOW
  Standard VGA dark yellow/brown.
- GDI_COLOR_QFOREFILL
  Standard object fill color for set pixels in the fill pattern.
- GDI_COLOR_QFRAME
  Standard object border/line color.
- GDI_COLOR_QGRAY
  Standard VGA gray.
- GDI_COLOR_QGREEN
  Standard VGA green.
- GDI_COLOR_QMAGENTA
  Standard VGA magenta.
- GDI_COLOR_QRED
  Standard VGA red.
- GDI_COLOR_QWHITE
  Standard VGA white.
- GDI_COLOR_QYELLOW
  Standard VGA yellow.
- GDI_COLOR_SCROLLBAR
  Scrollbar background color.
- GDI_COLOR_WINDOW
  Window background color.
- GDI_COLOR_WINDOWFRAME
  Window frame color.
- GDI_COLOR_WINDOWTEXT
  Color for text in windows.

qColorEntry

This structure is passed to some of the GDI picture functions. It specifies the color by breaking the color into its red, green and blue parts.

```
typedef struct
{
  qbyte  mBlue;
 qbyte  mGreen;
 qbyte mRed;
  qbyte  mReserved;
} qColorEntry
```

qdim

This basic type stores the screen coordinates. The *qdim* is defined as an integer under WIN16 (16bit integer) and WIN32 (32bit integer), and as a 16 bit short under MacOS.

The following predefined constants exist:

- qdimnone
  not a valid coordinate (defined as -32767)
- maxqdim
  maximum value which can be stored in a qdim (defined as 32766)

qdmd

An enum defining the drawing modes supported by the GDI. qdmd is used with GDIsetDrawingMode to change the current transfer mode when drawing. The following modes are supported:

- dmdCopy
  Overwrite all screen pixels with source pixels
- dmdOr
  Overwrite where source pixel is set
- dmdXor
  Invert where source pixel and screen pixel are both set

qfnt

The qfnt type specifies the font, size, and extra spacing. There are various qfnt related GDI functions (GDIfontXxx) for manipulating a qfnt. *Never* manipulate a qfnt directly.

The following are predefined qfnt constants:

- fnt6Gen
  Tiny Geneva font. (WIN = Helv 6, MacOS = Geneva 6)
- fnt6Mon
  Tiny Monaco font (WIN = Helv 6, MacOS = Monaco 6)
- fntButt
  Default button font (WIN = Helv 8, MacOS = Geneva 10)
- fntCheck
  Default check box & radio button font (WIN = Helv 8, MacOS = Geneva 10)
- fntChicago
  Standard font (WIN = System 10, MacOS = Chicago 12)

- fntEdit
  Default edit field font (WIN = Helv 8, MacOS = Geneva 9)
- fntFindRep
  Find and replace window controls font (WIN = Helv 8, MacOS = Chicago 12)
- fntFix
  Used when fixed point font is required (WIN = System 9, MacOS = Monaco 9)
- fntFmtList
  Formats small list (WIN = Helv 8, MacOS = Chicago 12)
- fntGen
  General purpose font (WIN = Helv 8, MacOS = Geneva 9)
- fntLabel
  Default static text/labels font (WIN = Helv 8, MacOS = Geneva 9)
- fntList
  Default list field font (WIN = Helv 8, MacOS = Geneva 10)
- fntMenubar
  Menubar font (WIN95 = system setting, WIN = System 10, MacOS = Chicago 12)
- fntNone
  Represents no legal font. The object's default font should be used
- fntPromptRS
  Prompt for report/search font (WIN = Helv 8, MacOS = Chicago 12)
- fntSlist
  Default list field font when smaller font is required (WIN = Helv 8, MacOS = Geneva 9)
- fntSmallFnt
  Small font (WIN = Small Fonts 7, MacOS = Geneva 9)
- fntStatusBar
  Default status bar font (WIN95 = system setting, WIN = Helv 8, MacOS = Geneva 9)
- fntSystem
  Systems default font (WIN = System 10, MacOS = Chicago 12)

qjst

An enum defining the text justifications supported by the GDI.

- jstCenter
  Draws text center justified, text is drawn centrally to the specified horizontal coordinate.
- jstExtending
  Draws text left justified, text is drawn to the right of the specified horizontal coordinate. This justification has a special meaning to report text fields, making them vertically grow if the text doesn't fit.
- jstLeft
  Draws text left justified, text is drawn to the right of the specified horizontal coordinate.
- jstNone
  Represents no legal justification. The objects default justification should be used.
- jstRight
  Draws text right justified, text is drawn to the left of the specified horizontal coordinate.
- jstRightToLeft
  Draws text from right to left, text is draw to the left of the specified horizontal coordinate.

qpat

The qpat is a simple unsigned short which is used to index an array of patterns which is defined in the GDI module. The following patterns are supported:

- patGrayFrame
  pattern used for painting dotted frame.
- patDash
  standard dash pattern (4 pixels set, 4 pixels cleared). Used for drawing dashed lines.
- patDkgray
  gray pattern (50% set bits)
- patEmpty
  same as patStd1 (empty fill)
- patFill
  same as patStd0 (solid pattern)
- patGray, patGrayor
  two gray patterns (25% set bits)
- patGray45
  same as patDkgray (50% set bits), but the set bits are arranged at a different angle
- patGrayrev
  same as patDkgray, but pattern is reversed (set bits a clear, clear bits are set)
- patLtgray
  gray pattern (12.5% set bits)
- patPen8 to patPen14
  these are the standard 7 pen patterns which are supported by the design interface for user objects.
- patStd0 to patStd15
  these are the standard 16 patterns supported by the Omnis design interface for user objects. patStd15 indicates that the object is to be transparent and these objects will not be filled with any pattern.
- patTransparent
  same as patStd15 (no fill, background is transparent)

qpen

The qpen class contains the ID of an Omnis 8 byte pattern array (qpat), the pattern color (qcol), and the pen width. It is typically used to specify frame pattern, color and thickness. The definition is as follows:

```
class qpen
{
  public:
     qdim    mWidth;    // the pen width in screen units
     qpat    mPat;      // the pattern id
     qcol    mColor;    // the pen color
     qbool   mIsHairLine;  // true if it is a hairline (printing only)
  public:
    qpen();
    qpen( qpen* pPen );
    qpen( qdim pWidth, qcol pColor = colBlack, qpat pPat = patFill, qbool pIsHairLine = qfalse );
    qdim getWidth() { return mWidth; }
    qpat getPat() { return mPat; }
    qcol getColor() { return mColor; }
    qbool isHairLine() { return mIsHairLine; }
    void setWidth( qdim pWidth ) { mWidth = pWidth; }
```

```
    void setPat( qpat pPat ) { mPat = pPat; }
    void setColor( qcol pColor ) { mColor = pColor; }
    void setHairline( qbool pHairLine ) { mIsHairLine = pHairLine; }
};
```

qpoint

The qpoint is a simple structure storing horizontal and vertical coordinates. The order and storage size of the qdim members may vary from platform to platform.

Example (Macintosh):

```
struct qpoint
{
  qdim v;
  qdim h;
  qpoint(): h(0), v(0) {}
  qpoint( qdim h1, qdim v1 ): h(h1), v(v1) {}
  qpoint( Point pt ): h(pt.h), v(pt.v) {}
  operator Point&() { return (Point&)*this; }
  void operator =( Point &pt ) { *this = (qpoint&)pt; }
};
```

qrect

The qrect is a simple structure storing the left, top, right and bottom to define a rectangle. The order and storage size of the qdim members may vary from platform to platform.

Example (Macintosh):

```
struct qrect
{
  qdim top;
  qdim left;
  qdim bottom;
  qdim right;
  qrect() {}
  qrect( qdim left1, qdim top1, qdim right1, qdim bottom1 )
      : left(left1), top(top1), right(right1), bottom(bottom1) {}
  qdim width() { return right - left + 1; }
  qdim height() { return bottom - top + 1; }
  qpoint topLeftPt() { return qpoint( left, top ); }
  qpoint botRightPt() { return qpoint( right, bottom ); }
};
```

qrgn

The qrgn is a simple structure which encapsulates the system's own Region type.

```
// MacOS example:
struct qrgn
{
  public:
    RgnHandle    rgn;
    qrgn() { if ( ! ( rgn = NewRgn() ) ) merror(); }
    qrgn( RgnHandle pRgn ) { rgn = pRgn; }
    ~qrgn() { DisposeRgn( rgn ); }
};
```

qsty

The qsty is a simple unsigned char used to specify the style for text drawing. The GDI supports the following styles:

- styBold
- styCondense (MacOS only)
- styExtend (MacOS only)
- styItalic
- styOutline (MacOS only)
- styPlain
- styShadow (MacOS only)
- styUnderline

In addition the following style constants complement some of the qfnt constants:

- styButt
  Default style for pushbutton text. Used with fntButt. (WIN = styPlain, MacOS = styBold)
- stylist
  Default list field style. Used with fntList. (styPlain)
- stySlist
  Default list field style for small font. Used with fntSlist. (styPlain)
- styEdit
  Default edit field style. Used with fntEdit. (styPlain)
- styLabel
  Default static text/labels style. Used with fntLabel. (styPlain)
- styPromptRS
  Prompt for report/search style. Used with fntPromptRS. (styPlain)
- styFindRep
  Find and replace window controls style. Used with fntFindRep. (styPlain)
- styStatusBar
  Default status bar style. Used with fntStatusBar. (styPlain)

Functions

GDIalignPicture()

```
qbool GDIalignPicture(qrect &pRect, qrect
&pPictureRect, qshort pPictureAlignment,
qbool pStretch, qbool pKeepAspectRatio,
qbool pHasHscroll, qbool pHasVscroll )
```

GDIalignPicture is a utility function which returns a rect to which to draw a picture. The rect is calculated depending on the destination rect, picture rect and various other given options. The result is returned in pRect. After calling this function you simply call GDIdrawPicture using pRect.

- **pRect –** The destination rect. Receives the rect to which to draw the picture.
- **pPictureRect –** The picture bounds.

- **pPictureAlignment –** Specifies the vertical and horizontal required alignment. If pStretch is true, this parameter is ignored. The vertical alignment must be passed in the high nibble of the low byte, and the horizontal alignment in the low nibble. For example: GDIpictVertAlignTop << 4 + GDIpictHorzAlignLeft
  You can use the following constants for this parameter.
  GDIpictVertAlignTop
  Align the picture at the top of pRect
  GDIpictVertAlignCenter
  Align the picture vertically central of pRect
  GDIpictVertAlignBottom
  Align the picture at the bottom of pRect
  GDIpictHorzAlignLeft
  Align the picture at the left of pRect
  GDIpictHorzAlignCenter
  Align the picture horizontally central of pRect
  GDIpictHorzAlignRight
  Align the picture at the right of pRect
- **pStretch** – If true, and pKeepAspectRatio is false, pRect is returned unaltered. If pKeepAspectRatio is true, the picture rect is stretched to the maximum possible while maintaining the aspect ratio of the picture.
- **pKeepAspectRatio** – This parameter is only relevant when pStretch is true. See pStretch for full details.
- **pHasHscroll** – If true, the horizontal picture alignment is ignored, and the picture rect is not altered horizontally.
- **pHasVscroll** – If true, the vertical picture alignment is ignored, and the picture rect is not altered vertically.

Example:

```
qrect clientRect; WNDgetClientRect( theHwnd, &clientRect );
qrect pictRect; GDIpictGetBounds( thePict, &pictRect );
GDIalignPicture( clientRect, pictRect, GDIpictVertAlignCenter<<4+GDIpictHorzAlignCenter, qfalse, qfalse, qfals
```

GDIalphaBitmapToPIXMAP() (v3.3)

HPIXMAP GDIalphaBitmapToPIXMAP(HBITMAP pBitmap)

Converts an alpha bitmap image to PIXMAP picture format, returning the converted image.

- **pBitmap -** The HBITMAP to be converted.

GDIalphaPixmapToBitmap() (v3.3)

HBITMAP GDIalphaPixmapToBitmap(HPIXMAP pPixmap)

Converts an alpha PIXMAP image to an alpha bitmap image, returning the converted image.

- **pPixmap** – The HPIXMAP to be converted.

GDIanimatePalette()

void GDIanimatePalette (HPALETTE pPalette, qshort pStart, qshort pCount, qColorEntry* pEntries)

GDIanimatePalette function replaces entries in the specified HPALETTE. The caller does not have to update the client area when he/she calls GDIanimatePalette, it will be mapped to the system palette immediately.

- **pPalette -** Identifies the HPALETTE changes are to made to.
- **pStart -**The starting place into the palette.
- **pCount** - The number of color entries to be set.
- **pEntries**- Pointer to the Color Entries.

GDIbitmapToColorShared()

```
qbool GDIbitmapToColorShared( HBITMAP pSource, EXTfldval& pData )
```

Converts a HBITMAP to an Omnis color shared picture format.

- **pSource** - The HBITMAP to be converted.
- **pData** - Where the color shared picture data will be stored.
- **returns** - qtrue if successfully converted the HBITMAP to color shared format.

See also GDIpixmapToColorShared

GDIbitmapToMonoPIXMAP() (v3.3)

```
HPIXMAP GDIbitmapToMonoPIXMAP( HBITMAP pBitmap )
```

Converts a HBITMAP to monochromatic PIXMAP picture format, returning the converted image.

- **pBitmap -** The HBITMAP to be converted.

GDIbitmapToPIXMAP() (v3.3)

```
PIXMAP GDIbitmapToPIXMAP( HBITMAP
pBitmap ) HPIXMAP GDIbitmapToPIXMAP(
HBITMAP pBitmap, quint pDepth )
```

Converts a HBITMAP to a PIXMAP picture format, returning the converted image.

- **pBitmap -** The HBITMAP to be converted.
- **pDepth –** The depth of the bitmap in bits, i.e. 8,16,24 or 32.

GDIcalcJstEscLen() (v5.0)

```
qdim GDIcalcJstEscLen(qchar* pText,qlong pLen,qbool pIsAscii)
```

Calculates the width of text resulting from the specified escape sequence. Escape characters for embeded style characters are defined in gdi.he (used by GDIdrawTextJst).

- **pText** – A string containing the character escape sequence including the escape delimiter.
- **pLen** – The length in characters of the text.

- **pIsAscii** – If qtrue, indicates that the text contains non-Unicode data.

Example:

```
qdim escLen = 0;
while ( theLen < pDataLen )
{
  if ( *pData==txtEsc )
  { // Skip over escape sequences
    qdim len = GDIcalcJstEscLen(pData,pDataLen,qtrue);
    if ( len==0 ) len++; else escLen+=len;
    pData = incads( pData, len );
    theLen+=len;
  }
  else if ( *pData== crch || *pData == spacech || *pData == tabch || (mBreakOnFsch && *pData == fsch) || ( (th
  {
    thisBreak = theLen + 1;
    break;
  }
  else
  {
    pData = incads( pData, 1 );
    theLen++;
  }
}
```

GDIcgContextFlush() (v4.1, Mac OSX only)

```
void GDIcgContextFlush(HDC pHdc)
```

Used at the end of drawing/painting on Mac OSX platform, GDIcgContextFlush calls CGContextFlush, which forces all pending drawing operations in a window context to be rendered immediately to the destination device.

- **pHdc** – Identifies the destination device.

GDIcharWidth()

```
qdim GDIcharWidth( qchar pChar ) qdim
GDIcharWidth( qchar pChar,
GDItextSpecStruct* pTextSpec ) qdim
GDIcharWidth( HDC pHdc, qchar pChar)
```

Returns the width of the given character.

If only pChar is specified, the current HFONT in the temp DC is used (see GDIgetTempDC).

If pTextSpec is specified, the function bases the char width on an HFONT based on the text spec.

If pHdc is specified, the width is based on the current HFONT in the given DC.

- **pChar** - The character for which to calculate the width in screen units.
- **pTextSpec** - Specifies the font and style.
- **pHdc** - Identifies the device which contains the HFONT of interest.
- **return** - returns the width of the character in screen units.

See also GDItextWidth

GDIcheckPort()

```
void GDIcheckPort( HDC pHdc )
```

This function does nothing under Windows. Under MacOS it sets the current port to the specified port. You shouldn't ever need to call this function from outside the GDI module, unless direct calls to the MacOS API are made which require the current port to be set.

- **pHdc** - Identifies the port to be made current.

GDIclearClip()

```
void GDIclearClip( HDC pHdc )
```

Clears any clipping in the given device by setting the clipping region to an arbitrary large area.

- **pHdc** - Identifies the device for which the clipping will be cleared.

See also GDIsetClipRect, GDIsetClipRgn

GDIclearClipAlpha() (v5.0)

```
qbool GDIclearClipAlpha(HDC pHdc)
```

Clears any clipping in the given device by setting the clipping region to an arbitrary large area. The device must be created using GDIcreateAlphaDC()

- **pHdc** - Identifies the device for which the clipping will be cleared.

GDIconvToSysPict()

```
qbool GDIconvToSysPict(qchar* add, qlong len, qlong& Picture)
```

Coverts Omnis picture data into Operating System Specific picture.

- **add-**Points to Omnis picture data.
- **len**- The length of the Omnis picture.
- **Picture**- The Operating System specific picture.

Returns qtrue if an Operating System specific pictures was successfully created.

GDIconvToOmnisPict ()

```
void GDIconvToOmnisPict(qchar* add, qlong len, EXTfldval& pData)
```

Converts an Operating System picture to Omnis picture data.

- **add** - Points to Operating System Picture data.
- **len** - The length of Operating System Picture data.
- **pData** - The Omnis Picture data is store in store in the EXTfldval.

GDIconvToSharedPict()

qbool GDIconvToSharedPict(EXTfldval &pPictureToConvert)

Coverts Omnis picture to Omnis color-shared picture format.

- **pPictureToConvert-** EXTfldval which contains the Omnis picture data is then converted to Omnis color shared picture data.

GDIcopyBits()

void GDIcopyBits(HDC pSrcDC, HDC pDestDC, qrect* pSrcRect, qrect* pDestRect, qbool pStretch)

Copies the pixels enclosed by the specified source rect from the source DC to the specified destination rect in the destination DC.

- **pSrcDC** - Identifies the source device from which the pixels are copied.
- **pDestDC** - Identifies the destination device to which the pixels are copied.
- **pSrcRect** - If this parameter is NULL, all the pixels in the source dc are copied, otherwise only the part of the dc enclosed by the rectangle is copied.
- **pDestRect** - If this parameter is NULL, the copied pixels are drawn to the whole of the specified DC, otherwise they are only drawn to the part of the DC enclosed by the rectangle.
- **pStretch** - If qtrue, the pixels are stretched or shrunk to fit exactly into pDestRect, otherwise the pixels are drawn as is.

Example:

```
// this example copies a bitmap image to the screen during a
// paint message of a window
qrect cRect;
WNDgetClientRect( hWnd, &cRect );
WNDpaintStruct paintInfo;
WNDbeginPaint( hWnd, &paintInfo );
  HDC tempDC = GDIgetTempDC();
  HBITMAP oldBitmap = GDIselectBitmap( tempDC, myBitmap );
  GDIcopyBits( tempDC, paintInfo.hdc, &myBitmapRect, &cRect, qtrue );
  GDIselectBitmap( tempDC, oldBitmap );
WNDendPaint( hWnd, &paintInfo );
```

See also GDIdrawBitmap

GDIcopyRect()

void GDIcopyRect(qrect* pDestRect, qrect* pSrcRect)

Copies the dimensions stored in pSrcRect to pDestRect.

- **pDestRect** - Points to the destination qrect structure.
- **pSrcRect** - Points to the source qrect structure.

See also GDIsetRect, GDIsetRectEmpty

GDIcreateAlphaDC() (v5.0)

void GDIcreateAlphaDC(HDC *pHdc, qdim pWidth, qdim pHeight)

Creates an alpha HDC suitable for drawing into an alpha bitmap; creates alpha bitmap with specified dimensions and attaches it to the alpha HDC. Not supported for Linux and Mobile client platforms.

- **pHdc** – A pointer to the HDC to be initialised.
- **pWidth** – The width in pixels of the new DC.
- **pHeight** – The height in pixels of the new DC.

Example:

```
HDC hdc;
GDIcreateAlphaDC(&hdc, mTextRect.width(), mTextRect.height());
```

GDIcreateBitmap()

HBITMAP GDIcreateBitmap(qdim pWidth, qdim pHeight, qchar pDepth, HDC pHdc = 0)

Creates and returns an empty bitmap of the specified width, height, and depth. This bitmap can be selected into a drawing device/port which was created with GDIcreateScreenDC. The DC can be drawn to (off screen) using any of the GDI drawing functions.

- **pWidth** - the width in screen units of the bitmap.
- **pHeight** - the height in screen units of the bitmap.
- **pDepth** - the depth (color resolution) of the bitmap. Supported values are 1 and any other value; a value of 1 creates a monochrome bitmap, and other values always create a device-compatible bitmap i.e. the depth is that is required by the device.
- **pHdc -** the device context to be used when creating a device-compatible bitmap. GDIcreateBitmap ignores this argument when the depth is 1. When the depth is not 1, GDIcreateBitmap uses this device context if it is not zero, to create the compatible bitmap; if the device context is zero, GDIcreateBitmap creates a temporary device context for the main window, and uses that to create the compatible bitmap.
- **return** - returns the HBITMAP.

Example:

See GDIdragBitmapMove, GDIdrawBitmap.

See also GDIdeleteBitmap, GDIselectBitmap, GDIdrawBitmap, GDImaskFromBitmap

GDIcreateBitmapEx()

HBITMAP GDIcreateBitmapEx (qdim pWidth, qdim pHeight, qbyte pDepth,HDC pHdc = 0)

Creates and returns an empty alpha bitmap of the specified width, height, and depth. This bitmap can be selected into a drawing device/port which was created with GDIcreateScreenDC. The DC can be drawn to (off screen) using any of the GDI drawing functions.

- **pWidth** - the width in screen units of the bitmap.
- **pHeight** - the height in screen units of the bitmap.
- **pDepth** - the depth (color resolution) of the bitmap. Supported values are 1 and any other value; a value of 1 creates a monochrome bitmap, and other values always create a device-compatible bitmap i.e. the depth is that is required by the device.
- **pHdc -** the device context to be used when creating a device-compatible bitmap. GDIcreateBitmap ignores this argument when the depth is 1. When the depth is not 1, GDIcreateBitmap uses this device context if it is not zero, to create the compatible bitmap; if the device context is zero, GDIcreateBitmap creates a temporary device context for the main window, and uses that to create the compatible bitmap.
- **return** - returns the HBITMAP.

GDIcreateBrush()

HBRUSH GDIcreateBrush(qpat pPat)

Creates an HBRUSH for filling operations.

- **pPat** - Specifies the pattern to be used.
- **return** - Returns the HBRUSH.

Example:

See GDIcreatePoly, GDIfillPoly.

See also GDIgetStockBrush, GDIselectObject, GDIdeleteObject

GDIcreateCursor()

HCURSOR GDIcreateCursor (qpoint *pHotSpot, HBITMAP pColor, HBITMAPMASK pMask)

Creates a mouse cursor using the specified bitmap, color and hotspot.

Example:

```
WNDsetCapture( hwnd(), WND_CAPTURE_MOUSE );
HBITMAP color = mView->getCursorBitmap();
HBITMAPMASK mask = mView->getCursorMask(color);
qshort ind = mView->mPicPage->mCurrentCell.mVCell-1;
lookupIcon* icons = mEditor->iconArray();
qpoint hotpoint(0,0);
if ( icons[ind].mHotSpots )
{
  hotpoint = *(icons[ind].mHotSpots + mView->mPicPage->mCurrentCell.mHCell-1);
}
mCursor = GDIcreateCursor ( &hotpoint, color, mask );
GDIdeleteBitmap ( color );
GDIdeleteBitmap ( (HBITMAP)mask );
if ( mCursor ) WNDsetCursor(mCursor);
```

GDIcreateDcFont()

HFONT GDIcreateDcFont(qfnt* pFnt, qsty pSty, HDC pHdc) HFONT GDIcreateDCFont(qfnt* pFnt, qsty pSty, HDC pHdc eThemeTextMode pThemeTextMode)

This function creates a HFONT for text drawing operations. It uses the font family information associated with the font and DC.

- **pFnt** - Specifies the qfnt (font, font size and extra spacing).
- **pSty** - Specifies the font style.
- **pHdc** – The HFONT will use the font family information associated with the font and the given DC.
- **pThemeTextMode** – Specifies the modifiers for drawing theme fonts. This parameter is ignored if the specified font is not a theme font. If you need to paint disabled text you would specify eThemeTextInactive. pThemeTextMode can be one of the following:
  eThemeTextActive
  text will be drawn active
  eThemeTextInactive
  text will be drawn inactive
  eThemeTextPressed
  text will be drawn pressed
- **return** - Returns the HFONT.

Example:

See GDIdraw3DPushButton, GDIfontHeight, GDIinflateButtonRect.

See also GDIcreateFont, GDIselectObject, GDIdeleteObject

GDIcreateFont()

HFONT GDIcreateFont(qfnt* pFnt, qsty pSty) HFONT GDIcreateFont(qfnt* pFnt, qsty pSty, eThemeTextMode pThemeTextMode)

This function creates an HFONT for text drawing operations.

- **pFnt** - Specifies the qfnt (font, font size and extra spacing).
- **pSty** - Specifies the font style.
- **pThemeTextMode** – Specifies the modifiers for drawing theme fonts. This parameter is ignored if the specified font is not a theme font. If you need to paint disabled text you would specify eThemeTextInactive. pThemeTextMode can be one of the following:
  eThemeTextActive
  text will be drawn active
  eThemeTextInactive
  text will be drawn inactive
  eThemeTextPressed
  text will be drawn pressed
- **return** - Returns the HFONT.

Example:

See GDIdraw3DPushButton, GDIfontHeight, GDIinflateButtonRect.

See also GDIcreateDcFont, GDIselectObject, GDIdeleteObject

GDIcreateHPIXMAP()

```
HPIXMAP GDIcreateHPIXMAP( qdim pWidth,
qdim pHeight, qchar pDepth, qbool
pOmnisColors )
```

Creates and returns an empty PIXMAP of the specified width, height and depth.

- **pWidth** - the width in screen units of the PIXMAP.
- **pHeight** - the height in screen units of the PIXMAP.
- **pDepth** - the depth (color resolution) of the PIXMAP. Supported values are 0, 1, 2, 4, 8, 16 and 32. If zero is specified the color resolution will be matched with that of the screen.
- **return** - Returns the HPIXMAP.

See also GDIdeleteHPIXMAP, GDIHPIXMAPfromSharedPicture

GDIcreatePalette()

```
HPALETTE GDIcreatePalette ( qshort pCount, qColorEntry* pEntries )
```

Creates and return a HPALETTE of colors specified by count and entries.

- **pCount**- The number of colors to be in the palette.
- **pEntries**- The color entries of the palette.

GDIcreatePen()

```
HPEN GDIcreatePen( qpen* pPen ) HPEN
GDIcreatePen( qdim pWidth = 1, qcol pCol =
GDI_COLOR_QFRAME, qpat pPat = patFill )
```

Creates an HPEN for line and frame operations.

- **pPen** - Points to a qpen structure which specifies the properties of an HPEN.

OR

- **pWidth** - Specifies the pen width.
- **pCol** - Specifies the pen color.
- **pPat** - Specifies the pen pattern.

Example:

See GDIfillPoly, GDIframeEllipse.

See also GDIgetStockPen, GDIselectObject, GDIdeleteObject

GDIcreatePixmapFromJPEG

(Studio 11)

GDIAPI HPIXMAP OMNISAPI GDIcreatePixmapFromJPEG(fldval& pPath, qdim& pWidth, qdim& pHeight)

Creates an HPIXMAP from JPEG image.

- **pPath** - The pathname of the file containing the JPEG image
- **pWidth** - If successful, receives the pixel width of the JPEG
- **pHeight** - If successful, receives the pixel height of the JPEG
- **return** - either the HPIXMAP for success, or NULL if an error occurs e.g. the file does not contain JPEG data

GDIAPI HPIXMAP OMNISAPI GDIcreatePixmapFromJPEG(qbyte* pJpeg, qlong pJpegLen, qdim& pWidth, qdim& pHeight)

Creates an HPIXMAP from JPEG image.

- **pJpeg** - The address of the in-memory JPEG image
- **pJpegLen** - The length in bytes of the in-memory JPEG image
- **pWidth** - If successful, receives the pixel width of the JPEG
- **pHeight** - If successful, receives the pixel height of the JPEG
- **return** - either the HPIXMAP for success, or NULL if an error occurs e.g. the supplied data is not JPEG data.

GDIcreatePolyRgn()

qrgn* GDIcreatePolyRgn(qpoint* pPoints, qshort pNumPoints)

Creates a polygon region from the specified points.

Warning: When finished with the region, the region must be deleted.

- **pPoints** - Points to an array of qpoints which specify the polygon shape.
- **pNumPoints** - Specifies the number of qpoints in the pPoints array.

Example:

```
// this example frames and fills a polygon with a patterned brushes
// create the polygon region ( diamond shape )
qpoint pts[5];
pts[0].h = pts[0].v = 0;
pts[1].h = pts[1].v = 10;
pts[2].h = 0; pts[2].v = 20;
pts[3].h = -10; pts[3].v = 10;
pts[4].h = pts[4].v = 0;
qrgn* myPoly = GDIcreatePolyRgn( &pts[0], 5 );
```

```
// create the brushes
HBRUSH frameBrush = GDIcreateBrush( patPen8 );
HBRUSH fillBrush = GDIcreateBrush( patStd10 );
// frame the polygon
GDIsetBkColor( theDC, GDI_COLOR_QGREEN );
GDIsetTextColor( theDC, GDI_COLOR_QBLUE );
GDIframeRgn( theDC, myPoly, frameBrush );
// fill the poly
GDIsetBkColor( theDC, GDI_COLOR_QRED );
GDIsetTextColor( theDC, GDI_COLOR_QYELLOW );
GDIfillRgn( theDC, myPoly, fillBrush );
// delete the objects
GDIdeleteObject( frameBrush );
GDIdeleteObject( fillBrush );
// delete the polygon
delete myPoly;
```

See also GDIcreateRectRgn, GDIcreateRoundRectRgn

GDIcreateRectRgn()

qrgn* GDIcreateRectRgn(qdim pLeft, qdim pTop, qdim pRight, qdim pBottom) qrgn* GDIcreateRectRgn(qrect* pRect)

Creates a rectangular region from the given coordinates or rectangle.

Warning: When finished with the region, the region must be deleted.

- **pLeft** - Specifies the left coordinate of the region.
- **pTop** - Specifies the top coordinate of the region.
- **pRight** - Specifies the right coordinate of the region.
- **pBottom** - Specifies the bottom coordinate of the region.
- **return** - Returns a pointer to the new region.

OR

- **pRect** - Points to the rectangle specifying the rectangular region.
- **return** - Returns a pointer to the new region.

Example:

```
// this example creates a rectangular region and fills it with the current text color
qrgn* myRgn = GDIcreateRectRgn( 10, 10, 50, 20 );
GDIfillRgn( theDC, myRgn, GDIgetStockBrush( BLACK_BRUSH ) );
delete myRgn;
```

See also GDIcreateRoundRectRgn, GDIcreatePolyRgn

GDIcreateRoundRectRgn()

```
qrgn* GDIcreateRoundRectRgn( qdim pLeft,
qdim pTop, qdim pRight, qdim pBottom,
qdim pWidthEllipse, qdim pHeightEllipse )
```

Creates a rounded corner rectangular region. The degree of rounding depends on the values passed in pWidthEllipse and pHeightEllipse.

Warning: When finished with the region, the region must be deleted.

- **pLeft** - Specifies the left coordinate of the region.
- **pTop** - Specifies the top coordinate of the region.
- **pRight** - Specifies the right coordinate of the region.
- **pBottom** - Specifies the bottom coordinate of the region.
- **pWidthEllipse** - Specifies the width defining curvature of corners.
- **pHeightEllipse** - Specifies the height defining curvature of corners.
- **return** - Returns a pointer to the new region.

Note: This function is like creating a rectangular region and drawing four identical ellipses to replace the regions rectangular corners. It is not particularly fast since drawing to an off screen port takes place to create this region.

Example:

```
// this example creates a rounded rectangular region subtracts it from
// another region and fills the remainder
qrgn* rrRgn = GDIcreateRoundRectRgn( 10, 10, 50, 50, 16, 16 );
qrgn clientRgn;
qrect cRect;
WNDgetClientRect( theHwnd, &cRect );
GDIsetRectRgn( &clientRgn, &cRect );
GDIrgnDiff( &clientRgn, &clientRgn, rrRgn );
GDIfillRgn( theDC, &clientRgn, GDIgetStockBrush( BLACK_BRUSH ) );
delete rrRgn;
```

See also GDIcreateRectRgn, GDIcreatePolyRgn

GDIcreateScreenDC()

```
HDC GDIcreateScreenDC()
```

Creates and returns a DC compatible with the screen. This function is usually used for off screen drawing by also creating an HBITMAP and selecting it into the screen DC created by this function (see GDIselectBitmap). When the DC is no longer required, it MUST be deleted by calling GDIdeleteScreenDC.

- **return** - Returns the newly created DC.

Example:

See GDIdragBitmapMove.

See also GDIdeleteScreenDC, GDIgetTempDC

GDIdeleteAlphaDC() (v5.0)

void GDIdeleteAlphaDC(HDC pHdc, HBITMAP *pReturnedAlphaBitmap)

Deletes the Alpha HDC created with GDIcreateAlphaDC(); sets *pReturnedAlphaBitmap to the alpha bitmap created using the alpha HDC.

- **pHdc** – Identifies the device to be deleted.
- **pReturnedAlphaBitmap** – (output) Returns the alpha bitmap that was created.

GDIdeleteBitmap()

void GDIdeleteBitmap(HBITMAP pBitmap)

Destroys and releases the memory occupied by the bitmap. Call this function if you have created a bitmap using GDIcreateBitmap, and you have finished with the bitmap.

- **pBitmap** - Identifies the bitmap to be deleted.

Example:

See GDIdragBitmapMove.

See also GDIcreateBitmap, GDIselectBitmap

GDIdeleteCursor()

void GDIdeleteCursor(HCURSOR pObject)

Destroys a cursor and frees any memory the cursor occupied.

- **pObject** – The cursor object to be destroyed.

Example:

```
if ( !GDIptInRect( &cRect, pPoint ) )
{
  WNDreleaseCapture( WND_CAPTURE_MOUSE );
  if ( mCursor )
  {
    GDIdeleteCursor(mCursor);
    mCursor = NULL;
  }
}
```

GDIdeleteHPIXMAP()

void GDIdeleteHPIXMAP(HPIXMAP pHPIXMAP)

Destroys and releases the memory occupied by the HPIXMAP. Call this function if you have created a bitmap using GDIcreateHPIXMAP or GDIHPIXMAPfromSharedPicture, and you have finished with the HPIXMAP.

- **pHPIXMAP** - Identifies the HPIXMAP to be deleted.

See also GDIcreateHPIXMAP, GDIHPIXMAPfromSharedPicture

GDIdeleteObject()

```
void GDIdeleteObject( object )
```

Deletes the given object which must have been created by GDIcreateBrush, GDIcreateFont, GDIcreatePen, GDIcreateCursor or GDIcreateBitmap.

- **object** - Specifies the object to be deleted. It can be an HBRUSH, HFONT, HPEN or HBITMAP.

Warning: Never delete an object if it is currently selected in a DC.

Example:

See GDIcreatePoly, GDIdraw3DPushButton, GDIfillPoly, GDIfontHeight, GDIframeEllipse, GDIinflateButtonRect.

See also GDIcreateBrush, GDIcreateFont, GDIcreatePen, GDIselectObject

GDIdeletePalette()

```
void GDIdeletePalette( HPALETTE pPalette )
```

This function must be called when a HPALETTE, which was created by GDIcreatePalette, is no longer required.

- **pPalette** - Identifies the HPALETTE to be deleted.

GDIdeleteScreenDC()

```
void GDIdeleteScreenDC( HDC pHdc )
```

This function must be called when a screen DC, which was created by GDIcreateScreenDC, is no longer required.

- **pHdc** - Identifies the HDC to be deleted.

Example:

See GDIdragBitmapMove.

See also GDIcreateScreenDC, GDIgetTempDC

GDIdragBitmapMove()

```
void GDIdragBitmapMove( HBITMAP
pDragBitmap, HBITMAPMASK pDragMask,
HBITMAP pScreenBitmap, qrect* pSrcRect,
qrect* pDestRect )
```

GDIdragBitmapMove is used together with GDIdragBitmapFromScreen and GDIdragBitmapToScreen to move an image specified by pDragBitmap and pDragMask around the screen without causing any flickering.

Before GDIdragBitmapMove is called to move an image, GDIdragBitmapSave has to be called once, to save the initial screen image of the starting rectangle, and GDIdragBitmapToScreen has to be called once specifying pDragBitmap to place the initial drag bitmap on screen. The image can be moved repeatedly by calling this function as many times as is required. At the end of the drag operation, GDIdragBitmapRestore must be called to restore the screen specifying pScreenBitmap.

GDIdragBitmapMove restores the pSrcRect with pScreenBitmap, and saves pDestRect into pScreenBitmap before drawing pDragBitmap in pDestRect. The whole operation is done off screen to avoid any flicker.

- **pDragBitmap** - Specifies the image to be drawn at pDestRect.
- **pDragMask** - Specifies the image mask.
- **pScreenBitmap** - Specifies the saved screen image at pSrcRect on entry. On exit it will specify the saved screen image of pDestRect.
- **pSrcRect** - Specifies the location on screen from where the image is to be moved.
- **pDestRect** - Specifies the new location on screen to where the image is to be moved.

Example:

```
// initialize the dragBitmap size ( 20 by 20 pixels )
qrect dragRect( 0, 0, 19, 19 );
// first allocate the bitmap to be dragged around screen and the bitmap
// for saving and restoring the screen
HBITMAP dragBitmap = GDIcreateBitmap( dragRect.width(), dragRect.height(), 0 );
HBITMAP screenBitmap = GDIcreateBitmap( dragRect.width(), dragRect.height(), 0 );
// now fill in the dragBitmap with some sort of image
// first create a DC to draw in and select the dragBitmap into it
HDC tmpDC;
HRESERVED hdcResv;
GDIcreateScreenDC( &tmpDC,&hdcResv );
BITMAP oldBitmap = GDIselectBitmap( tmpDC, dragBitmap );
// first we fill everything with white so everything but our image is
// masked out when we create our mask.
GDIsetBkColor( tmpDC, colWhite );
GDIsetTextColor( tmpDC, colBlack );
GDIfillRect( tmpDC, &dragRect, GDIgetStockBrush( WHITE_BRUSH ) );
// now we draw our image which in this case is a simple ellipse. We
// will fill it with a black and white pattern so we get a ghostly
// effect (every other pixel is masked out ).
GDIfillEllipse( tmpDC, &dragRect, GDIgetStockBrush( DKGRAY_BRUSH ) );
// now we create the mask
HBITMAPMASK dragMask = GDImaskFromBitmap(tmpDC,dragRect.width(),dragRect.height());
// restore bitmap in DC and delete it
GDIselectBitmap( tmpDC, oldBitmap );
GDIdeleteScreenDC( tmpDC, hdcResv );
// now we can drag the bitmap around the screen, following the screen
// cursor while the mouse button is down
qpoint lastPt, newPt;
// get the current cursor position and calculate the starting drag
// rect position centrally to that point
WNDgetCursorPos( &lastPt );
GDIoffsetRect( &dragRect, lastPt.h - dragRect.width() / 2, lastPt.v - dragRect.height() / 2 );
// first save the screen and draw the image at the start location
GDIdragBitmapFromScreen( screenBitmap, &dragRect );
GDIdragBitmapToScreen( dragBitmap, dragMask, &dragRect );
while ( WNDmouseLeftButtonDown() )
{
  // see if the cursor has moved
  WNDgetCursorPos( &newPt );
  if ( newPt.h != lastPt.h || newPt.v != lastPt.v )
  {
    // move the bitmap
    qrect newRect = dragRect;
    GDIoffsetRect( &newRect, newPt.h - lastPt.h, newPt.v - lastPt.v );
    GDIdragBitmapMove( dragBitmap, dragMask, screenBitmap, &dragRect, &newRect );
    dragRect = newRect;
    lastpt = newPt;
```

```
  }
}
// the mouse button has been released, restore the screen and delete
// the bitmaps
GDIdragBitmapToScreen( screenBitmap, NULL, &dragRect );
GDIdeleteBitmap( screenBitmap );
GDIdeleteBitmap( (HBITMAP)dragMask );
GDIdeleteBitmap( dragBitmap );
```

See also GDIdragBitmapToScreen, GDIdragBitmapFromScreen

GDIdragBitmapToScreen()

void GDIdragBitmapToScreen(HBITMAP pDragBitmap, HBITMAPMASK pDragMask, qrect* pDestRect)

GDIdragBitmapToScreen is used together with GDIdragBitmapFromScreen and GDIdragBitmapMove to move an image around the screen without causing any flickering.

GDIdragBitmapToScreen draws the bitmap specified by pScreenBitmap at the location specified by pSrcRect to screen.

See GDIdragBitmapMove for a full description of all three functions.

- **pDragBitmap** - Specifies the image to be drawn at pDestRect.
- **pDragMask** - Specifies the image mask. If NULL, no masking of the image takes place.
- **pDestRect** - Specifies the destination of the image on screen.

Example:

See GDIdragBitmapMove.

See also GDIdragBitmapMove, GDIdragBitmapFromScreen

GDIdragBitmapFromScreen()

void GDIdragBitmapFromScreen(HBITMAP pScreenBitmap, qrect* pSrcRect)

GDIdragBitmapFromScreen is used together with GDIdragBitmapToScreen and GDIdragBitmapMove to move an image around the screen without causing any flickering.

GDIdragBitmapFromScreen saves the area on screen specified by pSrcRect into pScreenBitmap.

See GDIdragBitmapMove for a full description of all three functions.

- **pScreenBitmap** - Specifies the destination bitmap for the saved screen image.
- **pSrcRect** - Specifies the area of the screen to be saved.

Example:

See GDIdragBitmapMove.

See also GDIdragBitmapMove, GDIdragBitmapToScreen

GDIdraw3DPushButton()

void GDIdraw3DPushButton(HDC pHdc, qrect* pRect, qcol pBorderColor, qcol pFaceColor, qulong pDrawFlags)

Draws the border and face (*not* the text or icon) of a rectangular 3D pushbutton where the corners of the outer border are clipped off by one pixel (Microsoft Windows style button).

Warning: On return the rectangle will be inset according to the border sizes of the button face, i.e. the result can be used as the boundary for the text of the button.

- **pHdc** - Identifies the device in which the button will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries of the pushbutton.
- **pBorderColor** - Specifies the color to be used to draw the outer single border (or double thickness border for default buttons). If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pFaceColor** - Specifies the color to be used to draw the face of the button. If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pDrawFlags** - Specifies one or more of the following.
  GDI_BUTT_DEFAULT
  If specified the outer border is drawn twice as thick.
  GDI_BUTT_HIGHLIGHT
  The button is drawn depressed. The light and dark shades are reversed.
  GDI_BUTT_TOOLBAR
  Windows 95, toolbar buttons are drawn differently when depressed, hence this flag.

Example:

```
// this is an example of a control which supports all of the button faces.
// The button drawing is encapsulated in one function
// all variables starting with ‘m’ are members of the class.
// The following are assumed to exist
// mFace ( qshort: button face style )
// mBorderColor ( qcol: border color )
// mForeColor ( qcol: foreground fill color )
// mBackColor ( qcol: background fill color )
// mFillPattern ( qpat: the fill pattern )
// mTextSpec ( GDItextSpecStruct: text font, style, size,
// justification, etc )
// mEnabled ( qbool: qtrue if button is enabled )
void clButt::paintButton( HDC pHdc, qrect* pRect, qchar* pText, qshort pTextLen, qbool pDepressed, qbool pDefa
{
  // draw the face
  qulong buttFlags = pDepressed ? GDI_BUTT_HIGHLIGHT : 0;
  if ( pDefault ) buttFlags |= GDI_BUTT_DEFAULT;
  switch ( mFace )
  {
    case GDI_BUTT_FACE_NOBORD:
      break;
    case GDI_BUTT_FACE_SYS:
      GDIdrawSystemPushButton( pHdc, pRect, mBorderColor, mFillPattern, mForeColor, mBackColor, mTextSpec.mTex
      break;
    case GDI_BUTT_FACE_SYS3D:
      GDIdrawSystem3dPushButton( pHdc, pRect, mBorderColor, mForeColor, buttFlags );
      break;
    case GDI_BUTT_FACE_3D:
```

```
      GDIdraw3dPushButton( pHdc, pRect, mBorderColor, mForeColor, buttFlags );
      break;
    case GDI_BUTT_FACE_HEADING:
      GDIdrawHeadingButton( pHdc, pRect, mBorderColor, mForeColor, buttFlags );
      break;
    case GDI_BUTT_FACE_COMBO:
      GDIdrawComboButton( pHdc, pRect, buttFlags );
      break;
  }
  // draw the text if we are not a combo button
  if ( mFace != GDI_BUTT_FACE_COMBO )
  {
    // make a copy of the text. GDItextBox will alter the text if
    // it doesn't fit
    str255 txt( pTextLen, pText );
    HFONT font = GDIcreateFont( &mTextSpec.mFnt, mTextSpec.mSty );
    font = GDIselectObject( pHdc, font );
    GDIsetTextColor( pHdc, mEnabled ? mTextSpec.mTextColor : GDI_COLOR_GRAYTEXT );
    GDItextBox( pHdc, pRect, &txt[1], txt[0], 255, mTextSpec.mJst );
    font = GDIselectObject( pHdc, font );
    GDIdeleteObject( font );
    if ( mHasFocus )
    {
      GDIdrawFocusRect( pHdc, pRect );
    }
  }
}
```

See also GDIdrawComboButton, GDIdrawHeadingButton, GDIdrawSystem3dPushButton, GDIdrawSystemPushButton, GDIinflate-ButtonRect, GDIinsetButtonRect

GDIdrawAlphaBitmap() (v3.3)

```
qbool GDIdrawAlphaBitmap ( HDC pHdc,
HBITMAP pBitmap, qrect* pSrcRect, qrect*
pDestRect, qbool pStretch )
```

Draws an alpha bitmap to the given DC.

- **pHdc** - Identifies the device into which the bitmap is drawn.
- **pBitmap** - Specifies the bitmap to be drawn.
- **pSrcRect** - If this parameter is NULL, the whole bitmap is drawn to the device, otherwise only the part of the bitmap enclosed by the rectangle is drawn.
- **pDestRect** - Specifies the destination coordinates for the bitmap local to the DC. The bottom and right edges of the rectangle are ignored if pStretch is false.
- **pStretch** - If true, the bitmap is stretched or shrunk to fit exactly into pDestRect, otherwise the bitmap is drawn as is.

GDIdrawBitmap()

```
void GDIdrawBitmap( HDC pHdc, HBITMAP
pBitmap, HBITMAPMASK pBitmapMask,
qrect* pSrcRect, qrect* pDestRect, qbool
pStretch )
```

Draws a bitmap to the given DC.

- **pHdc** - Identifies the device into which the bitmap is drawn.
- **pBitmap** - Specifies the bitmap to be drawn.
- **pBitmapMask** - If this parameter is NULL, no masking takes place and all of the bitmap is drawn. If a mask is specified, all pixels not set in the mask are not drawn.
- **pSrcRect** - If this parameter is NULL, the whole bitmap is drawn to the device, otherwise only the part of the bitmap enclosed by the rectangle is drawn.
- **pDestRect** - Specifies the destination coordinates for the bitmap local to the DC. The bottom and right edges of the rectangle are ignored if pStretch is false.
- **pStretch** - If true, the bitmap is stretched or shrunk to fit exactly into pDestRect, otherwise the bitmap is drawn as is.

Example:

```
// this example receives a paint message and paints a button off screen into
// a bitmap selected into a dc and draws that bitmap to screen using the
// same dc. This example utilizes the example given for GDIdraw3DPushButton
qulong clButt::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LPARAM uParam )
{
  switch ( message )
  {
    case WM_PAINT:
    {
      WNDpaintStruct ps;
      WNDbeginPaint( hWnd, &ps );
      qrect cRect;
      WNDgetClientRect( hWnd, &cRect );
      // create and select the bitmap into the dc, remembering
      // the old one
      HBITMAP bmp = GDIcreateBitmap( cRect.width(), cRect.height(), 0 );
      bmp = GDIselectBitmap( ps.hdc, bmp );
      // now paint the button off screen. See
      // GDIdraw3DPushButton example.
// The paintButton function alters the rect, so use a temp
      qrect tmpRect = cRect;
      paintButton( ps.hdc, &tmpRect, mText, mTextLen, qfalse, qfalse );
      // swap the bitmaps and draw the bitmap to screen
      bmp = GDIselectBitmap( ps.hdc, bmp );
      GDIdrawBitmap( ps.hdc, bmp, NULL, &cRect, &cRect, qfalse );
      WNDendPaint( hWnd, &ps );
      return 0L;
    }
  }
  return DefWindowProc( hWnd, message, wParam, lParam );
}
```

See also GDIdrawBitmapChisel, GDIdrawIcon, GDIdrawPicture, GDIhiliteBitmap, GDIselectBitmap

GDIdrawBitmapChisel()

```
void GDIdrawBitmapChisel( HDC pHdc,
HBITMAPMASK pBitmapMask, qrect*
pSrcRect, qrect* pDestRect, qbool pStretch )
```

Draws a chiseled bitmap to the given DC. This is typically used to draw disabled toolbar buttons.

- **pHdc** - Identifies the device/ into which the chiseled bitmap is drawn.
- **pBitmapMask** - This parameter must be a black and white bitmap which was created by GDIcreateBitmap (depth of 1).
- **pSrcRect** - If this parameter is NULL, the whole bitmap is drawn to the device, otherwise only the part of the bitmap enclosed by the rectangle is drawn.
- **pDestRect** - Specifies the destination coordinates for the bitmap local to the DC. The bottom and right edges of the rectangle are ignored if pStretch is qfalse.
- **pStretch** - If qtrue, the bitmap is stretched or shrunk to fit exactly into pDestRect, otherwise the bitmap is drawn as is.

See also GDIdrawBitmap

GDIdrawComboButton()

```
void GDIdrawComboButton( HDC pHdc,
qrect* pRect, qulong pDrawFlags )
```

Draws a button in the style as used by the platforms combo box.

Warning: On return the rectangle will be inset according to the border sizes of the button face, i.e. the result can be used as the boundary for the text of the button.

- **pHdc** - Identifies the device in which the button will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries of the pushbutton.
- **pDrawFlags** - Specifies one or more of the following:
  GDI_BUTT_HIGHLIGHT
  The button is drawn depressed. The light and dark shades are reversed.
  GDI_BUTT_NOCOMBOARROW
  If specified the button arrow is not drawn.
  GDI_BUTT_DISABLE
  If specified the button is drawn disabled.
  GDI_COMBO_ARROWDOWN
  If specified the combo arrow is drawn pointing down.
  GDI_COMBO_ARROWUP
  If specified the combo arrow is drawn pointing up.
  GDI_COMBO_ARROWLEFT
  If specified the combo arrow is drawn pointing left.
  GDI_COMBO_ARROWRIGHT
  If specified the combo arrow is drawn pointing right.
  GDI_COMBO_WIN95_STYLE
  If specified a WIN95 style combo button is drawn (all WIN platforms).
  GDI_COMBO_97_STYLE
  If specified a WIN97 style button face is drawn (all platforms).
  GDI_COMBO_97_STYLE_HILITED
  If specified a WIN97 style button with a highlighted border is drawn (all platforms).

Example:

See GDIdraw3DPushButton.

See also GDIdraw3DPushButton, GDIdrawHeadingButton, GDIdrawSystem3dPushButton, GDIdrawSystemPushButton, GDIinflate-ButtonRect, GDIinsetButtonRect

GDIdrawFocusRect()

void GDIdrawFocusRect(HDC pHdc, qrect* pRect)

Frames the specified rectangle using the appropriate focus rect pen pattern for the current platform.

- **pHdc** - Identifies the drawing device.
- **pRect** - Points to the qrect structure.

Example:

See GDIdraw3DPushButton, GDIinsetButtonRect.

See also GDIfillRect

GDIdrawHeadingButton()

void GDIdrawHeadingButton(HDC pHdc, qrect* pRect, qcol pBorderColor, qcol pFaceColor, qulong pDrawFlags)

Draws a button in the style as used in heading lists.

Warning: On return the rectangle will be inset according to the border sizes of the button face, i.e. the result can be used as the boundary for the text of the button.

- **pHdc** - Identifies the device in which the button will be drawn.
- **pRect** - Specifies the boundaries of the pushbutton.
- **pBorderColor** - Specifies the color to be used to draw the outer single border (or double thickness border for default buttons). If default colors are to be used, specify GDI_COLOR_QDEFAULT for this color.
- **pFaceColor** - Specifies the color to be used to draw the face of the button. If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pDrawFlags** - Specifies one or more of the following:
  GDI_BUTT_HIGHLIGHT
  The button is drawn depressed.
  GDI_BUTT_LISTHEADING
  Draws a list heading style button
  GDI_BUTT_NOSORT
  Only used with GDI_BUTT_LISTHEADING. Draws the button withpout a sort arrow.
  GDI_BUTT_SELECTED
  Only used with GDI_BUTT_LISTHEADING. Draws the button in selected state.
  GDI_BUTT_SORTUP
  Only used with GDI_BUTT_LISTHEADING. Draws the button with the sorted arrow pointing up. Otherwise the arrow points downwards.

Example:

See GDIdraw3DPushButton.

See also GDIdraw3DPushButton, GDIdrawComboButton, GDIdrawSystem3dPushButton, GDIdrawSystemPushButton, GDIinflateButtonRect, GDIinsetButtonRect

GDIdrawHPIXMAP()

```
void GDIdrawHPIXMAP( HDC pHdc,
HPIXMAP pHPIXMAP, qrect* pSrcRect, qrect*
pDestRect, qbool pStretch )
```

Draws the given HPIXMAP to the given device.

- **pHdc** - Identifies the device.
- **pHPIXMAP** - Specifies the HPIXMAP to be drawn.
- **pSrcRect** - The rectangle in pHPIXMAP which is to be drawn.
- **pDestRect** - Points to the destination rect within the device.
- **pStretch** - If qtrue, the HPIXMAP will be stretched to fit pDestRect.

See also GDIcreateHPIXMAP, GDIdeleteHPIXMAP, GDIdrawHPIXMAPchisel, GDIdrawHPIXMAPmask

GDIdrawHPIXMAPchisel()

```
void GDIdrawHPIXMAPchisel( HDC pHdc,
HPIXMAP pHPIXMAP, qrect* pDestRect, qcol
pTransparent = colWhite )
```

Draws the given HPIXMAP to the given device in a chiseled effect.

- **pHdc** - Identifies the device.
- **pHPIXMAP** - Specifies the HPIXMAP to be drawn.
- **pDestRect** - Points to the destination rect within the device. The HPIXMAP will be stretched to the rect if it does not match the size of the HPIXMAP.
- **pTransparent** - Specifies the color which is to be used for the transparent color. Pixels that match the color will not be drawn.

See also GDIcreateHPIXMAP, GDIdeleteHPIXMAP, GDIdrawHPIXMAP, GDIdrawHPIXMAPmask

GDIdrawHPIXMAPmask()

```
void GDIdrawHPIXMAPmask( HDC pHdc,
HPIXMAP pHPIXMAP, qrect* pDestRect, qcol
pTransparent = colWhite )
```

Draws the given HPIXMAP as a black and white bitmap to the given device. You can use this function to create a black and white mask from a color PIXMAP, by first selecting a black and white bitmap into the given dc and calling this function. All pixels in the given HPIXMAP which match pTransparent are drawn white, and all others are drawn black.

- **pHdc** - Identifies the device.
- **pHPIXMAP** - Specifies the HPIXMAP to be drawn.
- **pDestRect** - Points to the destination rect within the device. The HPIXMAP will be stretched to the rect if it does not match the size of the HPIXMAP.
- **pTransparent** - Specifies the color which is to be used for the transparent color.

See also GDIcreateHPIXMAP, GDIdeleteHPIXMAP, GDIdrawHPIXMAP, GDIdrawHPIXMAPchisel

GDIdrawIcon()

```
qbool GDIdrawIcon( HDC pHdc, qshort
pIconId, qrect* pRect, qbool pStretch, qbool
pCanDraw )
```

Draws the specified icon. The icon must exist in the Omnis resource fork/file. If an icon of the specified id can not be found the function returns false.

- **pHdc** - Identifies the device.
- **pIconId** - The id of the icon in the resources. Under MacOS it must be CICN, under Windows an ICON resource.
- **pRect** - Specifies the destination coordinates for the icon local to the DC. The bottom and right edges of the rectangle are ignored if pStretch is qfalse. If pStretch is qfalse the rect's bottom and right will have been updated to reflect the size of the icon.
- **pStretch** - If qtrue, the icon is stretched or shrunk to fit exactly into pRect, otherwise the icon is drawn as is.
- **pCanDraw** - If qtrue, the icon is drawn to the DC, otherwise only the pRect is updated to reflect the size of the icon.

See also GDIdrawBitmap, GDIdrawPicture

GDIdrawPicture()

```
void GDIdrawPicture( HDC pHdc, void*
pPictData, qlong pPictDataLen, qjst pJst,
qrect* pRect, qbool pStretch, qbool
pUsePalette )
```

Lets you draw various pictures and bitmaps, including Omnis color shared pictures.

- **pHdc** - Identifies the device in which the picture is drawn.
- **pPictData** - Address of the picture's data.
- **pPictDataLen** - Length of the picture's data in bytes.
- **pJst** - Specifies the justification of the picture within the destination rectangle.
- **pRect** - Specifies the destination coordinates for the picture local to the DC.
- **pStretch** - If qtrue, the picture is stretched or shrunk to fit exactly into pRect, otherwise the picture is drawn as is and positioned within pRect according to the justification.
- **pUsePalette** - If qtrue, the pictures own color palette is used to draw the picture (the devices color palette is altered). Otherwise the colors in the picture are mapped to the nearest colors supported by the device.

See also GDIdrawBitmap, GDIdrawIcon

GDIdrawSystem3dPushButton()

```
void GDIdrawSystem3dPushButton(HDC
pHdc, qrect* pRect, qcol pBorderColor, qcol
pFaceColor, qulong pDrawFlags)
```

Under WIN16 and WIN32, draws a standard pushbutton. Under MacOS, this function draws a standard rounded corner pushbutton in 3d, i.e. the rounded rectangle has light and dark shaded edges.

Warning: On return the rectangle will be inset according to the border sizes of the button face, i.e. the result can be used as the boundary for the text of the button.

- **pHdc** - Identifies the device in which the button will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries of the pushbutton.
- **pBorderColor** - Specifies the color to be used to draw the outer single border (or double thickness border for default buttons). If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pFaceColor** - Specifies the color to be used to draw the face of the button. If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pDrawFlags** - Specifies one or more of the following:

GDI_BUTT_ALPHA (Mac OSX only)
Tells the function to darken the push button by the given alpha value stored within the high byte of pDrawFlags.
GDI_BUTT_ALPHA_INTERVAL
The recommended interval for flashing
GDI_BUTT_ALPHA_MASK (0xFF000000)
The mask used for extracting the alpha value for darkening the button face.
GDI_BUTT_ALPHA_MAX
The maximum recommended alpha value.
GDI_BUTT_ALPHA_SHIFT
The value for shifting the alpha for storing in pDrawFlags.

Example:

```
// draw the button face and darken it by the maximum value
qulong alpha = GDI_BUTT_ALPHA_MAX;
qulong drawFlags = GDI_BUTT_DEFAULT | GDI_BUTT_ALPHA;
drawFlags |= ( alpha << GDI_BUTT_ALPHA_SHIFT );
GDIdrawSystem3dPushButton( hdc, &rect, borderCol, faceCol, drawFlags );
```

GDI_BUTT_DEFAULT

If specified under Windows the standard default button border is drawn, under MacOS the standard default button outline is drawn.

GDI_BUTT_HIGHLIGHT

The button is drawn depressed.

GDI_BUTT_HOT (v3.2 WindowsXP)

The button is drawn in hot mode, the mouse is hovering over the button.

GDI_BUTT_ROUND

Draws round system button

Example:

See GDIdraw3DPushButton.

See also GDIdraw3DPushButton, GDIdrawComboButton, GDIdrawHeadingButton, GDIdrawSystemPushButton, GDIinflateButtonRect, GDIinsetButtonRect

GDIdrawSystemPushButton()

void GDIdrawSystemPushButton(HDC pHdc, qrect* pRect, qcol pBorderColor, qpat pFacePat, qcol pFaceFColor, qcol pFaceBColor, qcol pTextColor, qulong pDrawFlags)

Under WIN16 and WIN32, draws the standard 3D pushbutton. Under MacOS, this function draws the border and face of a rounded standard pushbutton.

Warning: On return the rectangle will be inset according to the border sizes of the button face, i.e. the result can be used as the boundary for the text of the button.

- **pHdc** - Identifies the device in which the button will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries of the pushbutton.
- **pBorderColor** - Specifies the color to be used to draw the outer single border (or double thickness border for default buttons). If default colors are to be used specify GDI_COLOR_QDEFAULT for this color.
- **pFacePat** - Specifies the pattern to be used when filling the face of the button. (MacOS only, it is always solid fill on other platforms)
- **pFaceFColor** - Specifies the foreground color for the face pattern. If default colors are to be used specify GDI_COLOR_QDEFAULT for this color, and use a solid fill for the pattern.
- **pFaceBColor** - Specifies the background color for the face pattern. (MacOS only)
- **pTextColor** - Specifies the foreground color to be used for the face pattern when the button is highlighted, pressed. (MacOS only). Ideally this should be the color of the text on the button, and the text should subsequently be drawn using pFaceFColor.
- **pDrawFlags** - Specifies one or more of the following:
  GDI_BUTT_HIGHLIGHT
  The button is drawn depressed. Under Windows, the light and dark shades are reversed. Under MacOS the pTextColor color is used to draw the face, and the text should subsequently be drawn using the pFaceFColor.
  GDI_BUTT_DEFAULT
  If specified under WIN16 and WIN32 the standard default button border is drawn, under MacOS the standard default button outline is drawn.

Example:

See GDIdraw3DPushButton.

See also GDIdraw3DPushButton, GDIdrawComboButton, GDIdrawHeadingButton, GDIdrawSystem3dPushButton, GDIinflateButtonRect, GDIinsetButtonRect

GDIdrawText()

void GDIdrawText(HDC pHdc, qchar* pText, qshort pTextLen) void GDIdrawText(HDC pHdc, qdim pX, qdim pY, qchar* pText, qshort pTextLen, qjst pJst)

Draws the given text in the specified justification, starting at the current cursor position in the given device.

Draws the given text in the specified justification, starting at horizontal position pX and vertical position pY. GDIdrawText will use the selected HFONT and HPEN in the given device.

- **pHdc** - Identifies the device.
- **pX** - The horizontal position. The actual starting position of the text will depend on the justification which is specified by **pJst**.
- **pY** - The vertical position of the top of the text. Text is drawn below this coordinate.
- **pText** - The text to be drawn.
- **pTextLen** - The length of the text.
- **pJst** - The horizontal justification. See description of **qjst** for more detail.

Example:

```
str255 txt = str255( "Some text" );
GDImoveTo( theDC, 15, 20 );
GDIdrawText( theDC, &txt[1], txt[0] );
// Is the same as
GDIdrawText( theDC, 15, 20, &txt[1], txt[0], jstLeft );
```

See also GDIdrawTextJst, GDItextBox

GDIdrawTextFastAlpha() (v5.0)

```
qbool GDIdrawTextFastAlpha(HDC pHdc,
qoschar* pText, qshort pTextLen, qbool
pRightToLeft)
```

Draws pText to the specified device in the current text color, including the alpha component.

- **pHdc** - Identifies the device.
- **pText** – The text to be drawn, one character per qoschar position.
- **pTextLen** – The length of the text in characters.
- **pRightToLeft** – If qtrue, characters will be read from right to left.

GDIdrawTextJst()

```
qbool GDIdrawTextJst( GDIdrawTextStruct* pTextStruct )
```

This function is a more complex version of GDIdrawText. It supports embedded escape characters in the text string to be drawn, making it possible to draw the text in various styles and colors, at various tab positions and justifications, and to embed icons and bitmaps. The following byte constants can be embedded in the text string to cause these style changes:

txtEsc

this character must always be embedded and indicates to the function that the next 3 or more bytes will specify a style change, tab position, justification etc.

txtEscCol

this character can be embedded following the txtEsc character to change the text color of all subsequent text. Following the txtEscCol must be the qcol, specifying the new color. Use *memcpy* and *sizeof* to embed the qcol.

If pIsAscii is qtrue, the qcol should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '00FFFFFF' which is white.

txtEscSty

this character can be embedded following the txtEsc character to change the style of all subsequent text. Following the txtEscSty must be the qsty, specifying the new style. Use *memcpy* and *sizeof* to embed the qsty.

If pIsAscii is qtrue, the qsty should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '00000003' which is bold and italic.

txtEscLTab

this character can be embedded following the txtEsc character to specify a new left justified tab position. Following the txtEscLTab must be the qdim, specifying the new position. Use *memcpy* and *sizeof* to embed the qdim.

If pIsAscii is qtrue, the qdim should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '000000FF' specifies a left tab at position 255.

txtEscCTab

same as txtEscLTab, except it centers the next run of text around the new tab position.

Note: A subsequent txtEsc character will stop the text run and the next run will be drawn left justified unless specified otherwise.

If pIsAscii is qtrue, the qdim should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '000000C8' specifies a center tab at position 200.

txtEscRTab

same as txtEscLTab, except it draws the next run of text on the left of the new tab position (right justified text).

Note: A subsequent txtEsc character will stop the text run and the next run will be drawn left justified unless specified otherwise.

If pIsAscii is qtrue, the qdim should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '00000064' specifies a right tab at position 100.

txtEscIcon

This character can be embedded following the txtEsc character to specify the id and position of an icon to be drawn (the icon must be located in the Omnis resource fork/file; under MacOS it must be CICN, under Windows an ICON resource.). The id must be a *qshort* followed by a *qdim* for the position of the icon. Use *memcpy* and *sizeof* to embed the qshort and qdim.

If pIsAscii is qtrue, only the icon id should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '000007d0' will draw icon 2000 at the current position.

txtEscNextCol

this character is used in conjunction with an array of column positions specified by the **pColumnArray** parameter. It tells the function to move to the next column position specified by the array. The internal index into the array is incremented by one for every use of this escape.

txtEscBmp

this character is used to specify the id of a bitmap to be drawn. The bitmap must exist in the Omnis bitmap data file. The id must be a *qlong*. Use *memcpy* and *sizeof* to embed the qlong.

If pIsAscii is qtrue, the bitmap id should be embedded as an 8 byte HEX string followed by txtAsciiEnd, i.e. '00000bb8' will draw bitmap 3000 at the current position.

txtEscBmpCentreLarge

same as txtEscBmp except that the bitmap is center justified (instead of left justified for txtEscBmp) at the current position assuming a bitmap size of 32 by 32 pixels.

txtEscBmpHandle

same as txtEscBmpCentreLarge except that an HBITMAP handle is expected instead of an id.

- **pTextStruct** – This structure contains the following information:
  mHdc Identifies the device.
  mX The horizontal position. The initial position of the text will depend on the justification which is specified by the *mJst* member of *pTextSpec*.
  mY The vertical position of the top (NOT the bottom) of the text.
  mText The text to be drawn.
  mTextLen The length of the text.
  mTextSpec Is a structure specifying the font and initial text style and justification.
  mColumnArray Specifies an array of column positions which are used in conjunction with the txtEscNextCol escape character.
  mColumnCount Specifies the number of array entries in mColumnArray.
  mIsAscii If true, all data embedded for the various escapes must be embedded as 8 byte hex strings followed by txtAsciiEnd.
  mApp Pointer to the Omnis library. Required for library based info, i.e. bitmaps.
  mColumnJsts Array of justifications for mColumnArray
  mFontHdc HDC to be used for creating HFONTs
- **return** - Returns false if errors occurred during the drawing of the text. This is usually caused by incorrect escape sequences.

Example:

```
// first we need a function to insert a style change into a string
void addStyle( qchar pStyle, qulong pValue, str255& pTxt )
{
// first add the escape and style chars
qshort len = pTxt[0];
pTxt[ len + 1 ] = txtEsc;
pTxt[ len + 2 ] = pStyle;
// now add the long value starting at the low nibble
register qchar lookup[16] = { '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F' };
register qchar* add = &pTxt[ len + 10 ];
for ( qshort cnt = 1 ; cnt <= 8 ; cnt++ )
{
```

```
*add = lookup[ pValue & 0x0000000F ];
add--;
pValue = pValue >> 4;
}
// add the ascii end char
pTxt[ len + 11 ] = txtAsciiEnd;
pTxt[0] = len + 11;
}
// prepare the multi style text string
str255 txt = str255("Normal Text ");
addStyle( txtEscSty, styBold, txt );
txt.concat( str255("Bold Text ") );
addStyle( txtEscSty, styItalic, txt );
txt.concat( str255("Italic text " ) );
addStyle( txtEscSty, styPlain, txt );
addStyle( txtEscCol, GDI_COLOR_QRED, txt );
txt.concat( str255("Red Text" ) );
// prepare the text structure
GDIdrawTextStruct tInfo;
tInfo.mHdc = tInfo.mFontHdc = theDc;
tInfo.mX = 10; tInfo.mY = 20;
tInfo.mText = &txt[0]; tInfo.mTextlen = txt[1];
tInfo.mTextSpec = GDItextSpecStruct(fntSystem, styPlain, colBlack, jstLeft);
tInfo.mColumnArray = 0; tInfo.mColumnJst = 0; tInfo.mColumnCount = 0;
tInfo.mIsAscii = qtrue;
tInfo.mApp = ECOgetApp( eci->mInstLocp );
// draw the text
GDIdrawTextJst( &tInfo );
```

See also GDItextWidthJst, GDIdrawText, GDItextBox

GDIendText() (v5.0)

void GDIendText(HDC pHdc,GDItextSpecStruct* pTextSpec)

Ends a text drawing operation.

- **pHdc** – Identifies the drawing device.
- **pTextSpec** – Returns GDItextSpecStruct information about the text .

Example: See GDIstartText()

GDIequalRect()

qbool GDIequalRect(qrect* pRect1, qrect* pRect2)

Returns qtrue if both specified rectangles are identical.

- **pRect1** - Points to the first rectangle to be compared.
- **pRect2** - Points to the second rectangle to be compared.

Example:

```
qrect rect1( 10, 15, 25, 40 );
qrect rect2( 12, 13, 80, 30 );
if ( GDIequalRect( &rect1, &rect2 ) )
{
  // there is something wrong here
}
```

See also GDIequalRgn

GDIequalRgn()

qbool GDIequalRgn(qrgn* pRgn1, qrgn* pRgn2)

Returns an indication of whether two Regions are identical in size shape and location.

- **pRgn1** - Points to region one to be compared.
- **pRgn2** - Points to region two to be compared.
- **return** - Returns qtrue if both regions are identical.

See also GDIequalRect

GDIexcludeClipRect()

void GDIexcludeClipRect(HDC pHdc, qrect* pRect)

Subtracts the specified rectangle from the clipping region of the given device, effectively disabling drawing within the specified rectangle.

- **pHdc** - Identifies the device for which the clipping will be cleared.
- **pRect** - Points to a qrect structure specifying the rectangular area to be excluded.

See also GDIexcludeClipRgn, GDIunionClipRect, GDIunionClipRgn

GDIexcludeClipRgn()

void GDIexcludeClipRgn(HDC pHdc, qrgn* pRgn)

Subtracts the specified region from the clipping region of the given device, effectively disabling drawing within the specified region.

- **pHdc** - Identifies the device for which the clipping will be cleared.
- **pRect** - Points to a qrgn structure specifying the region to be excluded.

See also GDIexcludeClipRect, GDIunionClipRect, GDIunionClipRgn

GDIfillEllipse()

void GDIfillEllipse(HDC pHdc, qrect* pRect, HBRUSH pBrush)

Fills an elliptical shape within the specified rectangle, using the color and pattern of the specified brush and the current back color.

- **pHdc** - Identifies the device in which the shape will be drawn.
- **pRect** - Point to the qrect which specifies the boundaries for the ellipse.
- **pBrush** - Specifies the brush to be used for the fill.

Example:

See GDIdragBitmapMove.

See also GDIframeEllipse, GDIfillRect, GDIfillRoundRect, GDIfillPoly, GDIfillRgn, GDIfloodFill

GDIfillPoly()

void GDIfillPoly(HDC pHdc, qpoint* pPoints, qshort pNumPoints)

Fills the specified polygon (array of qpoints) using the color and pattern of the current brush, and the current background color.

- **pHdc** - Identifies the device in which the polygon is filled.
- **pPoints** - Points to an array of qpoints which specify the polygon shape.
- **pNumPoints** - Specifies the number of qpoints in the pPoints array.

Example:

```
// this example frames and fills a polygon with patterned pen and brush
// create the polygon points ( diamond shape )
qpoint pts[5];
pts[0].h = pts[0].v = 0;
pts[1].h = pts[1].v = 10;
pts[2].h = 0; pts[2].v = 20;
pts[3].h = -10; pts[3].v = 10;
pts[4].h = pts[4].v = 0;
// fill the polygon
GDIsetBkColor( theDC, GDI_COLOR_QRED );
GDIsetTextColor( theDC, GDI_COLOR_QYELLOW );
HBRUSH brush = GDIcreateBrush( patStd10 );
brush = GDIselectObject( theDC, brush );
GDIfillPoly( theDC, &pts[0], 5 );
brush = GDIselectObject( theDC, brush );
GDIdeleteObject( brush );
// frame the poly
GDIsetBkColor( theDC, GDI_COLOR_QGREEN );
HPEN pen = GDIcreatePen( 1, GDI_COLOR_QBLUE, patPen8 );
pen = GDIselectObject( theDC, pen );
GDIframePoly( theDC, &pts[0], 5 );
pen = GDIselectObject( theDC, pen );
GDIdeleteObject( pen );
```

See also GDIframePoly, GDIfillRect, GDIfillRgn, GDIfloodFill

GDIfillRect()

```
void GDIfillRect( HDC pHdc, qrect* pRect, HBRUSH pBrush )
```

Fills the specified rectangle using the color and pattern of the specified brush and the current back color.

- **pHdc** - Identifies the device in which the rectangle will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries for the fill.
- **pBrush** - Specifies the brush to be used for the fill

Example:

See GDIdragBitmapMove.

See also GDIframeRect, GDIfillRoundRect, GDIfillRgn, GDIfillEllipse, GDIfillPoly, GDIfloodFill

GDIfillRgn()

```
void GDIfillRgn( HDC pHdc, qrgn* pRgn, HBRUSH pBrush )
```

Fills all pixels enclosed by a region with the color and pattern of the specified brush, and the current background color. The background color is used for all white pixels in the pattern of the brush.

- **pHdc** - Identifies the device in which the region is filled.
- **pRgn** - Points to the region to be filled.
- **pBrush** - Specifies the brush to be used.

Example:

See GDIcreatePoly, GDIcreateRectRgn, GDIcreateRoundRectRgn.

See also GDIframeRgn, GDIfillRect, GDIfillPoly, GDIfloodFill

GDIfillRoundRect()

```
void GDIfillRoundRect( HDC pHdc, qrect*
pRect, qdim pWidthEllipse, qdim
pHeightEllipse, HBRUSH pBrush )
```

Fills the specified rounded corner rectangle using the color and pattern of the specified brush and the current back color. The degree of rounding depends on the values passed in pWidthEllipse and pHeightEllipse.

- **pHdc** - Identifies the device in which the rounded rectangle will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries for the fill.
- **pWidthEllipse** - Width defining curvature of corners.
- **pHeightEllipse** - Height defining curvature of corners.
- **pBrush** - Specifies the brush to be used for the fill.

Note: This function is like filling a rectangle and drawing four identical ellipses to replace the rectangles corners:

See also GDIframeRoundRect, GDIfillRect, GDIfillRgn, GDIfillEllipse, GDIfillPoly, GDIfloodFill

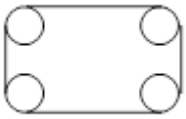

Figure 2:

GDIfloodFill()

void GDIfloodFill(HDC pHdc, qdim pX, qdim pY, qcol pCol)

Fills an area in the device using the current brush. It begins at the coordinate specified by pX and pY and continues in all directions, filling all adjacent areas containing the color of the starting position with the color specified by pCol.

- **pHdc** - Identifies the device.
- **pX** - Specifies the horizontal coordinate of the starting position.
- **pY** - Specifies the vertical coordinate of the starting position.
- **pCol** - Specifies the color to be used for the fill.

Example:

```
// this example draws an enclosed odd space using lines
// and fills that shape using flood fill.
GDImoveTo( theDC, 20, 20 );
GDIlineTo( theDC, 35, 25 );
GDIlineTo( theDC, 20, 40 );
GDIlineTo( theDC, 15, 38 );
GDIlineTo( theDC, 17, 30 );
GDIlineTo( theDC, 20, 20 );
// make sure the point is within the shape
GDIfloodFill( theDC, 20, 22, GDI_COLOR_QDKRED );
```

See also GDIframeRoundRect, GDIfillRect, GDIfillRgn, GDIfillEllipse, GDIfillPoly, GDIfillRoundRect

GDIflushDC() (v3.1)

void GDIflushDC(HDC pHdc)

On Mac OSX and Linux, drawing occurs inside buffers. Changes do not appear on screen until these buffers are flushed to the screen. Flushing of window buffers will naturally occur when the process is less busy, so you should not need to call this function. The only time you should call these function is, if you animating some screen drawing from within a tight loop.

- **pHdc** - Identifies the device to be flushed. On Mac OSX only the Macintosh window owning the DC is flushed. On Linux these parameter is ignored, and all window buffers are flushed.

See also GDIsetFlush

GDIfontDecSize()

```
qbool GDIfontDecSize( qfnt* pFnt )
```

Decrements the size of the given font to the next smaller possible size for that font. If the font size can not be reduced any further, the function returns qfalse.

- **pFnt** - Points to the qfnt.
- **return** - Returns qtrue if successful.

See also GDIfontIncSize

GDIfontGetExtra()

```
qshort GDIfontGetExtra( qfnt* pFnt )
```

Returns the extra spacing of the given font.

- **pFnt** - Points to the qfnt.
- **return** - Returns extra spacing in points.

See also GDIfontSetExtra, GDIfontGetSize, GDIfontSetSize

GDIfontGetSize()

```
qshort GDIfontGetSize( qfnt* pFnt )
```

Returns the size of the font in points.

- **pFnt** - Points to the qfnt.
- **return** - Returns the font size in points.

See also GDIfontSetSize, GDIfontGetExtra, GDIfontSetExtra

GDIfontHeight()

```
qdim GDIfontHeight( qfnt* pFnt ) qdim
GDIfontHeight( HDC pHdc, qshort pExtra = 0
)
```

Returns the total of ascent, descent, leading and extra spacing of the font in screen units (this is effectively the line height of a font, and can be used in multi line text based controls).

- **pFnt** - Points to the qfnt.

OR

- **pHdc** - Points to the device which has the HFONT of interest selected.

- **pExtra** - Additional extra spacing in points to be added to the font height. pExtra has a default value of zero and does not need to be specified.
- **return** - Returns the font height in screen units.

Example:

```
HFONT fnt = GDIcreateFont( &fntSystem, styPlain );
fnt = GDIselectObject( theDC, fnt );
qdim ht = GDIfontHeight( theDC );
fnt = GDIselectObject( theDC, fnt );
GDIdeleteObject( fnt );
```

See also GDIfontPart, GDIfontGetSize, GDIfontGetExtra

GDIfontIncSize()

qbool GDIfontIncSize(qfnt* pFnt)

Increments the size of the specified font to the next greater possible size for that font. If the font size can not be increased any further, the function returns qfalse.

- **pFnt** - Points to the qfnt.
- **return** - Returns qtrue if successful.

See also GDIfontDecSize

GDIfontIsReal()

qbool GDIfontIsReal(qfnt* pFnt) qbool GDIfontIsReal(HDC pHdc)

Returns qfalse if the given font is valid (i.e. the font exists in the size as specified by the qfnt or selected HFONT).

- **pFnt** - Points to the qfnt.

OR

- **pHdc** - Points to the device which has the HFONT of interest selected.
- **return** - returns qtrue if the font and font size are valid.

See also GDIfontIsTrueType

GDIfontIsTrueType()

qbool GDIfontIsTrueType(qfnt* pFnt, qbool pIsPrinter) qbool GDIfontIsTrueType(HDC pHdc, qbool pIsPrinter)

Returns qtrue if the given font is a True Type font. If **pIsPrinter** is true, the specified font is checked against the printer fonts.

- **pFnt** - Points to the qfnt.

OR

- **pHdc** - Points to the device which has the HFONT of interest selected.
- **pIsPrinter** - If qtrue is specified, the font is checked against the printer fonts.
- **return** - returns qtrue if the font is a True Type font.

See also GDIfontIsReal

GDIfontPart()

qdim GDIfontPart(qfnt* pFnt, qsty pSty, eGDIfontPart pFontPart) qdim GDIfontPart(HDC pHdc, eGDIfontPart pFontPart)

Returns the dimension of given font part for the given font and style.

- **pFnt** - Specifies the font information.
- **pSty** - Specifies the font style.

OR

- **pHdc** - Points to the device which has the HFONT of interest selected.
- **pFontPart** - Specifies the font part of interest. One of the following font parts can be requested:
  eFontAscent
  the ascent of the font.
  eFontDescent
  the descent of the font.
  eFontLeading
  the leading of the font.
  eFontHeight
  the total of the ascent and descent of the font.
  eFontLineHeight
  the total of the ascent, descent and leading of the font.
  eFontMaxWidth
  the width of the widest character of the given font and style.
- **return** - returns the size of the font part in screen units.

Example:

See GDIinflateButtonRect.

See also GDIfontHeight, GDItextWidth

GDIfontSetExtra()

void GDIfontSetExtra(qfnt* pFnt, qshort pExtra)

Sets the extra spacing of the given font.

- **pFnt** - Points to the qfnt.
- **pExtra** - Specifies the extra spacing in points.

See also GDIfontGetExtra, GDIfontSetSize

GDIfontSetSize()

void GDIfontSetSize(qfnt* pFnt, qshort pSize)

Sets the size in points of the font.

- **pFnt** - Points to the qfnt.
- **pSize** - Specifies the new size of the font.

See also GDIfontGetSize, GDIfontGetExtra, GDIfontSetExtra

GDIframeEllipse()

void GDIframeEllipse(HDC pHdc, qrect* pRect)

Frames an elliptical shape within the specified rectangle, using the current pen.

- **pHdc** - Identifies the device in which the shape will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries for the ellipse.

Example:

```
// this example creates a thick black pen and draws an Ellipse
qrect theRect( 10, 10, 50, 100 );
HPEN pen = GDIcreatePen( 4 );
pen = GDIselectObject( theDC, pen );
GDIframeEllipse( theDC, &theRect );
pen = GDIselectObject( theDC, pen );
GDIdeleteObject( pen );
```

See also GDIfillEllipse, GDIframeRect, GDIframeRoundRect, GDIframePoly, GDIframeRgn

GDIframePoly()

```
void GDIframePoly( HDC pHdc, qpoint* pPoints, qshort pNumPoints )
```

Frames the specified polygon (array of qpoints) using the color and line style of the current pen.

- **pHdc** - Identifies the device in which the polygon is framed.
- **pPoints** - Points to an array of qpoints which specify the polygon shape.
- **pNumPoints** - Specifies the number of qpoints in the pPoints array.

Example:

See GDIfillPoly.

See also GDIfillPoly, GDIframeRect, GDIframeRgn

GDIframeRect()

```
void GDIframeRect( HDC pHdc, qrect* pRect )
```

Frames the specified rectangle using the current pen.

- **pHdc** - Identifies the device in which the rectangle will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries for the frame.

Note: The frame is drawn inside (including the boundaries) the specified rectangle regardless of the pen width.

See also GDIfillRect, GDIframeEllipse, GDIframeRoundRect, GDIframePoly, GDIframeRgn

GDIframeRgn()

```
void GDIframeRgn( HDC pHdc, qrgn* pRgn, HBRUSH pBrush )
```

Frames the specified region using the color and pattern of the specified brush. All white pixels in the brush will be transparent.

- **pHdc** - Identifies the device in which the region is framed.
- **pRgn** - Points to the region.
- **pBrush** - Specifies the brush to be used.

Example:

See GDIcreatePoly.

See also GDIfillRgn, GDIframeRect, GDIframePoly

GDIframeRoundRect()

```
void GDIframeRoundRect( HDC pHdc, qrect*
pRect, qdim pWidthEllipse, qdim
pHeightEllipse )
```

Frames the specified rounded corner rectangle using the current pen. The degree of rounding depends on the values passed in pWidthEllipse and pHeightEllipse.

- **pHdc** - Identifies the device in which the rounded rectangle will be drawn.
- **pRect** - Points to the qrect which specifies the boundaries for the frame.
- **pWidthEllipse** - Width defining curvature of corners.
- **pHeightEllipse** - Height defining curvature of corners.

Note: The frame is drawn inside (including the boundaries) the specified rectangle regardless of the pen width. For more information on the curvature see GDIfillRoundRect.

See also GDIfillRoundRect, GDIframeEllipse, GDIframeRect, GDIframePoly, GDIframeRgn

GDIgetBkColor()

```
qcol GDIgetBkColor( HDC pHdc )
```

Returns the current background color from the given device.

- **pHdc** - Identifies the device.
- **return** - Returns the current background color.

See also GDIgetTextColor, GDIsetBkColor, GDIsetTextColor

GDIgetBkColorAlpha() (v5.0)

```
qbyte GDIgetBkColorAlpha( HDC pHdc )
```

Returns the alpha component of the device's background color.

- **pHdc** - Identifies the device.

GDIgetClipRect()

```
void GDIgetClipRect( HDC pHdc, qrect* pRect )
```

Returns the bounding rectangle of the current clipping region of the given device.

- **pHdc** - Identifies the device for which to return the current clipping.
- **pRect** - Points to a qrect structure which is to receive the coordinates of the bounding rectangle for the current clipping.

See also GDIgetClipRgn, GDIsetClipRect, GDIsetClipRgn

GDIgetClipRgn()

```
void GDIgetClipRgn( HDC pHdc, qrgn* pRgn )
```

Returns a copy of the clipping region of the given device.

- **pHdc** - Identifies the device for which to return the current clipping.
- **pRgn** - Points to a qrgn structure which is to receive a copy of the current clipping region.

See also GDIgetClipRect, GDIsetClipRect, GDIsetClipRgn

GDIgetColorEntries()

```
qshort GDIgetColorEntries( HPIXMAP
pPixMap, qshort pStart, qshort pCount,
qColorEntry* pEntries )
```

Get the color entries of a HPIXMAP. Returns the number of colors actually retrieved from the HPIXMAP.

- **pPixMap**- Identifies the HPIXMAP.
- **pStart**- The starting place to get color entries.
- **pCount**- The number of color entries to retrieve.
- **pEntries**- Points to the buffer of color entries to stored.

GDIgetDarkerShade()

```
qcol GDIgetDarkerShade( qcol pCol )
```

Returns a darker shade of the specified color by halving the individual RGB values. The RGB values are shifted down by one bit, shifting in zeros at the high bit end.

- **pCol** - Specifies the color for which to return a darker shade. The qcol can contain a system color index or direct RGB values.
- **return** - Returns a darker shade of the given color.

See also GDIgetLighterShade

GDIgetFlagsForAlphaDC() (v5.0)

```
qulong GDIgetFlagsForAlphaDC(HDC pHdc)
```

Returns the flags used to create the alpha drawing device.

- **pHdc** - Identifies the device.

See also GDIsetFlagsForAlphaDC

GDIgetFontName()

qshort GDIgetFontName(qfnt* pFnt, qchar *pBuffer, qshort pMaxLen)

Returns the name of the font as a text string.

- **pFnt** - Points to the qfnt.
- **pSize** - Points to the buffer for the font name.
- **pMaxLen** - Specifies the size of the given buffer which should be a minimum size of 32 bytes.
- **return** - Returns the length of the returned font name.

Example:

```
str255 fntName;
fntName[0] = GDIgetFontName(&fntSystem, &fntName[1], 255 );
```

See also GDIsetFontName

GDIgetHPIXMAPinfo()

void GDIgetHPIXMAPinfo(HPIXMAP pPixMap, HPIXMAPinfo* pPixMapInfo)

Gets Information about a particular HPIXMAP.

- **pPixMap-** Identifies which HPIXMAP information is required.
- **pPixMapInfo**- Points to structure to store the returned information.

GDIgetLighterShade()

qcol GDIgetLighterShade(qcol pCol)

Returns a lighter shade of the specified color by multiplying the individual RGB values by two. The RGB values are shifted up by one bit, shifting in set bits at the low bit end.

- **pCol** - Specifies the color for which to return a lighter shade. The qcol can contain a system color index or direct RGB values.
- **return** - Returns a lighter shade of the given color.

See also GDIgetDarkerShade

GDIgetMenubarHeight()

qdim GDIgetMenubarHeight()

Return the height of the menu bar in screen units.

GDIgetNativeGraphicsObjectForAlphaDC() (v5.0)

void *GDIgetNativeGraphicsObjectForAlphaDC(HDC pHdc)

Returns Graphics pointer to GDI+ object on Win32, or CGContextRef on MacOSX; returns NULL if the HDC is not an alphaDC. Normally used in detecting an alpha capable device.

- **pHdc** – Identifies the device.

Example:

```
qbool alphaDC = qfalse;
#if defined(iswin32) && !defined(isunix)
alphaDC = (GDIgetNativeGraphicsObjectForAlphaDC(pHDC) != 0);
#endif
```

GDIgetPaletteEntries()

qshort GDIgetPaletteEntries(HPALETTE pPalette, qshort pStart, qshort pCount, qColorEntry* pEntries)

GDIgetPaletteEntries retrieves a range of palette entries in given HPALETTE.

- **pPalette** - Identifies the HPALETTE.
- **pStart** - The starting position.
- **pCount** - The number of color entries to retrieve.
- **pEntries** - The buffer for the color entries to be stored in.

GDIgetPixel()

qcol GDIgetPixel(HDC pHdc, qdim pX, qdim pY)

Retrieves the color of the screen pixel at the specified location in the given device.

- **pHdc** - Identifies the device.
- **pX** - Specifies the horizontal coordinate.
- **pY** - Specifies the vertical coordinate.
- **return** - Returns the qcol of the specified screen pixel.

See also GDIsetPixel

GDIgetRealColor()

void GDIgetRealColor(qcol& pCol)

Converts a system color index to a real RGB value. If the qcol already contains a RGB value, it does nothing. There should never be a need to call this function from outside the GDI module.

- **pCol** - Is a reference to the qcol which is to be converted.

GDIgetRgnBox()

```
void GDIgetRgnBox( qrgn* pRgn, qrect* pRect )
```

Returns the bounding rectangle of the given region.

- **pRgn** - Points to the region.
- **pRect** - Points to the qrect which is set to receive the bounding rectangle.

GDIgetScreenRect()

```
void GDIgetScreenRect( qulong pFlags, qrect* pRect )
```

Under Windows, returns the Omnis program window client rect (or the screen rect if GDI_SCR_GET_ALL is specified) after subtracting the specified restrictions. Under MacOS, this function returns the main monitor's screen rect (or the union of all connected monitors if GDI_SCR_GET_ALL is specified) after subtracting the specified restrictions.

Note: Using any of the GDI_SCR_SUB_xxx flags together with GDI_SCR_GET_ALL does not work very well. The resulting rect may be somewhat incorrect.

- **pFlags** - Specifies the areas to be subtracted prior to returning the main monitors screen rect. The following flags can be specified:
  GDI_SCR_SUB_TOP
  Subtracts height of top toolbar
  GDI_SCR_SUB_BOT
  Subtracts height of bottom toolbar and helpbar
  GDI_SCR_SUB_LEFT
  Subtracts width of left toolbar
  GDI_SCR_SUB_RIGHT
  Subtracts width of right toolbar
  GDI_SCR_SUB_ALL
  Subtracts all of the above
  GDI_SCR_GET_ALL
  Returns the union of all connected monitors if specified. MacOS Only.
- **pRect** - The screen rect is returned in this parameter.

See also GDIptInScreen

GDIgetScreenResolution()

```
void GDIgetScreenResolution( qdim* pHorz, qdim* pVert )
```

Returns the pixels per inch for the current graphics mode.

- **pHorz** - Horizontal pixels per inch.
- **pVert** - Vertical pixels per inch.

GDIgetStockBrush()

HBRUSH GDIgetStockBrush(qlong pWhich)

Returns an HBRUSH from stock for drawing.

Warning: The HBRUSH must NOT be deleted when finished with.

- **pWhich** - Specifies one of the following values:
  BLACK_BRUSH
  Returns a solid brush (the area field with the brush will be filled with the current textcolor).
  DKGRAY_BRUSH
  Returns a dark gray pattern brush.
  GRAY_BRUSH
  Returns a gray pattern brush.
  LTGRAY_BRUSH
  Returns a light gray pattern brush.
  WHITE_BRUSH
  Returns a empty brush (the area field with the brush will be filled with the current backcolor).
- **return** - returns the specified stock brush.

Example:

See GDIcreateRectRgn.

See also GDIgetStockPen, GDIcreateBrush

GDIgetStockPen()

HBRUSH GDIgetStockPen(qlong pWhich)

Returns an HPEN from stock for drawing. All pens returned have a thickness of 1 pixel and a solid line style.

Warning: The HPEN must NOT be deleted when finished with.

- **pWhich** - Specifies one of the following values:
  BLACK_PEN
  Returns a black solid pen.
  WHITE_PEN
  Returns a white solid pen.
- **return** - returns the specified stock pen.

See also GDIgetStockBrush, GDIcreatePen

GDIgetTempDC()

HDC GDIgetTempDC()

Returns a general purpose screen DC. It can be used for all non-drawing operations which require a screen dc. In some cases it is more efficient to use this temp DC, i.e. when it is required to calculate the text width for several strings of text, based on the same font.

Warning: The temp DC must never be deleted, and it is not available to call other functions which may also use the temp DC, while it is being used in the calling function.

- **return** - Returns the temp DC.

Example:

See GDIcopyBits, GDIinflateButtonRect.

See also GDIcreateScreenDC

GDIgetTextColorAlpha() (v5.0)

```
qbyte GDIgetTextColorAlpha( HDC pHdc )
```

- **pHdc** - Identifies the device.

GDIgetTextColor()

```
qcol GDIgetTextColor( HDC pHdc )
```

Returns the current text color from the given device.

- **pHdc** - Identifies the device.
- **return** - Returns the current text color.

See also GDIgetBkColor, GDIsetBkColor, GDIsetTextColor

GDIgetViewportOrg()

```
void GDIgetViewportOrg ( HDC pHdc, qdim* pX, qdim* pY )
```

Retrieves the given devices horizontal and vertical origin.

- **pHdc** - Identifies the device.
- **pX** - Points to the qdim which is to receive the horizontal origin.
- **pY** - Points to the qdim which is to receive the horizontal origin.

See also GDIsetViewportOrg

GDIhasAlphaSupport()

```
qbool OMNISAPI GDIhasAlphaSupport()
```

Returns a boolean value indicating whether the current platform supports alpha-blended images.

GDIhiliteBitmap()

```
void GDIhiliteBitmap( HDC pHdc,
HBITMAPMASK pBitmapMask, qrect
*pSrcRect, qrect *pDestRect, qcol pHiliteCol )
```

Highlights a bitmap by setting every other pixel in the non-transparent parts of the bitmap to the supplied color.

- **pHdc** - Identifies the device in which the bitmap to be highlighted has already been drawn.
- **pBitmapMask** - The handle to the bitmap mask. If this is NULL, the whole bitmap is highlighted.
- **pSrcRect** - The rectangle in pBitMapMask which is to be drawn.
- **pDestRect** - The rectangle in pHdc which is to be highlighted.
- **pHiliteCol** - the color to be used to highlight the bitmap.

See also GDIdrawBitmap

GDIhiliteRect()

```
void GDIhiliteRect( HDC pHdc, qrect* pRect )
```

Highlights the specified rectangle using the systems highlight color and mode.

- **pHdc** - Identifies the device in which the rectangle will be highlighted.
- **pRect** - Points to the qrect which specifies the boundaries for the highlight.

See also GDIinvertRect

GDIhiliteRgn()

```
void GDIhiliteRgn( HDC pHdc, qrgn* pRgn )
```

Highlights the specified region using the systems highlight color and mode.

- **pHdc** - Identifies the device in which the region will be highlighted.
- **pRgn** - Points to the region to be highlighted.

See also GDIinvertRgn

GDIhiliteTextEnd() (v3.1)

```
void GDIhiliteTextEnd( HDC pHdc, qrect*
pRect, qcol pTheTextColor )
```

For full description see GDIhiliteTextStart below.

GDIhiliteTextStart() (v3.1)

```
void GDIhiliteTextStart( HDC pHdc, qrect*
pRect, qcol pTheTextColor )
```

With the numerous platforms now supported by Omnis and the various ways in which text is hilited on the various platforms, we have introduced GDIhiliteTextStart and GDIhiliteTextEnd. When you need to hilite some of your text, you simply call GDIhiliteTextStart, draw your text, than call GDIhiliteTextEnd. You must have calculated your texts bounding rectangle prior. When GDIhiliteTextStart returns, the correct text color will have been selected in the DC, ready for you to draw your text.

- **pHdc** - Identifies the device in which the text will be highlighted.
- **pRect** - Your texts bounding rect.
- **pTheTextColor** - The color of the text (if it wasn't hilited).

Example:

```
GDIhiliteTextStart( theDC, &theRect, theTextColor );
GDIdrawText(theDC, theRect.left, theRect.top, theText, theTextLen, jstLeft);
GDIhiliteTextEnd( theDC, &theRect, theTextColor );
```

See also GDIhiliteTextEnd

GDIHPIXMAPfromSharedPicture()

```
HPIXMAP GDIHPIXMAPfromSharedPicture(
void* pPictData, qlong pPictDataLen )
```

Creates an HPIXMAP from an Omnis shared Picture.

- **pPictData** - Address of the pictures data.
- **pPictDataLen** - Length of the pictures data in bytes.
- **return** - Returns the HPIXMAP. The caller is responsible for deleting the HPIXMAP when it is no longer needed.

See also GDIcreateHPIXMAP, GDIdeleteHPIXMAP

GDIinflateButtonRect()

```
void GDIinflateButtonRect( qrect* pRect, qshort pFaceType )
```

Inflates the given rectangle by the border sizes of to the specified button face style. This is the reverse of what the above button drawing functions do to their pRect parameter. This function is useful when you need to calculate the overall size of a pushbutton based on its text width, height and button face.

- **pRect** - Points to the rectangle to be inflated.
- **pFaceType** - Specifies one of the following constants:
  GDI_BUTT_FACE_SYS
  System Button Face (GDIdrawSystemPushButton).
  GDI_BUTT_FACE_SYS3D
  System 3D Button Face (GDIdrawSystem3dPushButton).
  GDI_BUTT_FACE_3D
  3D Button Face (GDIdraw3DPushButton).
  GDI_BUTT_FACE_HEADING
  Heading Button Face (GDIdrawHeadingButton).
  GDI_BUTT_FACE_COMBO
  Combo Button Face (GDIdrawComboButton).

Example:

```
// this example calculates the minimum height and width of a button based
// on the font, font size, font style.
str255 txt = str255("Button");
qrect cRect( 1, 1, 1, 1 );
// get the temp dc and select the font into it
HDC dc = GDIgetTempDC();
HFONT fnt = GDIcreateFont( &fntButt, styButt );
fnt = GDIselectObject( dc, fnt );
// calculate the width. Allow an extra 4 pixels (+8) at either end.
// It looks better.
cRect.right = GDItextWidth( dc, &txt[1], txt[0] ) + 8;
// calculate the height. Buttons center the ascent part of a font,
// so we need to add the
// descent twice, once for above the ascent part and once for below.
cRect.bottom = GDIfontPart( dc, eFontAscent ) + GDIfontPart( dc, eFontDescent ) * 2;
// now add the border of the button
GDIinflateButtonRect( &cRect, GDI_BUTT_FACE_SYS3D );
```

```
// now we have the minimum height and width.
qdim theHeight = cRect.height();
qdim theWidth = cRect.width();
// restore the font in the dc, and delete the one we created
fnt = GDIselectObject( dc, fnt );
GDIdeleteObject( fnt );
```

See also GDIinsetButtonRect

GDIinflateRect()

void GDIinflateRect(qrect* pRect, qdim pXAmt, qdim pYAmt)

Inflates the rectangle outwards by the specified amount. Passing negative values for the amounts will shrink the rectangle.

- **pRect** - Points to the qrect to be inflated.
- **pXAmt** - Specifies the amount to inflate the rectangle horizontally.
- **pYAmt** - Specifies the amount to inflate the rectangle vertically.

See also GDIinsetRect, GDIoffsetRect

GDIinsetButtonRect()

void GDIinsetButtonRect(qrect* pRect, qshort pFaceType)

Insets the given rectangle by the border sizes of the specified button face style. This function is useful when you need to calculate the size of a pushbutton inner face for drawing text or icons within the face.

- **pRect** - Points to the rectangle to be inset.
- **pFaceType** - Specifies one of the following constants.
  GDI_BUTT_FACE_SYS
  System Button Face (GDIdrawSystemPushButton).
  GDI_BUTT_FACE_SYS3D
  System 3D Button Face (GDIdrawSystem3dPushButton).
  GDI_BUTT_FACE_3D
  3D Button Face (GDIdraw3DPushButton).
  GDI_BUTT_FACE_HEADING
  Heading Button Face (GDIdrawHeadingButton).
  GDI_BUTT_FACE_COMBO
  Combo Button Face (GDIdrawComboButton).

Example:

```
// in this example a button has just received the focus and needs
// to draw a focus rect
HDC dc = WNDstartDraw( theHwnd );
qrect cRect;
WNDgetClientRect( theHwnd, &cRect );
GDIinsetButtonRect( &cRect, GDI_BUTT_FACE_SYS3D );
GDIdrawFocusRect( dc, &cRect );
WNDendDraw( theHwnd, dc );
```

See also GDIinflateButtonRect

GDIinsetRect()

```
void GDIinsetRect( qrect* pRect, qdim pXAmt, qdim pYAmt )
```

Insets (reverse of GDIinflateRect) the rectangle by the specified amount inwards. Passing negative values for the amounts will inflate the rectangle.

- **pRect** - Points to the qrect to be inset.
- **pXAmt** - Specifies the amount to inset the rectangle horizontally.
- **pYAmt** - Specifies the amount to inset the rectangle vertically.

See also GDIinflateRect, GDIoffsetRect

GDIintersectClipRect()

```
void GDIintersectClipRect( HDC pHdc, qrect* pRect )
```

Sets the current clipping region of the given device to the intersection of the clipping region and the specified rectangle, effectively disabling drawing outside the specified rectangle.

- **pHdc** - Identifies the device in which to intersect the clipping.
- **pRect** - Points to a qrect structure which specifies the rectangle to be intersected with the current clipping region.

See also GDIintersectClipRgn, GDIsetClipRect, GDIsetClipRgn

GDIintersectClipRgn()

```
void GDIintersectClipRgn( HDC pHdc, qrgn* pRgn )
```

Sets the current clipping of the given device to the intersection of the clipping region and the specified region, effectively disabling drawing outside the specified region.

- **pHdc** - Identifies the device in which to intersect the clipping.
- **pRgn** - Points to a qrgn structure which specifies the region to be intersected with the current clipping region.

See also GDIintersectClipRect, GDIsetClipRect, GDIsetClipRgn

GDIintersectRect()

```
qbool GDIintersectRect( qrect* pDestRect,
qrect* pSrcRect1, qrect* pSrcRect2 )
```

Returns the intersection of two rectangles in a third rectangle, and returns true if the resulting rectangle is not empty.

- **pDestRect** - Points to the destination rectangle for the intersection.
- **pSrcRect1** - Points to rectangle one to intersect.
- **pSrcRect2** - Points to rectangle two to intersect.

- **return** - returns true if the resulting rectangle is not empty.

Example:

```
qrect r1( 10, 10, 20, 20 );
qrect r2( 15, 15, 25, 25 );
qrect r3;
if ( GDIintersectRect( &r3, &r1, &r2 ) )
{
  // r3 should equal to 15, 15, 20, 20
}
```

See also GDIunionRect

GDIinvertRect()

```
void GDIinvertRect( HDC pHdc, qrect* pRect,
qbool pInvertBackground )
```

Inverts the specified rectangle in the specified device. All set pixels will be cleared and all clear pixels will be set.

- **pHdc** - Identifies the device in which the rectangle will be inverted.
- **pRect** - Points to the qrect which specifies the boundaries for the invert.
- **pInvertBackground** - When set to kTrue this will invert only the background of a transparent HWND within the specified pRect. This will cause those HWNDs which are visible in pRect to be redrawn and inverted. If set to kFalse this inverts only what has been drawn in the current HWND. This is used only on macOS and is ignored on other platforms.

See also GDIhiliteRect

GDIinvertRgn()

```
void GDIinvertRgn( HDC pHdc, qrgn* pRgn )
```

Inverts the specified region in the specified device. All set pixels will be cleared and all clear pixels will be set.

- **pHdc** - Identifies the device in which the region will be inverted.
- **pRgn** - Points to the region to be inverted.

See also GDIhiliteRgn

GDIisAlphaImage() (v3.3)

```
qbool GDIisAlphaImage( void* pPictData, qlong pPictDataLen )
```

Returns a boolean value indicating whether the supplied image data contains an alpha layer. GDIisAlphaImage() determines the picture format from the image data, which must be ptypColShared24 (defined in gdipict.he) in order to support alpha-blending.

- **pPictData** – The binary image data.
- **pPictDataLen** – The length of the image data in bytes.

GDIisRectEmpty()

qbool GDIisRectEmpty(qrect* pRect)

Returns true if the bottom of the rectangle is smaller than or equal to the top OR the right is smaller or equal to the left.

- **pRect** - Points to the rectangle to be tested.
- **return** - Returns qtrue if the rectangle is empty.

Example:

```
qrect theRect( 0, 0, -1, -1 );
if ( GDIisRectEmpty( &theRect ) )
{
  // we should get here
}
```

See also GDIequalRect, GDIptInRect

GDIlineTo()

void GDIlineTo(HDC pHdc, qdim pXpos, qdim pYpos) void GDIlineTo(HDC pHdc, qpoint* pPoint)

Draws a line from the current drawing cursor position to the specified position within the specified device/port. The drawing cursor is positioned at the new coordinate. GDIlineTo uses the current pen settings to draw the line.

- **pHdc** - Identifies the device/port
- **pXPos** - Horizontal position to which to draw the line
- **pYPos** - Vertical position to which to draw the line
- **pPoint** - Position to which to draw the line

Example:

See GDIfloodFill.

See also GDImoveTo

GDIlockHPIXMAP()

void* GDIlockHPIXMAP(HPIXMAP pPixMap)

Returns the base of a HPIXMAP image data, and locks the image data. Use this function to get at the HPIXMAP image bits.

- **pPixMap**- Identifies the HPIXMAP.

See Also GDIunlockHPIXMAP

GDImakeGrayScale()

```
HBITMAP GDImakeGrayScale( HPIXMAP pSource )
```

Converts the supplied HPIXMAP image to a gray-scale HBITMAP, returning the converted image.

- **pSource** – The HPIXMAP (alpha-blended) image to convert.

GDImakeHilited() (v4.0)

```
HBITMAP GDImakeHilited( HPIXMAP
pSource, qbyte pPercent, qcol pHiliteCol )
```

Merges the supplied HPIXMAP with the specified hilite color and converts to a HBITMAP, returning the converted image. Includes alpha support.

- **pSource** – The source image to be converted.
- **pPercent** – The percentage hilite required.
- **pHiliteCol** – The color to be used for the hilite.

GDImakeOptionClickProc()

```
FARPROC GDImakeOptionClick-
Proc(GDIoptionClickFunc
pGDIoptionClickFunc, HINSTANCE
pInstance)
```

Returns a FARPROC which then can be passed to GDIsetOptionClick.

- **pGDIoptionClickFunc** - The Option Click procedure.
- **pInstance** - Instance of the component.

See Also GDIsetOptionClick

GDImaskFromBitmap()

```
HBITMAPMASK GDImaskFromBitmap( HDC
pHdc, qdim pWidth, qdim pHeight , qcol
pTransColor)
```

Returns a mask for the bitmap which has been drawn at the coordinates 0,0 in the supplied device context. The mask is suitable for drawing the bitmap transparently using GDIdrawBitmap. It is the caller's responsibility to delete the mask when they have finished using it.

- **pHdc** - Identifies the device containing the bitmap for which the mask is to be created.
- **pWidth** - Is the width of the bitmap in pixels.
- **pHeight** - Is the height of the bitmap in pixels.

- **pTransColor-** The transparent color of the bitmap selected in pHdc.
- **return** - Returns the bitmap mask.

Example:

See GDIdragBitmapMove.

See also GDIcreateBitmap, GDIdrawBitmap

GDImoveTo()

```
void GDImoveTo( HDC pHdc, qdim pXPos,
qdim pYPos ) void GDImoveTo( HDC pHdc,
qpoint* pPoint )
```

Moves the drawing cursor to the specified position within the given device. It effects the starting position for GDIlineTo and GDIdrawText.

- **pHdc** - Identifies the device
- **pXPos** - New horizontal position
- **pYPos** - New vertical position
- **pPoint** - New position of drawing cursor

Example:

See GDIdrawText, GDIfloodFill.

See also GDIlineTo, GDIdrawText

GDIoffscreenPaintBegin() (v3.1)

```
void* GDIoffscreenPaintBegin( void*
pOffscreenPaintInfo, HDC& pHdc, qrect&
pRect, qrect& pUpdateRect)
```

Generally, off screen painting is used to avoid flicker on the screen, when painting complex or large objects. Flicker usually occurs when backgrounds are erased prior to painting the foreground of the object. However, on platforms like Mac OSX and Linux, this is not an issue, since all drawing occurs in a window buffer, and the screen is only updated when this buffer is flushed. Keeping the existing off screen painting code is unnecessary and wasteful on these platforms.

For this reason we have introduced two cross platform functions to do the right thing on each platform. You simply call GDIoffscreenPaintBegin, do your painting, and when finished, call GDIoffscreenPaintEnd.

- **pOffscreenPaintInfo** – The previous off screen paint info if calling recursively. Calling GDIoffscreenPaintBegin recursively is required when your object requires you to paint a row at the time for example.
- **pHdc** – Identifies the device into which you need to paint. On some platforms a off screen device will be returned in this parameter.
- **pRect** – The coordinates at which you need to draw. On some platforms this may be altered on return. You will need to paint at the new coordinates.
- **pUpdateRect** – The area to be updated. You usually pass a copy of the rcPaint rect of the WNDpaintStruct. On some platforms this may be altered on return. You will need to use the new coordinates if you wish to use this rect to determine which parts of your object require painting.

- **returns** – the off screen paint info which you need to pass to GDIoffscreenPaintEnd when you are finished, or to GDIoffscreenPaintBegin if calling recursively. If this function returns NULL, no painting is required. This only occurs if the pRect and pUpdateRect do not intersect.

Simple Example:

```
// this example shows how to draw off screen without recursion.
WNDpaintStruct ps;
WNDbeginPaint( theHwnd, &ps );
// get the client and update rects and the DC
qrect cRect,updateRect; HDC hdc;
WNDgetClientRect( theHwnd,&cRect );
updateRect = ps.rcPaint;
hdc = ps.hdc;
// prepare for offscreen paint
void* info = GDIoffscreenPaintBegin( NULL, hdc, cRect, updateRect );
if ( info )
{
  // now do your painting using hdc, cRect and updateRect
  // when done call GDIoffscreenPaintEnd
  GDIoffscreenPaintEnd( info );
}
WNDendPaint( theHwnd, &ps );
```

Complex Example:

```
// this example shows how to draw off screen with recursion,
// simulating what a list control would have to do to draw
// a row at a time off screen.
WNDpaintStruct ps;
WNDbeginPaint( theHwnd, &ps );
// calculate the rect of the first row and prepare for loop
qrect cRect, rowRect; WNDgetClientRect( theHwnd,&cRect );
rowRect = cRect; rowRect.bottom = rowRect.top + theRowHeight - 1;
void* info = 0;
HDC paintDC = ps.hdc;
// start the loop
while ( rowRect.top <= cRect.bottom )
{
  qrect paintRect = rowRect;
  qrect updateRect = ps.rcPaint;
  // store info in a temp local, in case NULL is returned from
  // GDIoffscreenPaintBegin. We must not loose the previous info.
  void* info2 = GDIoffscreenPaintBegin( info, paintDC, paintRect, updateRect );
  if ( info2 )
  {
    // if info2 is not NULL remember it in info
    info = info2;
    // now paint your row using paintDC, paintRect and updateRect
  }
  // prepare for the next row
  GDIoffsetRect( &rowRect, 0, theRowHeight );
}
// finish off
if ( info )
{
  GDIoffscreenPaintEnd( info );
}
WNDendPaint( theHwnd, &ps );
```

See also GDIoffscreenPaintEnd

GDIoffscreenPaintEnd() (v3.1)

```
void GDIoffscreenPaintEnd( void* pOffscreenPaintInfo )
```

Ends off screen painting which must have been initiated by calling GDIoffscreenPaintBegin, and updates the screen (on some platforms). For a full description see GDIoffscreenPaintBegin.

- **pOffscreenPaintInfo** – The off screen paint info returned by GDIoffscreenPaintBegin.

See also GDIoffscreenPaintBegin

GDIoffsetRect()

```
void GDIoffsetRect( qrect* pRect, qdim pXAmt, qdim pYAmt )
```

Offsets the specified rectangle by the specified amounts.

- **pRect** - Points to the rectangle to be offset.
- **pXAmt** - Specifies the amount to offset the rectangle horizontally.
- **pYAmt** - Specifies the amount to offset the rectangle vertically.

Example:

See GDIdragBitmapMove.

See also GDIinsetRect, GDIinflateRect, GDIsetRect

GDIoffsetRgn()

```
void GDIoffsetRgn( qrgn* pRgn, qdim pXAmt, qdim pYAmt )
```

Offsets the given region by the specified amounts.

- **pRgn** - Points to the region to be offset.
- **pXAmt** - Specifies the amount to offset the region horizontally.
- **pYAmt** - Specifies the amount to offset the region vertically.

See also GDIsetRectRgn, GDIoffsetRect

GDIpictGetBounds()

```
void GDIpictGetBounds( EXTfldval pPicture, qrect* pBounds )
```

Retrieves the bound's of the color shared picture.

- **pPicture** - The data of a color shared picture.
- **pBounds**- Points to qrect where the bounds will be stored.

GDIpixmapToColorShared()

qbool GDIpixmapToColorShared(HPIXMAP pSource, EXTfldval& pData)

Converts a HPIXMAP to an Omnis color shared picture format.

- **pSource** - The HPIXMAP to be converted.
- **pData** - Where the color shared picture data will be stored.
- **returns** - qtrue if successfully converted the HPIXMAP to color shared format.

See also GDIbitmapToColorShared

GDIptInRect()

qbool GDIptInRect(qrect* pRect, qpoint* pPoint)

Returns true if the specified point is located within the specified rectangle (inclusive of the rectangles border).

- **pRect** - Points to the qrect to test.
- **pPoint** - Points to the qpoint to test.
- **return** - Returns qtrue if the point is within the specified rectangle.

Example:

```
qrect theRect( 50, 50, 100, 100 );
qpoint thePoint( 75, 75 );
qbool ok = GDIptInRect( &theRect, &thePoint );
// ok should be qtrue
```

See also GDIptInRgn, GDIrectInRgn, GDIequalRect, GDIisRectEmpty

GDIptInRgn()

qbool GDIptInRgn(qrgn* pRgn, qdim pXPos, qdim pYPos) qbool GDIptInRgn(qrgn* pRgn, qpoint* pPoint)

Returns true if the specified point is located within the specified region.

- **pRgn** - Points to the region to be tested.
- **pXPos** - Specifies the horizontal position to test.
- **pYPos** - Specifies the vertical position to test.
- **return** - Returns qtrue if the point is located within the region.

OR

- **pRgn** - Points to the region to be tested.

- **pPoint** - Points to the qpoint to be tested.
- **return** - returns qtrue of point is located within the region.

Example:

```
qrgn theRegion1;
qrgn theRegion2;
GDIsetRectRgn( &theRegion1, 50, 50, 100, 100 );
GDIsetRectRgn( &theRegion2, 70, 70, 80, 80 );
// subtract region two from region one
GDIrgnDiff( &theRegion1, &theRegion1, &theRegion2 );
qbool ok1 = GDIptInRgn( &theRegion1, 75, 75 );
qbool ok2 = GDIptInRgn( &theRegion2, 75, 75 );
// ok1 should be qfalse, ok2 should be qtrue
```

See also GDIrectInRgn, GDIptInRect

GDIptInScreen()

```
qbool GDIptInScreen( qulong pFlags, qpoint* pPoint )
```

Tests whether or not the specified point is within the screen rect.

- **pFlags** - Specifies which areas to include or exclude. The following flags can be specified:
  GDI_SCR_SUB_TOP
  Subtracts height of top toolbar
  GDI_SCR_SUB_BOT
  Subtracts height of bottom toolbar and helpbar
  GDI_SCR_SUB_LEFT
  Subtracts width of left toolbar
  GDI_SCR_SUB_RIGHT
  Subtracts width of right toolbar
  GDI_SCR_SUB_ALL
  Subtracts all of the above
  GDI_SCR_GET_ALL
  Includes all connected monitors if specified (MacOS only).
- **pPoint** - Specifies the point to be tested.
- **return** - returns true if the point is within the screen rect.

See also GDIgetScreenRect

GDIrectInRgn()

```
qbool GDIrectInRgn( qrgn* pRgn, qrect* pRect )
```

Returns an indication of whether any pixel enclosed by a specified rectangle intersects with a specified region.

- **pRgn** - Points to the region to be tested.
- **pRect** - Points to the rectangle to be tested.
- **return** - Returns qtrue if the rectangle intersects the region.

Example:

```
qrgn theRegion1;
qrgn theRegion2;
qrect theRect( 32, 32, 78, 78 );
GDIsetRectRgn(theRegion1, 20, 20, 30, 30 );
GDIsetRectRgn(theRegion2, 80, 80, 100, 100 );
GDIrgnOr(theRegion1, theRegion1, theRegion2 );
qbool ok = GDIrectInRgn( &theRegion, &theRect );
// ok should be qfalse
```

See also GDIptInRgn, GDIptInRect

GDIreadVersion()

qshort GDIreadVersion(qchar* pVersion, qshort pMaxLen)

Returns the version string from the components resources. For web client components, the version number of the component must be implemented as a string in the string resources of the component. The web client plug-in reads this string for the purpose of the automated download mechanism. The format of the string resource must be as follows:

```
// RESOURCE_ID "VER MAJOR_VERSION MINOR_VERSION %%ORFC_VER%%"
//   = space
```

for example

```
31020 "VER 1 19 %%ORFC_VER%%"
// NOTE: the string resource ID must be 31020
```

GDIrgnAnd()

void GDIrgnAnd(qrgn* pRgnDest, qrgn* pRgnSrc1, qrgn* pRgnSrc2)

Calculates the intersection of two regions and stores the result in a third region.

- **pRgnDest** - Points to the region which is to receive the intersection of the two source regions. It is OK if this parameter is the same as either of the source regions.
- **pRgnSrc1** - Points to the first source region.
- **pRgnSrc2** - Points to the second source region.

Example:

```
qrgn rgn1, rgn2, rgn3;
GDIsetRectRgn( &rgn1, 10, 10, 50, 50 );
GDIsetRectRgn( &rgn1, 45, 45, 70, 70 );
GDIrgnAnd( &rgn3, &rgn1, &rgn2 );
// rgn3 should equal to a region of 45, 45, 50, 50
```

See also GDIrgnDiff, GDIrgnOr, GDIrgnXor

GDIrgnCopy()

```
void GDIrgnCopy( qrgn* pRgnDest, qrgn* pRgnSrc )
```

Takes a copy of source region and returns it in destination region.

- **pRgnDest** - Points to the destination region.
- **pRgnSrc** - Points to the source region.

See also GDIsetRectRgn, GDIcreateRectRgn

GDIrgnDiff()

```
void GDIrgnDiff( qrgn* pRgnDest, qrgn*
pRgnSrc1, qrgn* pRgnSrc2 )
```

Calculates the difference of two regions by subtracting source region two from source region one and stores the result in a third region.

Warning: The order of the two source regions is important. Subtracting region one from region two would yield entirely different results.

- **pRgnDest** - Points to the region which is to receive the difference of the source regions. It is OK if this parameter is the same as either of the source regions.
- **pRgnSrc1** - Points to the first source region.
- **pRgnSrc2** - Points to the second source region.

Example:

See GDIcreateRoundRectRgn, GDIptInRgn.

See also GDIrgnAnd, GDIrgnOr, GDIrgnXor

GDIrgnOr()

```
void GDIrgnOr( qrgn* pRgnDest, qrgn* pRgnSrc1, qrgn* pRgnSrc2 )
```

Calculates the union of two regions and stores the result in a third region.

- **pRgnDest** - Points to the region which is to receive the union of the source regions. It is OK if this parameter is the same as either of the source regions.
- **pRgnSrc1** - Points to the first source region.
- **pRgnSrc2** - Points to the second source region.

Example:

See GDIrectInRgn.

See also GDIrgnAnd, GDIrgnDiff, GDIrgnXor

GDIrgnXor()

void GDIrgnXor(qrgn* pRgnDest, qrgn* pRgnSrc1, qrgn* pRgnSrc2)

Calculates the union of two regions except for any portions that overlap, and stores the result in a third region.

- **pRgnDest** - Points to the region which is to receive the difference of the two regions. It is OK if this parameter is the same as either of the source regions.
- **pRgnSrc1** - Points to the first source region.
- **pRgnSrc2** - Points to the second source region.

See also GDIrgnAnd, GDIrgnDiff, GDIrgnOr

GDIselectBitmap()

HBITMAP GDIselectBitmap(HDC pHdc, HBITMAP pBitmap)

Replaces the device's bitmap with the specified bitmap, returning the HDC's previous bitmap. This can be used to switch a screen DC to a memory DC to draw off screen.

Example:

```
HBITMAP myBitmap = GDIcreateBitmap( 100, 100, 0 );
// creates an empty screen compatible bitmap
HDC theDC = GDIcreateScreenDC();
// creates a screen dc
HBITMAP screenBitmap = GDIselectBitmap( theDC, myBitmap );
// sets the dc bitmap to your bitmap returning the screen bitmap
// now use any of the GDI drawing functions to draw to your bitmap
GDIselectBitmap( theDC, screenBitmap );
// select the screen bitmap back into the DC so we can delete it
// without destroying our bitmap
GDIdeleteScreenDC( theDC );
// now the bitmap can be drawn to another DC
GDIdeleteBitmap( myBitmap );
// delete the bitmap if no longer required
```

- **pHdc** -Identifies the device into which to select the bitmap.
- **pBitmap** - Specifies the bitmap to select into the device.
- **return** - returns the previous bitmap.

Example:

See GDIdragBitmapMove, GDIcopyBits.

See also GDIcreateBitmap, GDIdeleteBitmap, GDIdrawBitmap

GDIselectObject()

object GDIselectObject(HDC pHdc, object)

Selects the given object into the specified DC, returning the previous object. It can take an HBRUSH, HFONT or HPEN object for its **object** parameter and will return the same type.

- **pHdc** - Identifies the DC into which the object will be selected.
- **object** - Specifies the object that is to be selected into the DC. This can be an HBRUSH, HFONT or HPEN object.
- **return** - Returns the object of the same type that was selected in the DC.

Example:

See GDIdraw3DPushButton, GDIfillPoly, GDIfontHeight, GDIframeEllipse, GDIinflateButtonRect.

See also GDIcreateBrush, GDIcreateFont, GDIcreatePen, GDIdeleteObject

GDIsetAlphaLevel() (v5.0)

HBITMAP GDIsetAlphaLevel(HBITMAP pSourceBMP, qbyte pAlphaLevel)

Processes an HBITMAP image, applying the specified alpha level, returning the converted image.

- **pSourceBMP** – The bitmap image to be converted.
- **pAlphaLevel** – The alpha color colonent required (0-255).

Example:

```
if (isAlpha) pDragDrop->mDragBitmap = GDIsetAlphaLevel(pDragDrop->mDragBitmap, 127);
```

GDIsetBkColor()

void GDIsetBkColor(HDC pHdc, qcol pColor)

Sets the current background color in the given device.

- **pHdc** - Identifies the device.
- **pColor** - Specifies the new background color.

Example:

See GDIdragBitmapMove, GDIcreatePoly, GDIfillPoly.

See also GDIgetBkColor, GDIgetTextColor, GDIsetTextColor

GDIsetBkColorAlpha() (v5.0)

qbyte GDIsetBkColorAlpha(HDC pHdc, qbyte pAlpha)

Sets the alpha component of the background color, returning the previous background alpha. Only applies when pHdc is created with GDIcreateAlphaScreenDC.

- **pHdc** - Identifies the device.

GDIsetClipRect()

```
void GDIsetClipRect( HDC pHdc, qrect* pRect )
```

Sets the clipping region of the given device to the specified rectangle, effectively disabling drawing outside the specified rectangle and enabling drawing inside the specified rectangle.

- **pHdc** - Identifies the device in which to set the clipping.
- **pRect** - Points to a qrect structure which specifies the new clipping region.

See also GDIsetClipRgn, GDIgetClipRect, GDIgetClipRgn

GDIsetClipRgn()

```
void GDIsetClipRgn( HDC pHdc, qrgn* pRgn )
```

Sets the clipping region of the given device to the specified region, effectively disabling drawing outside the specified region and enabling drawing inside the specified region.

- **pHdc** - Identifies the device in which to set the clipping.
- **pRgn** - Points to a qrgn structure which specifies the new clipping region.

See also GDIsetClipRect, GDIgetClipRect, GDIgetClipRgn

GDIsetColorEntries()

```
qshort GDIsetColorEntries( HPIXMAP
pPixMap, qshort pStart, qshort pCount,
qColorEntry* pEntries )
```

Set the color entries of a HPIXMAP. Returns the number of color entries which where actually stored.

- **pPixMap**- Identifies the HPIXMAP.
- **pStart**- The starting place to set color entries.
- **pCount**- The number of color entries to store in HPIXMAP.
- **pEntries**- Points to the buffer of color entries to stored.

GDIsetDrawingMode()

```
qdmd GDIsetDrawingMode( HDC pHdc, qdmd pMode )
```

Sets the current drawing mode in the specified device.

- **pHdc** - Identifies the drawing device.
- **pMode** - Specifies the new mode which can be one of the following:
  dmdCopy
  Overwrite all screen pixels with source pixels.
  dmdOr
  Overwrite where source pixel is set.
  dmdXor
  Invert where source pixel and screen pixel are both set.
- **return** - Returns the previous drawing mode.

GDIsetFlagsForAlphaDC() (v5.0)

qulong GDIsetFlagsForAlphaDC(HDC pHdc, qulong pFlags)

Sets flags for the alpha drawing device, returning the previous state of the flags. Alpha HDC flags are defined in GDI.HE.

- **pHdc** – Idenifies the device
- **pFlags** – Contains the new flag settings (logically ORed together)

Example:

```
mAdc = 0;
GDIcreateAlphaDC(&mAdc, pRect.width(), pRect.height());
GDIsetFlagsForAlphaDC(mAdc, GDI_ADC_BLENDMODE_BLEND | GDI_ADC_FLAG_WIN32_INSET_PEN);
GDIsetTextColorAlpha(mAdc, 255);
GDIsetBkColorAlpha(mAdc, 255);
mIsAlpha = qtrue;
```

GDIsetFlush() (v3.1)

void GDIsetFlush(qbool pEnable)

This function enables or disables all screen updates on Mac OSX and Linux. It is re-enterable, but each call to disable screen updates must be balanced with a call to enable screen updates.

- **pEnable** – If false, it increments the disabled count, otherwise it decrements it.

See also GDIflushDC

GDIsetFontName()

void GDIsetFontName(qfnt* pFnt, qchar *pFontName, qshort pLen)

Sets the qfnt's font name.

- **pFnt** - Points to the qfnt.
- **pFontName** - Points to the buffer which contains the name of the font. On Mac OSX, in addition to the standard system fonts, you can specify one of the following:
  "ThemeLabel"
  Standard Label font
  "ThemePushButton"
  Standard Push button font
  "ThemeApplication"
  Standard application font
  "ThemeSystem"
  Standard system font
  "ThemeEmphasizedSystem"
  Standard bold system font
  "ThemeSmallSystem"
  Standard small system font
  "ThemeSmallEmphasizedSystem"
  Standard small bold system font

“ThemeMenuTitle”
Standard menu title font
“ThemeMenuItem”
Standard menu item font
“ThemeMenuItemMark”
Standard menu item mark font
“ThemeMenuItemCmdKey”
Standard menu item command key font
“ThemeWindowTitle”
Standard window title font
“ThemeUtilityWindowTitle”
Standard utility window title font
“ThemeAlertHeader”
Standard alert header font
“ThemeViews”
Standard views font

- **pLen** - Specifies the length of the font name.

See also GDIgetFontName

GDIsetPalette()

```
qbool GDIsetPalette(HWND pHwnd, HPALETTE pPalette)
```

Sets the palette of a particular HWND.

- **pHwnd-** identifies the HWND.
- **pPalette** - palette to be used.

GDIsetPaletteEntries()

```
qshort GDIsetPaletteEntries( HPALETTE
pPalette, qshort pStart, qshort pCount,
qColorEntry* pEntries )
```

GDIsetPaletteEntries function sets an array of qColorEntries in a range of entries in the given HPALETTE.

- **pPalette** - Identifies the HPALETTE.
- **pStart** - The starting position.
- **pCount** - The number of color entries to set.
- **pEntries** - Points to array of qColorEntries.

GDIsetPixel()

```
void GDIsetPixel( HDC pHdc, qdim pX, qdim pY, qcol pCol)
```

Changes the color of the screen pixel below the specified location in the given device.

- **pHdc** - Identifies the device.

- **pX** - Specifies the horizontal coordinate of the screen pixel.
- **pY** - Specifies the vertical coordinate of the screen pixel.
- **pCol** - Specifies the new color for the screen pixel.

See also GDIgetPixel

GDIsetPortClipRegion() (v3.1, Mac OSX only)

```
void GDIsetPortClipRegion( HDC pHdc, RgnHandle pTheRegion )
```

Sets the viewport clipping region.

- **pHdc** - Identifies the device.
- **pTheRegion** – Identifies the rectangular region to be clipped.

GDIsetRect()

```
void GDIsetRect( qrect* pRect, qdim pLeft,
qdim pTop, qdim pRight, qdim pBottom )
```

Sets the given rectangle from the specified coordinates.

- **pRect** - Points to the qrect to be set.
- **pLeft** - Specifies the new left coordinate.
- **pTop** - Specifies the new top coordinate.
- **pRight** - Specifies the new right coordinate.
- **pBottom** - Specifies the new bottom coordinate.

See also GDIcopyRect, GDIsetRectEmpty

GDIsetRectEmpty()

```
void GDIsetRectEmpty( qrect* pRect )
```

Sets all coordinates of the given rect to zero.

- **pRect** - Points to the qrect.

See also GDIsetRect, GDIisRectEmpty

GDIsetRectRgn()

```
void GDIsetRectRgn( qrgn* pRgn, qdim pLeft, qdim pTop, qdim pRight, qdim pBottom ) void GDIsetRectRgn( qrgn* pRgn, qrect* pRect )
```

Sets an existing region to a rectangular area from the specified coordinates or rectangle.

- **pRgn** - Points to the region which will be altered.
- **pLeft** - The new left edge of the region.
- **pTop** - The new top edge of the region.
- **pRight** - The new right edge of the region.
- **pBottom** - The new bottom edge of the region.

OR

- **pRgn** - Points to the region which will be altered.
- **pRect** - The rectangle specifying the rectangular region.

Example:

See GDIcreateRoundRectRgn, GDIptInRgn, GDIrectInRgn, GDIrgnAnd.

See also GDIcreateRectRgn, GDIgetRgnBox

GDIsetTextColor()

```
void GDIsetTextColor( HDC pHdc, qcol pColor )
```

Sets the current text color in the specified device.

- **pHdc** - Identifies the device.
- **pColor** - Specifies the new text color.

Example:

See GDIdragBitmapMove, GDIcreatePoly, GDIdraw3DPushButton, GDIfillPoly.

See also GDIgetBkColor, GDIgetTextColor, GDIsetBkColor

GDIsetTextColorAlpha() (v5.0)

```
qbyte GDIsetTextColorAlpha( HDC pHdc, qbyte pAlpha )
```

Sets the alpha component of the text color, returning the previous text alpha.

- **pHdc** - Identifies the device.

GDIsetViewportOrg()

void GDIsetViewportOrg (HDC pHdc, qdim pX, qdim pY)

Redefines the local coordinates for the given device. Altering the view port origin alters the location of all subsequent drawing to that device, for example, specifying -10 for the pX coordinate would add 10 to all horizontal coordinates for all subsequent drawing.

- **pHdc** - Identifies the device.
- **pX** - Specifies the new horizontal origin.
- **pY** - Specifies the new vertical origin.

See also GDIgetViewportOrg

GDIstartText() (v5.0)

void GDIstartText(HDC pHdc,
GDItextSpecStruct* pTextSpec, HDC
pFontHdc = 0)

Begins a text drawing operation. Must be accompanied by GDIendText() in order to release the device for other GDI operations.

- **pHdc** – Identifies the drawing device.
- **pTextStruct** – Structure containing font, style and justification information.
- **pFontHDC** – If specified, the HFONT will use the font family information associated with the font and the given DC.

Example:

```
//calculate width of popup window (max width of text in list)
qdim tqfPopupWindow::calcWidth(GDItextSpecStruct &pTextSpec, qlist &pList)
{
  qlong lineCount = pList.linecnt();
  if (lineCount)
  {
    HDC hdc = GDIgetTempDC();
    GDIstartText(hdc, &pTextSpec);
    qdim maxWidth = 0;
    str255 line; qlong lineNumber;
    for (lineNumber = 1; lineNumber <= lineCount; ++lineNumber)
    {
      pList.getline(lineNumber, &line);
      line.convcset(csetApi);
      stripblank(line, qfalse, qtrue);
      qdim width = GDItextWidth(hdc, &line[1], line.length());
      if (width > maxWidth) maxWidth = width;
    }
    GDIendText(hdc,&pTextSpec);
    // Allow for scroll bar, margins and borders
    return maxWidth + scbarwid + 6;
  }
  return 0;
}
```

GDItextBox()

```
qshort GDItextBox( HDC pHdc, qrect *pRect,
qchar *pText, qshort *pTextLen, qshort
pBufLen, qjst pJst, qbool pDraw = qtrue )
```

Adjusts the supplied text if necessary, so that it fits in the supplied rectangle. If the text is too long to draw in the rectangle, it replaces the minimum number of characters at the end of the text with an ellipsis (...), so the text fits in the rectangle. It then optionally draws the text string.

- **pHdc** - Identifies the device; the font must already be selected.
- **pRect** - Identifies the rectangle in which the text must fit; updated on return, ready for drawing the text (possibly modified to include an ellipsis) at coordinates left, top, and using justification jstLeft.
- **pText -** The text. This may be updated, so that the ellipsis replaces some of the text.
- **pTextLen -** The length of the text; updated on return with the new number of characters in the buffer, including the ellipsis if one has been added.
- **pBufLen -** The length of the buffer starting at address pText. This allows GDI to check how much space is available for appending the ellipsis.
- **pJst** - The horizontal justification. See description of **qjst** for more detail.
- **pDraw** - If qfalse, GDI does everything except draw the text. If qtrue, GDI also draws the text, in which case the text color must be set in the DC.
- **return** - Returns the new length of the text after it has been made to fit without counting the ellipsis.

Example:

See GDIdraw3DPushButton.

See also GDIdrawText, GDIdrawTextJst

GDItextWidth()

```
qdim GDItextWidth( qchar* pText, qshort
pTextLen ) qdim GDItextWidth( qchar* pText,
qshort pTextLen, GDItextSpecStruct*
pTextSpec ) qdim GDItextWidth( HDC pHdc,
qchar* pText, qshort pTextLen )
```

Returns the width of the text in screen units in the specified font and style (mFnt and mSty of pTextSpec). The text must not contain escape characters as used by GDIdrawTextJst.

If only the text and text length are specified, the current HFONT in the temp DC is used.

If pTextSpec is specified, the function bases the text width on an HFONT based on the text spec.

If pHdc is specified, the width is based on the current HFONT in the specified DC.

- **pText** - The text from which to calculate the screen units.
- **pTextLen** - The length of the text.
- **pTextSpec** - Specifies the font and style.
- **pHdc** - Identifies the device which contains the HFONT.
- **return** - Returns the width of the text in screen units.

Example:

See GDIinflateButtonRect.

See also GDIfontHeight, GDIfontPart

GDItextWidthJst()

qdim GDItextWidthJst(GDIdrawTextStruct* pTextStruct)

Calculates the text width of more complex text strings. For a full description see GDIdrawTextJst.

- **pTextStruct** – GDIdrawTextStruct structure with information about the text string. See GDIdrawTextJst for a full description.

return - Returns the width of the complex string in pixels.

See also GDIdrawTextJst

GDIthemeText() (v3.1, Mac OSX only)

qlong GDIthemeText(HDC pHdc, qchar* pText, qshort pTextLen, eThemeTextMode pMode)

GDIthemeText measures or paints OSX theme text. The theme font must have been already selected into the DC prior to calling this function. It is usually not required to call this function. Once a theme font has been selected into a DC, the standard GDI text drawing/measure functions will call this function to render or measure the text.

Theme fonts can be created by setting the font name of a qfnt to a theme font name (see GDIsetFontName).

- **pHdc** - Identifies the device with the selected font.
- **pText** – pointer to the text.
- **pTextLen** – length in characters of the text.
- **pMode** – the theme mode. This can be one of the following modes:
  eThemeTextWidth
  tells GDIthemeText to measure and return the width of the text
  eThemeTextDraw
  tells GDIthemeText to render the given text
  the following are modifiers of which one must be passed in addition to one of the modes above:
  eThemeTextActive
  text is active (enabled)
  eThemeTextInactive
  text is inactive (disabled)
  eThemeTextPressed
  text is pressed (hilited)

Example:

```
// create the font
qfnt theFnt( eThemePushButtonFont );
HFONT hfont = GDIcreateFont( &theFnt, styPlain );
// select the font and draw the text
hfont = GDIselectObject( theDC, hfont );
GDIthemeText( theDC, theTextPtr, theTextLen, eThemeTextDraw | eThemeTextPressed );
hfont = GDIselectObject( theDC, hfont );
// destroy the font
GDIdeleteObject( hfont );
// the following example achieves the same result. Note the difference when calling
// GDIcreateFont
qfnt theFnt( eThemePushButtonFont );
```

```
// or we could also use qfnt theFnt = fntButt;
HFONT hfont = GDIcreateFont( &theFnt, styPlain, eThemeTextPressed );
hfont = GDIselectObject( theDC, hfont );
GDIdrawText( theDC, theTextPtr, theTextLen );
hfont = GDIselectObject( theDC, hfont );
GDIdeleteObject( hfont );
```

See also GDIsetFontName, GDIcreateFont, qfnt::qfnt

GDIthemeText() (v3.1, Mac OSX only)

qlong GDIthemeText(GDItextSpecStruct* pTextSpec, qchar* pText, qshort pTextLen, eThemeTextMode pMode)

This version is almost identical to GDIthemeText above. The only difference is, you do not need to create and select a HFONT into a DC, but must make sure to set the current DC (see GDIcheckPort) if you whish to render the text.

- **pTextSpec** – structure containing font information.
- **pText** – pointer to the text.
- **pTextLen** – length in characters of the text.
- **pMode** – the theme mode. This can be one of the following modes:
  eThemeTextWidth
  tells GDIthemeText to measure and return the width of the text
  eThemeTextDraw
  tells GDIthemeText to render the given text
  the following are modifiers of which one must be passed in addition to one of the modes above:
  eThemeTextActive
  text is active (enabled)
  eThemeTextInactive
  text is inactive (disabled)
  eThemeTextPressed
  text is pressed (hilited)

See also GDIsetFontName, GDIcreateFont, qfnt::qfnt

GDIunionClipRect()

void GDIunionClipRect(HDC pHdc, qrect* pRect)

Sets the clipping region of the given device to the union of the clipping region and the specified rectangle, effectively enabling drawing inside the specified rectangle and the existing clipping region.

- **pHdc** - Identifies the device in which to set the clipping.
- **pRect** - Points to a qrect structure which specifies the rectangle to be added to the clipping region.

See also GDIexcludeClipRect, GDIexcludeClipRgn, GDIunionClipRgn

GDIunionClipRgn()

```
void GDIunionClipRgn( HDC pHdc, qrgn* pRgn )
```

Sets the clipping region of the given device to the union of the clipping region and the specified region, effectively enabling drawing inside the specified region and the existing clipping region.

- **pHdc** - Identifies the device in which to set the clipping.
- **pRgn** - Points to a qrgn structure which specifies the region to be added to the clipping region.

See also GDIexcludeClipRect, GDIexcludeClipRgn, GDIunionClipRect

GDIunionRect()

```
qbool GDIunionRect( qrect* pDestRect,
qrect* pSrcRect1, qrect* pSrcRect2 )
```

Returns the union of two rectangles in a third rectangle, and returns true if the resulting rectangle is not empty.

- **pDestRect** - Points to the destination rectangle for the union.
- **pSrcRect1** - Points to source rectangle one.
- **pSrcRect2** - Points to source rectangle two.

return - Returns qtrue if the resulting rectangle is not empty.

See also GDIintersectRect

GDIunlockHPIXMAP()

```
void GDIunlockHPIXMAP( HPIXMAP pPixMap )
```

This function unlocks the HPIXMAP bits. Use this function when you have finished setting the bits of a HPIXMAP.

See Also GDIlockHPIXMAP

GDIuseStyledTextColors() (V2.2)

```
void GDIuseStyledTextColors( qbool pUseColors )
```

This method allows you to turn of color styles in styled text when calling GDIdrawTextJst. This is useful when drawing highlighted text or highlighted list rows. It looks strange when drawing highlighted text and words within the text display in various colors.

IMPORTANT: This affects a global setting which must be restored to true when drawing is complete.

pUseColors - specifies whether to turn styled text colors off or on .

Example:

```
GDIuseStyledTextColors( qfalse );
// draw your text
GDIuseStyledTextColors( qtrue );
```

See Also GDIdrawTextJst

Chapter 13—PRI Reference

This chapter describes the public interface of the PRI module, which is the Omnis cross-platform print manager. This chapter includes a description of the Structures, Data types, and Defines required by some PRI functions, the Messages sent to the printing and custom device message procedures, and PRI Functions.

The Omnis print manager can be used by all external components, visual or non-visual.

The print manager consists of three parts:

- **The input manager.**

The input manager allows the creation of a print job and supports the printing of various types of objects. Objects given to the input manager are formatted according to their properties and the properties of the print job.

- **The output manager.**

The output manager is responsible for managing the various output devices and sending the formatted objects to the chosen output device.

- **Internal output devices.**

The print manager implements a number of output devices. These are standard devices which send their output to Printers, Screen Previews, Ports, Text files, etc.

External components can utilize all three parts of the print manager. It is possible to write external components which use the print manager to print data of visual or non visual components and to write output components which convert the formatted objects to the relevant output format, and which can be derived from any of the internal devices. There is no restriction to what an individual external component library can do. One single library could implement both external controls or methods which can print data, and one or more external output devices which can receive the formatted data.

The Input Manager

The input manager takes care of the following tasks:

- Page/Job formatting. There are various functions to manage page and job formatting information. These are functions to open page setup and job setup dialogs, and to manipulate and manage the resulting information.
- Page generation. There are various functions to control and manipulate the generation of pages. The print manager will take care of page generation if enabled, and the generation of page headers and footers.
- Page buffering and ejection. All objects are buffered until a whole page is completed. Once complete, the page is ejected (if auto ejection is enabled) and destroyed if the destination device no longer requires it (i.e. Printer). If the destination device is a device which requires the page to be kept (i.e. Page preview), the page will be buffered on disk and read into memory when required. There are various functions to control page ejection.
- Object formatting. There are various functions to add to and manage objects of a print job. The various object types supported by the print manager have properties which give some individual control over their formatting and appearance. The print manager works in resolutions of 1/1000th of a millimeter (1 qpridim). There are various conversion functions to convert to and from centimeters, inches, and device coordinates.

In order to print, two things have to be implemented:

- **The message class**

The message procedure will receive various messages during the printing process. It will be responsible for dealing with page header and footer objects and various other tasks.

- **The printing method**

The printing loop will be responsible for starting the print job, printing the data, and closing the print job. This method can be made a member of the message class which is recommended.

A simple print job

When we want to print some data, we need to divide our data into three parts.

- Data for each page header.
- Data for each page footer.
- Data for the main body of the document.

The print manager operates in a number of coordinate spaces. We will only use three of them. Page header data is added to the page header coordinate space (ePosHeader), Page footer data is added to the page footer coordinate space (ePosFooter), and the main data is added to the main body coordinate space (ePosGlobal).

The following example code prints data directly from an Omnis list. The list data is printed to ePosGlobal, the column names of the list are printed to ePosHeader, and the page number is printed to ePosFooter.

Declaration

```
class myPrintClass: public PRIprocClass
{
  private:
    EXTCompInfo* mEci;
    EXTqlist*    mList;
    qlong  mPageNumber;
    qfnt   mFnt;
  public:
    // constructor and destructor
    myPrintClass( EXTCompInfo* pEci, EXTqlist* pList, qfnt* pFnt );
    ~ myPrintClass();
    // main printing function
    qprierr print();
    // message function
    virtual qprierr PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam );
};
```

Implementation

```
// ***** constructor *****
myPrintClass::myPrintClass( EXTCompInfo* pEci, EXTqlist* pList, qfnt* pFnt )
{
  mEci = pEci;
  mList = pList;
  mPageNumber = 0;
  mFnt = *pFnt;
}
// ***** destructor *****
myPrintClass::~myPrintClass()
{
}
// ***** Main print loop *****
qprierr myPrintClass::print()
{
  // open the print manager. This must always be done.
  qprierr err = PRIopen();
  if ( err != PRI_ERR_NONE ) return err;
  // start the print job
  // we will get the destination parameters from Omnis. The user will
  // have set the current destination and its related parameters.
```

```
// we will also turn on horizontal pages, so all columns will be printed.
PRIparmStruct jobParms(qnil);
jobParms.mProc = this;
jobParms.mGeneratePages = qtrue;
jobParms.mAutoEject = qtrue;
jobParms.mDestParms = ECOgetDeviceParms( mEci->mInstLocp );
jobParms.mApp = ECOgetApp( mEci->mInstLocp );
jobParms.mHorzPages = qtrue;
err = PRIstartJob( & jobParms );
// Print the list data
// this will generate new pages, page header and footer messages as needed
// we can reuse the same PRIobjectStruct for all list cells. We only need to change
// the data and position of the object every time round.
// Note: when we construct the PRIobjectStruct with qnil, all members are initialized
// to zero we only need to set the members which we do NOT want to be zero.
// Note: we only print text, number and date list data
qpridim  lastBotom = 0;
EXTfldval  fvalp;
fftType    fft;
PRIobjectStruct obj(qnil);
obj.mType = PRI_OBJ_TEXT;
obj.mFnt = mFnt;
obj.mVertExtend = qtrue;
obj.mMultiLine = qtrue;
obj.mPos.mMode = ePosGlobal;
obj.mData = fvalp.getFldVal();
// Outer loop based on list rows
for ( qlong row = 1 ; row <= pList->rowCnt() ; row++ )
{
  obj.mPos.top = obj.mPos.bottom = lastBottom;
  // inner loop based on list columns
  for ( qshort col = 1 ; col <= pList->colCnt() ; col++ )
  {
    // get the data from the list but no need to take a copy. The print manager will
    // copy the data.
    pList->getColValRef( row, col, fvalp, qfalse );
    // test the data type to see if we can print it
    fvalp.getType( fft )
    switch ( fft )
    {
      case fftCharacter: case fftBoolean: case fftDate:
      case fftNumber: case fftInteger: case fftConstant:
      {
        break;
      }
      default:
      {
        continue;
      }
    }
    // set the horizontal position based on column count
    obj.mPos.left = (PRI_INCH*2+PRI_1_4INCH)*(col-1);
    obj.mPos.width( PRI_INCH*2 );
    // now add the object and check for errors
    err = PRIaddObject( jobParms.mJob, &obj );
    if ( err != PRI_ERR_NONE ) break;
    // on returning from PRIaddObject, obj.mPos.bottom will have been altered to
    // fit all of the text. We will remember the greatest value in lastBottom so we
    // can use it as our new top when printing the next row.
    if ( obj.mPos.bottom > lastBottom ) lastBottom = obj.mPos.bottom;
```

```
    }
    // we want to leave a two point gap between rows.
    lastBottom += PRI_POINT * 2;
    if ( err != PRI_ERR_NONE ) break;
  }
  // close the print job
  err = PRIendJob(jobParms.mJob);
  // close the print manager
  if ( err )
    // if we had a job error we wish to return the job error from our function, and not
    // the error returned by PRIclose (which after all may be successful). So we ignore
    // the result.
    PRIclose();
  else
    err = PRIclose();
  return err;
}
// end myPrintClass::print
// *** the message proc ***
qprierr myPrintClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_INIT_PAGE:
    {
      // initialize the local, header and footer boundaries
      // we only need to do it for the first page. All subsequent pages will
      // inherit the settings of the previous page
      PRIpageStruct* pgInfo = (PRIpageStruct*)lParam;
      if ( pgInfo->mPage.mVert == 1 )
      {
        // we will set ½ inch margins all around, inset from
        // the paper edge
        pgInfo->mLocalBounds = pgInfo->mPaperBounds;
        pgInfo->mLocalBounds.inset(PRI_1_2INCH,PRI_1_2INCH);
        // within that we will fit our page header and footer
        // and make them ½ inch tall
        pgInfo->mHeaderBounds = pgInfo->mLocalBounds;
        pgInfo->mHeaderBounds.height(PRI_1_2INCH);
        pgInfo->mFooterBounds = pgInfo->mLocalBounds;
        pgInfo->mFooterBounds.top = pgInfo->mFooterBounds.bottom - PRI_1_2INCH + 1;
        // reduce the local bounds vertically so it doesn't overlap the
        // header and footer bounds
        pgInfo->mLocalBounds.inset( 0, PRI_1_2INCH );
      }
      return PRI_ERR_NONE;
    }
    case PM_ADD_HEADER_OBJECTS:
    {
      // add objects to page header
      // here we add our list column names
      // Note: we will only receive one message for horizontal page 1,
      // although horizontal pages are enabled. If we wanted to receive a
      // message for every horizontal page we would also have to enable
      // horizontal headers ( PRIparmStruct.mHorzHeaders )
      PRIpageStruct* pgInfo = (PRIpageStruct*)lParam;
      EXTfldval   fval;
      str255      colName;
      PRIobjectStruct obj(qnil);
      obj.mType = PRI_OBJ_TEXT;
```

```
    obj.mFnt = mFnt;
    obj.mTextStyle = styBold;
    obj.mData = fval.getFldVal();
    // initialize the position to a page header position
    obj.mPos = qpripos( ePosHeader, pgInfo->mPage.mHorz, pgInfo->mPage.mVert, 0, 0, PRI_INCH*2, PRI_1_2INCH
    // add the column names
    for ( qshort col = 1 ; col <= pList->colCnt() ; col++ )
    {
      // set the horizontal position based on column count
      obj.mPos.left = (PRI_INCH*2+PRI_1_4INCH)*(col-1);
      obj.mPos.width( PRI_INCH*2 );
      // get the column name
      mList->getCol( col, qfalse, colName );
      fval.setChar( colName );
      // add the object and check for errors
      qprierr err = PRIaddObject( pJob, &obj );
      if ( err != PRI_ERR_NONE ) return err;
    }
    return PRI_ERR_NONE;
  }
  case PM_ADD_FOOTER_OBJECTS:
  {
    // add objects to page footer
    // we just want to display the page number centrally to the footer
    PRIpageStruct* pgInfo = (PRIpageStruct*)lParam;
    EXTfldval  fval;
    PRIobjectStruct obj(qnil);
    obj.mType = PRI_OBJ_TEXT;
    obj.mFnt = mFnt;
    obj.mTextJust = jstCenter;
    obj.mData = fval.getFldVal();
    // initialize the position to a page footer position
    obj.mPos = qpripos( ePosHeader, pgInfo->mPage.mHorz, pgInfo->mPage.mVert, 0, 0, pgInfo->mFooterBounds.wi
    // convert the page number to text
    str255 text = str255("Page $");
    str15 num; qlongToString( page->mPage.mVert, num );
    text.insertStr( num );
    fval.setChar( text );
    // add the object and return error
    return PRIaddObject( pJob, &obj );
  }
  case PM_CLOSE:
  {
    // the last output device has been closed.
    // we are no longer required. So delete our selves
    delete this;
    return PRI_ERR_NONE;
  }
  case PM_OUT_PRINTER:
  case PM_OUT_PAGE:
  case PM_OUT_PREVIEW:
  case PM_OUT_DISK:
  {
    // these are messages generated by clicks on the buttons
    // of the screen report or page preview toolbar.
    // we simply call the default output procedure.
    // if we don't, nothing will happen
    return PRIdefOutputProc( pJob, NULL, pMessage, lParam, 0, 0 );
  }
}
```

```
  return PRI_ERR_NONE;
}
// end myPrintClass::PriProc
```

The Output Manager

The output manager takes care of the following tasks:

- Registration of output devices. When Omnis starts up the startup function of the print manager registers all internally implemented output devices with the output manager. When the external component manager initializes it sends connect messages to all external components. Any external components that implement output devices must register them with the print manager at this stage. The output manager will always instanciate one instance of all output devices on registration.
- Instanciating and destroying output devices when required. When the first page of a print job is ejected the input manager sends the page to the output manager. It is then that the output device for the job is instanciated. Which device depends on the destination parameters that were passed to the print job on creation. It is possible that more than one output is instanciated for any one job, i.e. the screen report output invokes the page preview output when the user clicks on the page preview button, or the printer output when the print button is clicked.
- Sending objects of an ejected page to the output device. It is the responsibility of the output device to tell the output manager to send the objects of a page. Some devices will request the objects when the page just has been ejected (the output devices receives a PM_OUT_ADDPAGE message), others when their HWND receives a paint message.

A simple external output device

An external output device in its simplest form has to do the following.

- Decide from which internal output device to derive from. This depends very much on the required functionality to be inherited. A device that draws to the screen should most probably be derived from the PRI_DEST_EXTHDC device (an internal output device with capabilities of drawing objects to a dc). If designing a device which primarily generates text, the device should probably derive from PRI_DEST_EXTTEXT (an internal output device with capabilities of preparing formatted pages of text). For a full description see Internal output devices.
- Implement a message function to receive the various messages generated by the output manager.
- Create an output class which can be instanciated and destructed when required.
- Implement ECM_CONNECT and ECM_DISCONNECT messages in the GenericWndProc of your external component.

The following example code will generate a simple text based HTML file which can be viewed from any browser. All objects other than text are ignored. The sample device will derive from PRI_DEST_EXTFILE and is only a very basic example. For a more complete example of the HTML device refer to the HTML component source which ships with Omnis Studio.

```
// ***** the output class *****
class htmlOutputClass
{
  public:
  htmlOutputClass( htmlOutputClass* pOutput );
  ~ htmlOutputClass();
  Qprierr openDevice( PRIdestParmStruct* pDestParms );
  qprierr closeDevice();
  qprierr writeDevice( qbyte* pData, qlong pDataLen, qbool pLineFeed );
  qprierr writeDevice( qfldval pData );
  qprierr sendText( qchar* pText, qlong pTextLen, qbool pLineFeed, qbool pFormFeed );
  qprierr sendData( qbyte* pData, qlong pDataLen );
  // inlines
  qbool isDeviceOpen() { return smTheDevice != NULL; }
  qprierr setJob( PRIjob pJob ) { smJob = pJob; return PRI_ERR_NONE; }
  qprierr clearJob() { smJob = NULL; return PRI_ERR_NONE; }
```

```
  private:
  static FARPROC smPriDeviceProc;
  static qlong smPriDeviceID;
  static qfileptr smTheDevice;
  static PRIjob smJob;
  qbool mDeviceIsMine;
};
// ***** the message function *****
qprierr DeviceFunc( PRIjob pJob, void* pOutput, qulong pData, UINT pMessage, LPARAM lParam1, LPARAM lParam2, L
{
  // typecast our data send to as in pData
  // WARNING: this may be NULL the first time PM_OUT_CONSTRUCT is send
  htmlOutputClass* output = (htmlOutputClass*)pData;
  switch ( pMessage )
  {
    case PM_OUT_CONSTRUCT:
    {
      // construct an instance of our output class. If this is the first call,
      // output will be NULL, otherwise we should inherit properties
      // from the output send to us.
      // lParam1 points to a long storage in which we must
      // return our new instance pointer.
      qulong* retLong = (qulong*)lParam1;
      *retLong = (qulong) new htmlOutputClass( output );
      return *retLong ? PRI_ERR_NONE : PRI_ERR_MEMORY;
    }
    case PM_OUT_DESTRUCT:
    {
      // destroy an instance of our output class.
      // lParam1 points to a long storage which contains the pointer to the
      // output class to be destroyed
      htmlOutputClass* delOutput =
htmlOutputClass*)(*(qulong*)lParam1;
      if ( delOutput ) delete delOutput;
      return PRI_ERR_NONE;
    }
    case PM_OUT_GETDLGIDS:
    {
      // return dialog modes for the parameter and page size panes of the
      // destination dialog. We want to inherit the dialog panes of both
      // panes so we return PRI_DLG_INHERIT for both.
      rstrno* parmPane = (rstrno*)lParam1;
      rstrno* sizePane = (rstrno*)lParam2;
      *parmPane = PRI_DLG_INHERIT;
      *sizePane = PRI_DLG_INHERIT;
      return PRI_ERR_NONE;
      // if we wanted to implement our own panes and parameters we would
      // have to implement the messages PM_OUT_GETPARMDLG,
      // PM_OUT_GETPAGEDLG, PM_OUT_LOADPARMS,
      // PM_OUT_SAVEPARMS, PM_OUT_VALIDATEPARMS,
      // PM_OUT_GETPARM, PM_OUT_SETPARM, PM_OUT_AFTER,
      // and PM_OUT_CLICK.
    }
    case PM_OUT_OPEN:
    {
      // an Omnis print job has opened the device. We don't actually open
      // the device here, but remember that a print job is using the device.
      // this is important so we can prevent calls to PM_OUT_SENDATA
      // and PM_OUT_SEND_TEXT. Some devices may choose to allow
      // raw text and data to be send while a print job is going. Ours doesn't.
```

```
  return output->setJob( pJob );
}
case PM_OUT_CLOSE:
{
  // the Omnis job is closing. We simply clear the job.
  return output->clearJob();
}
case PM_OUT_OPENDEVICE:
{
  // we have been requested to open our device
  // lParam1 contains a pointer to the destination parameters
  return output->openDevice( (PRIdestParmStruct*)lParam1 );
}
case PM_OUT_CLOSEDEVICE:
{
  // we have been requested to close our device
  return output->closeDevice();
}
case PM_OUT_ISDEVICEOPEN:
{
  // return true if the device is open. lParam1 points to a qbool
  qbool* isOpen = (qbool*)lParam1;
  isOpen = output->isDeviceOpen();
  return PRI_ERR_NONE;
}
case PM_OUT_GETEOL:
{
  // return end of line characters so super device can format the page data
  // lParam1 = strxxx*
  strxxx* eol = (strxxx*)lParam1;
  *eol = str15("<BR>");
  return PRI_ERR_NONE;
}
case PM_OUT_SENDTEXT:
{
  // request to send text directly to device
  // lParam1 = pointer to text
  // lParam2 = text length
  // lParam3 = HIWORD = line feed, LOWORD = form feed
  return output->sendText( (qchar*)lParam1, (qlong)lParam2, (qbool)HIWORD(lParam3), (qbool)LOWORD(lParam3)
}
case PM_OUT_SENDDATA:
{
  // request to send data directly to device
  // lParam1 = pointer to data
  // lParam2 = data length
  return output->sendData( (qbyte*)lParam1, (qlong)lParam2 );
}
case PM_OUT_SENDPAGE:
{
  // our super device has formatted a complete page which we can
  // write to our device
  // lParam1 = PRIpageStruct*
  // lParam2 = qfldval containing the page data
  // we will write the data directly to the device
  return output->writeDevice( (qfldval)lParam2 );
}
case PM_OUT_JOBSETUP: // v3.0 only
{
  // send to custom devices overloading default printer device when
```

```
      // PRIopenJobSetupDialog is called
      // You may call PRIopenJobSetupDialog from here
      // lParam1 = PRIjob
      // lParam2 = PRIpageSetup*
      // lParam3 = qbool*
      return PRIopenJobSetupDialog( (PRIjob)lParam1, (qbool*)lParam2 );
    }
  }
  return PRIdefOutputProc( pJob, pOutput, pMessage, lParam1, lParam2, lParam3 );
}
// ***** the GenericWndProc *****
extern "C" qlong OMNISWNDPROC GenericWndProc(HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo*
{
  // Initialize callback tables - THIS MUST BE DONE
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    case ECM_CONNECT:
    {
      // register the device
      PRIdeviceInfoStruct info(qnil);
      info.mBitmap = RESloadBitMap( gInstLib, DEV_BASE_ID );
      info.mExtInfo = eci->mPrivate;
      // set our properties
      // we can be opened directly ($open() and $close())
      info.mCanOpenDirect = qtrue;
      // we can receive text and data directly ($sendtext(), $senddata())
      info.mCanSendText = info.mCanSendData = qtrue;
      // we can appear in the destination dialog
      info.mShowInDialog = qtrue;
      // set the device name
      RESloadString( gInstLib, DEV_NAME_ID, info.mName );
      // create message function pointer and remember it
      htmlOutputClass::smPriDeviceProc = PRImakeCustomProc( DeviceFunc, gInstLib );
      // register
      qprierr err = PRIregisterOutput( &info, PRI_DEST_EXTFILE, htmlOutputClass::smPriDeviceProc );
      // after registration we will have been given a unique id, remember it
      htmlOutputClass::smPriDeviceID = info.mID;
      // do default action for connect message
      qlong ret = WNDdefWindowProc( hwnd, Msg, wParam, lParam, eci );
      // tell the component that we must remain loaded, that we can always be used,
      // and that we are a device output component and return
      ret |= EXT_FLAG_REMAINLOADED | EXT_FLAG_ALWAYS_USABLE;
      ret |= EXT_FLAG_PRI_OUTPUT;
      return ret;
    }
    case ECM_DISCONNECT:
    {
      PRIunregisterOutput( htmlOutputClass::smPriDeviceID );
      PRIdisposeCustomProc( htmlOutputClass::smPriDeviceProc );
      return WNDdefWindowProc( hwnd, Msg, wParam, lParam, eci );
    }
    // implement any other messages required for an external library
    // i.e. ECM_GETCOMPLIBINFO
  }
  return WNDdefWindowProc( hwnd, Msg, wParam, lParam, eci );
}
// ***** the htmlOutputClass implementation*****
// initialize static members
FARPROC htmlOutputClass::smPriDeviceProc = NULL;
```

```
qlong htmlOutputClass::smPriDeviceID = 0;
qfileptr htmlOutputClass::smTheDevice = NULL;
PRIjob htmlOutputClass::smJob = NULL;
// constructor
htmlOutputClass::htmlOutputClass( htmlOutputClass* pOutput )
{
  mDeviceIsMine = qfalse;
  if ( pOutput )
  {
    // if we had our own parameters we would copy their values
    // from the given instance
}
}
// destructor
htmlOutputClass::~htmlOutputClass()
{
}
// open device
qprierr htmlOutputClass::openDevice( PRIdestParmStruct* pDestParms )
{
  // first check if we are already open
  if ( htmlOutputClass::smTheDevice ) return PRI_ERR_ALREADY_OPEN_CLOSED;
  // delete existing file if it exists ( ignore error )
  FILEdelete( pDestParms->mTextFile );
  // now create the output file
  htmlOutputClass::smTheDevice = FILEcreateInst();
  if ( !htmlOutputClass::smTheDevice ) return PRI_ERR_MEMORY
  qret e = FILEcreate( htmlOutputClass::smTheDevice,
  pDestParms->mTextFile, qtrue );
  if ( e != e_ok )
  {
    FILEdestroyInst( htmlOutputClass::smTheDevice );
    htmlOutputClass::smTheDevice = NULL;
    return PRI_ERR_OMSERROR + e;
  }
  mDeviceIsMine = qtrue;
  // write the HTML header
  str255 txt("<html><body>");
  return writeDevice( &txt[1], txt[0] );
}
// close device
qprierr htmlOutputClass::closeDevice()
{
  qprierr err = PRI_ERR_NONE;
  if ( mDeviceIsMine )
  {
    // first check if we are already closed
    if ( ! htmlOutputClass::smTheDevice )
      return PRI_ERR_ALREADY_OPEN_CLOSED;
    // write the HTML end
    str255 txt("</body></html>");
    err = writeDevice( &txt[1], txt[0] );
    // close the device
    FILEclose( htmlOutputClass::smTheDevice );
    FILEdestroyInst( htmlOutputClass::smTheDevice );
    mDeviceIsMine = qfalse;
    htmlOutputClass::smTheDevice = NULL;
  }
  return err;
}
```

```
// write byte data to device
qprierr htmlOutputClass::writeDevice( qbyte* pData, qlong pDataLen, qbool pLineFeed );
{
  // first check if we are open
  if ( ! htmlOutputClass::smTheDevice ) return PRI_ERR_ALREADY_OPEN_CLOSED;
  // write the data
  qlong pos = FILEgetPosition( htmlOutputClass::smTheDevice );
  qret e = FILEwrite( htmlOutputClass::smTheDevice, pData, pos, pDataLen );
  if ( e != e_ok ) return PRI_ERR_OMSERROR + e;
  // write the line feed
  if ( pLineFeed )
  {
    static qbyte eol[2] = { 13, 10 };
    pos = FILEgetPosition( htmlOutputClass::smTheDevice );
    e = FILEwrite( htmlOutputClass::smTheDevice, &eol[0], pos, 2 );
    if ( e != e_ok ) return PRI_ERR_OMSERROR + e;
  }
  return PRI_ERR_NONE;
}
// write data from a qfldval to device
qprierr htmlOutputClass::writeDevice( qfldval pData )
{
  // construct a EXTfldval from the data
  EXTfldval fval( pData );
  // get a pointer of the data without making a copy of the data
  qHandle han = fval.getHandle( qfalse );
  qHandlePtr hp( han );
  // write data to device
  return writeDevice( &hp[0], hp.dataLen(), qtrue );
}
// send text direct to device
qprierr htmlOutputClass::sendText( qchar* pText, qlong pTextLen, qbool pLineFeed, qbool pFormFeed );
{
  // keep a static to remember if we need to start a new paragraph
  static qbool paraStarted = qfalse;
  // start new paragraph if we need to
  qprierr err = PRI_ERR_NONE;
  if ( ! paraStarted )
  {
    str15 para("<p>");
    err = writeDevice( &para[1], para[0], qfalse );
    if ( !err ) paraStarted = qtrue;
  }
  // write text to device
  if ( !err ) err = writeDevice( pText, pTextLen, qfalse );
  // end the paragraph if we have a line feed
  if ( !err && pLineFeed )
  {
    str15 paraEnd("</p>");
    err = writeDevice( &paraEnd[1], paraEnd[0], qtrue );
  }
  return err;
}
// send data to device
qprierr htmlOutputClass::sendData( qbyte* pData, qlong pDataLen )
{
  // we write the data as is
  return writeDevice( pData, pDataLen, qfalse );
}
```

Internal Output Devices

Omnis Studio ships with a number of fully functional output devices. These are devices which the user or developer can print to. In addition, there are three devices that exist purely to provide common functionality to internal and external output devices. All in all there are twelve internal devices from which an external device can inherit from. The following text lists all of these devices and explains the functionality these devices supply.

PRI_DEST_EXTBASE

This is the most basic device. All other devices inherit this device. It implements the following messages:

- **PM_OUT_DRAWPAGE** When this message is received by the device, it will generate PM_OUT_ADJPOS and PM_OUT_DRAWOBJECT messages for each object of the page. Objects which do NOT intersect the clipping area after the PM_OUT_ADJPOS call will NOT generate a PM_OUT_DRAWOBJECT message.
- **PM_OUT_GETEOL** Returns 0D for Macintosh, returns 0D and 0A for Windows.

PRI_DEST_EXTHDC

This device inherits from PRI_DEST_EXTBASE and implements functionality to paint all supported object types to a DC. The HDC must be supplied by the device which inherits this device. It implements the following messages:

- **PM_OUT_DRAWOBJECT** When this message is received, the device will draw the object to the HDC which must be set prior to this call.
- **PM_OUT_SET_HDC** This message can be sent to the device to set the HDC for drawing.

PRI_DEST_EXTTEXT

This device inherits from PRI_DEST_EXTBASE and implements functionality to format the text of a print job into pages of raw text. The text is positioned correctly by inserting spaces where necessary. This device generates a PM_OUT_SENDPAGE message, once a page is ready to be written to the device. All objects other then text are ignored by this device. It implements the following messages:

- **PM_OUT_ADDPAGE** Prepares a page buffer, then generates PM_OUT_DRAWPAGE and PM_OUT_SENDPAGE messages.
- **PM_OUT_ADJPOS** It alters the horizontal clipping for global objects, so all objects of the vertical page are included in the first horizontal page.
- **PM_OUT_DRAWOBJECT** When this message is received, the device will only add text objects to the current page of characters.

PRI_DEST_EXTPRINTER

This device inherits from PRI_DEST_EXTHDC and implements functionality to print to the printer. It implements the following messages:

- **PM_OUT_OPEN** Will open the device if not already open.
- **PM_OUT_CLOSE** Closes the device if it was opened by this instance.
- **PM_OUT_OPENDEVICE** Opens the printer document.
- **PM_OUT_CLOSEDEVICE** Closes the printer document.
- **PM_OUT_ISDEVICEOPEN** Returns 0 or 1.
- **PM_OUT_ADDPAGE** Opens a printer page, generates PM_OUT_DRAWPAGE and closes the printer page.
- **PM_OUT_SENDTEXT** Draws text to printer.

PRI_DEST_EXTPREVIEW

This device inherits from PRI_DEST_EXTHDC and implements functionality to display a page preview window on screen. It implements the following messages:

- **PM_OUT_OPEN** Opens a preview window instance.
- **PM_OUT_CLOSE** Closes a preview window instance.
- **PM_OUT_ADDPAGE** Adds to the scroll range of the preview window.

Note: This device generates PM_OUT_DRAWPAGE messages when receiving paint messages for its HWNDs. This device may be opened many times by one or more print jobs. Each instance will open a new window on screen.

PRI_DEST_EXTSCREEN

Same as PRI_DEST_EXTPREVIEW, except display is not paged. Additionally it implements the following message.

- **PM_OUT_ADJPOS** It alters the horizontal clipping for global objects, so all objects of the vertical page are included in the first horizontal page. It also manipulates the left and top of the object according to the scroll positions of the window.

PRI_DEST_EXTDISK

This device inherits from PRI_DEST_EXTHDC and implements functionality to save a report to a disk file. It implements the following messages:

- **PM_OUT_OPEN** Opens the destination file
- **PM_OUT_CLOSE** Closes the destination file.
- **PM_OUT_ADDPAGE** Prepares a page buffer, then generates a PM_OUT_DRAWPAGE and writes buffered page to disk.
- **PM_OUT_DRAWOBJECT** Adds object to current page buffer. All external objects and picture objects are converted to color shared pictures.
- **PM_OUT_BROWSE** Opens a pick file dialog.

PRI_DEST_EXTMEM

Same as PRI_DEST_EXTDISK. Job is saved to memory instead.

PRI_DEST_EXTCLIPBOARD

This device inherits from PRI_DEST_EXTTEXT. It implements the following messages:

- **PM_OUT_OPEN** Opens the device if not open.
- **PM_OUT_CLOSE** Closes the device if open.
- **PM_OUT_OPENDEVICE** Prepares a memory handle for buffering clipboard data.
- **PM_OUT_CLOSEDEVICE** Writes buffered data to clipboard.
- **PM_OUT_ISDEVICEOPEN** Returns 0 or 1.
- **PM_OUT_SENDPAGE** Sends the page to the clipboard.
- **PM_OUT_SENDTEXT** Buffers text in memory.

PRI_DEST_EXTPORT

This device inherits from PRI_DEST_EXTTEXT. All output is send to the destination port as specified by the device parameters. It implements the following messages:

- **PM_OUT_OPEN** Opens the device if not open.
- **PM_OUT_CLOSE** Closes the device if open.
- **PM_OUT_OPENDEVICE** Opens the port.
- **PM_OUT_CLOSEDEVICE** Closes the port.
- **PM_OUT_FLUSHDEVICE** Flushes the port.
- **PM_OUT_ISDEVICEOPEN** Returns 0 or 1.
- **PM_OUT_SENDPAGE** Sends the page to the port.
- **PM_OUT_SENDTEXT** Sends text to the port.
- **PM_OUT_SENDDATA** Sends data to the port.
- **PM_OUT_GETEOL** Returns 0D,0A for all platforms.

PRI_DEST_EXTFILE

This device inherits from PRI_DEST_EXTTEXT. All output is sent to the destination file as specified by the device parameters. It implements the following messages:

- **PM_OUT_OPEN** Opens the device if not open.
- **PM_OUT_CLOSE** Closes the device if open.
- **PM_OUT_OPENDEVICE** Opens the destination file.
- **PM_OUT_CLOSEDEVICE** Closes the destination file.
- **PM_OUT_FLUSHDEVICE** Flushes the file.
- **PM_OUT_ISDEVICEOPEN** Returns 0 or 1.
- **PM_OUT_SENDPAGE** Writes the page to the file.
- **PM_OUT_SENDTEXT** Writes text to the file.
- **PM_OUT_SENDDATA** Writes data to the file.
- **PM_OUT_BROWSE** Opens a pick file dialog.

PRI_DEST_EXTDDE_PUB (MAC)

This device inherits from PRI_DEST_EXTBASE. It implements the following messages:

- **PM_OUT_OPEN** Opens the device and publisher file if not open.
- **PM_OUT_CLOSE** Closes the device and publisher file if open.
- **PM_OUT_ADDPAGE** Generates a PM_OUT_DRAWPAGE.
- **PM_OUT_DRAWOBJECT** When this message is received, the device will write the text of text objects to the publisher file.

Structures, Data types and Defines

ePRIpos

This is the enumeration of the report position mode of the qpripos structure. It can be one of the following values:

- **ePosGlobal** the position is global to the print job, relative to the top-left of the local area of the first page.
- **ePosPaper** the position is relative to the top-left of the paper edge of the page specified by mVertPage and mHorzPage.
- **ePosPrintable** the position is relative to the top-left of the printable area of the page specified by mVertPage and mHorzPage.
- **ePosLocal** the position is relative to the top-left of the local area of the page specified by mVertPage and mHorzPage.
- **ePosSection** the position is relative to the top-left of the section specified by mSectionId.
- **ePosHeader** the position is relative to the top-left of the header area of the page specified by mVertPage and mHorzPage.
- **ePosFooter** the position is relative to the top-left of the footer area of the page specified by mVertPage and mHorzPage.

In addition, the following enumerators can be used with the PRIconvPos function to return the co-ordinates of an area on the page specified by mVertPage and mHorzPage.

- **eBndsGlobal** returns ePosGlobal co-ordinates. The top, left, width, and height are calculated to global coordinates of the local area of the page.
- **eBndsPaper** returns ePosPaper coordinates. The top and left are zero, and the height and width are calculated to the height and width of the paper of the page.
- **eBndsPrintable** returns ePosPrintable coordinates. The top and left are zero, and the height and width are calculated to the height and width of the printable area of the page.
- **eBndsLocal** returns ePosLocal coordinates. The top and left are zero, and the height and width are calculated to the height and width of the local area of the page.
- **eBndsHeader** returns ePosHeader coordinates. The top and left are zero, and the height and width are calculated to the height and width of the header area of the page.
- **eBndsFooter** returns ePosFooter coordinates. The top and left are zero, and the height and width are calculated to the height and width of the footer area of the page.

PRI_ERR_xxx

These are the print manager defined error codes. Most print manager functions have an error return value called qprierr. The errors are divided into two types. Fatal and non-fatal errors. When fatal errors occur during a print job function call, the error is additionally stored with the print job. Any subsequent calls to print job related functions will return without any further action once a fatal error has occurred. It is possible to clear the error condition of a print job by calling PRIsetError with PRI_ERR_NONE.

The errors listed next are internal non-fatal errors.

- **PRI_ERR_NONE** no error
- **PRI_ERR_INVALID_JOB** the specified PRIjob is not a valid job (it may have been deleted or NULL was specified)
- **PRI_ERR_INVALID_DEST** the specified report destination (output device) does not exist
- **PRI_ERR_INVALID_PROCINST** no PRIprocClass pointer specified for PRIstartJob
- **PRI_ERR_INVALID_REPDATA** the specified report data passed to PRIloadJob or report data destination passed to PRIredirectJob is not valid
- **PRI_ERR_INVALID_PARM** one of the parameters passed to a function or as part of a message is not valid
- **PRI_ERR_INVALID_DRVINFO** the given PRIpageSetup structure does not contain valid driver data or the specified flattened driver data is not valid

- **PRI_ERR_INSUFFICIENT_BUFFER** the destination buffer passed to PRIflattenDriverInfo is not large enough
- **PRI_ERR_INVALID_POS** the specified report position is not valid (page has been ejected or page/position is out of range)
- **PRI_ERR_INVALID_SECTION** the specified section does not exist
- **PRI_ERR_DUP_SECTION** attempt to add a section with an id which is already used by an existing section
- **PRI_ERR_SECTION_NOT_FOUND** specified section does not exist
- **PRI_ERR_NOT_IMPLEMENTED** feature has not been implemented
- **PRI_ERR_PAGE_INIT** attempt to change a print jobs page setup after the current page has been initialized
- **PRI_ERR_ALREADY_OPEN_CLOSED** attempt to open an already open, or close an already closed output device
- **PRI_ERR_TOMANY_OUTPUTS** attempt to register too many outputs (maximum is 32 including internal devices)
- **PRI_ERR_PAGE_CLOSED** attempt to set page information (PRIsetPageInfo) for a page which has already been closed

The errors listed next are internal fatal errors.

- **PRI_FATAL_START** first fatal error code
- **PRI_FATAL_END** last fatal error code
- **PRI_ERR_MANAGER_CLOSED** returned when calling a print manager function and the print manager has not been initialized by calling PRIopen
- **PRI_ERR_MEMORY** there was insufficient memory to complete the current operation
- **PRI_ERR_ABORT** the user has aborted the print job
- **PRI_ERR_BUFFER_READ** reported when an error occurred during page buffering reads
- **PRI_ERR_BUFFER_ WRITE** reported when an error occurred during page buffering writes

The next set of error defines are special return values and additional error flags allowing additional ranges of errors to be reported.

- **PRI_ERR_IGNORE** this error can be returned when custom devices receive the PM_OUT_SETDATA message to prevent any further action on return
- **PRI_ERR_SYSERROR** if this bit of an error code is set, the low 24 bits contain a system error code
- **PRI_ERR_OMSERROR** if this bit of an error code is set, the low 24 bits contain an Omnis error code which was returned to the print manager by another part of Omnis outside the print manager
- **PRI_ERR_CUSTOM** if this bit of an error code is set, the low 16 bits contain a non fatal custom device error, the low byte of the high word contains the custom device id
- **PRI_ERR_CUSTOM_FATAL** if this bit of an error code is set, the low 16 bits contain a fatal custom device error, the low byte of the high word contains the custom device id
- **PRI_ERR_SYSMASK** mask for retrieving system and Omnis error codes from a qprierr
- **PRI_ERR_NEGATIVE** mask for retrieving the negative sign of a system or Omnis error code from a qprierr
- **PRI_ERR_CUSTOM_MASK** mask for retrieving the custom device error code from a qprierr
- **PRI_ERR_CUSTOM_IDMASK** mask for retrieving the custom device id from a qprierr

PRI_xxx

These are predefined measurements to aid with conversions of measurements.

- **PRI_1_8INCH** Number of qpridims per one eighth of an inch.

(3175)

- **PRI_1_4INCH** Number of qpridims per one quarter of an inch.

(6350)

- **PRI_1_2INCH** Number of qpridims per one half of an inch.

(12700)

- **PRI_INCH** Number of qpridims per one inch. (25400)
- **PRI_POINT** Number of qpridims per point (1/72 inch). (352.77 approx.)
- **PRI_MM_PER_INCH** Number of millimeters per inch. (25.400)
- **PRI_INCH_PER_MM** Number of inches per millimeter. (0.03937 approx.)

PRIdestParmStruct

The PRIdestParmStruct structure contains the destination parameters for the output devices.

```
struct PRIdestParmStruct
{
  qlong    mDest; // the destination, one of the PRI_DEST_xxx defines
  qlong     mReserved1; // reserved
  PRIpageSetup* mPageSetup; // the page setup for the print job
  HWND     mHwnd; // hwnd for PRI_DEST_SCREEN and _PREVIEW
  qrect     mWRect;      // size of window for screen or preview
  str255     mRepFile;      // file name for PRI_DEST_DISK
  str255     mTextFile; // file name for PRI_DEST_FILE
  str255     mEdFile;      // edition file name for PRI_DEST_DDE_PUB
  str255     mPages;      // pages to be printed
  qfldval     mRepData; // destination for report data for PRI_DEST_MEM
  qpridim     mCharWidth; // character width for text devices
  qpridim     mLineHeight;// line height for for text devices
  qfldval     mExtParms; // output parameters for custom devices
  PRIportParmStruct     mPortParms; // port parameters
  qshort     mLinesPerPage;      // lines per page for text devices
  qshort     mCharsPerLine;      // characters per line for text devices
  qshort      mRtimeout;      // Port Timeout in seconds.
  qshort     mReserved3;      // reserved
  qbool     mGeneratePages:1;      // generate page headers and footers
  qbool     mSendFormFeed:1;      // send form feed (text devices)
  qbool     mRestrictPageWidth:1;      // text devices only
  qbool     mAppendFile:1;      // append file (PRI_DEST_FILE only)
  qbool     mIsText:1;      // only print text
  qbool     mStack:1;      // stack preview or screen report window
  qbool     mCenter:1;      // open screen or preview in center
  qbool     mShowBounds:1;      // show page bounderies (screen reports)
  qbool     mOpenModal:1; // open screen or preview modally
  qbool     mHide:1;      // hide screen or preview until complete
  qbool     mReserved4:1; // reserved
  qbool     mReserved5:1; // reserved
  qbool     mReserved6:1; // reserved
```

```
  qbool     mReserved7:1; // reserved
  qbool     mReserved8:1; // reserved
  qbool     mReserved9:1; // reserved
  PRIdestParmStruct() {}
  PRIdestParmStruct( qniltype qnil );
  ~PRIdestParmStruct() {}
};
```

The **PRIdestParmStruct** structure has been modified in Studio 11. Therefore, any xcomps that use PRIdestParmStruct must be rebuilt with the Studio 11 SDK. The following members of PRIdestParmStruct have been made private: mEdFile, mTextFile and mRepFile. As such, accessor methods have been added to the PRIdestParmStruct class:

void getRepFile(EXTfldval& pRetVal) const;
void setRepFile(EXTfldval& pNewVal);

void getTextFile(EXTfldval& pRetVal) const;
void setTextFile(EXTfldval& pNewVal);

void getEdFile(EXTfldval& pRetVal) const;
void setEdFile(EXTfldval& pNewVal);

These can be used by xcomps to retrieve and set the respective private members.

mDest specifies the destination device. It can be one of the following:

- **PRI_DEST_PRINTER** Report is printed to the printer.
- **PRI_DEST_PREVIEW** Report is printed to the page preview. Device parameters are mHwnd, mWRect, mStack, mCenter, mOpenModal, mHide.
- **PRI_DEST_SCREEN** Report is printed to the screen. Device parameters are mHwnd, mWRect, mStack, mCenter, mOpenModal, mHide.
- **PRI_DEST_DISK** Report is saved to a file on disk, so it can be loaded and printed later by calling PRIloadJob. Device parameters are mRepFile.
- **PRI_DEST_MEM** Same as PRI_DEST_DISK except that it is saved to a memory variable. Device parameters are mRepData.
- **PRI_DEST_CLIPBOARD** Report is printed to the clipboard. Only text is sent and the report is NOT paged. Device parameters are mCharWidth, mLineHeight.
- **PRI_DEST_PORT** Report is printed to the port. Device parameters are mPortParms, mCharWidth, mLineHeight, mLinesPerPage, mCharsPerLine, mGeneratePages, mSendFormFeed, mRestrictPageWidth.
- **PRI_DEST_FILE** Report is printed to a text file. Device parameters are mCharWidth, mLineHeight, mLinesPerPage, mCharsPerLine, mGeneratePages, mSendFormFeed, mRestrictPageWidth, mAppendFile.
- **PRI_DEST_DDE_PUB** Report is printed to publisher on Macintosh and DDE on windows. Device parameter is mEdFile on Macintosh only.

Additionally there may be custom devices which have been registered with the print manager, starting at PRI_DEST_CUSTOM_FST and continuing up to PRI_DEST_CUSTOM_LST.

mPageSetup points to the page setup information for the print job. If specified, the print manager will make a copy so the given page setup info can be safely deleted. If it is NULL, the print manager will use its own page setup information.

mHwnd specifies an alternative window for the screen report field or page preview field to appear. The device will take over this HWND while it is active. The WndProc method of the HWND will receive a WM_PRI_INSTALL (WM_USER+29) message when the device constructs, and a WM_PRI_REMOVE (WM_USER+30) when the device destructs.

mWRect if this rectangle is NOT empty it specifies the position and size of the preview or screen report window if mHwnd is NULL. Use GDIsetRectEmpty to clear this member.

mRepFile specifies the destination path and file name when mDest is PRI_DEST_DISK. If it is left empty, the user will be prompted when PRIstartJob is called.

mTextFile specifies the destination path and file name when mDest is PRI_DEST_FILE. If it is left empty, the user will be prompted when PRIstartJob is called.

mEdFile specifies the destination path and file name when mDest is PRI_DEST_DDE_PUB and the platform is Macintosh. If it is left empty, the user will be prompted when PRIstartJob is called.

mPages specifies the pages or ranges of pages to send to the device. The formatting manager will still build all pages in order to correctly format each page, but it will only eject the specified pages to the device. It does mean however that the formatting manager does not eject pages until the print job is complete.

Example ranges:

"1,3,5,2,10-20,o30-40,e40-30"

The 'o' stands for odd pages and 'e' stands for even pages the '-' specifies a range of pages.

mRepData specifies the storage for receiving the report data when mDest is PRI_DEST_MEM. If it is not set, an error will be returned.

mCharWidth specifies the width of a character in qpridims for all text-based devices.

mLineHeight specifies the height of a text line in qpridims for all text-based devices.

mExtParms contains the parameters for all custom devices. Do NOT manipulate this data from C++. There is currently no support for modifying these parameters safely outside the Omnis notation and the Destination dialog.

mPortParms contains the parameters for the PRI_DEST_PORT device.

This structure is defined as follows:

```
struct PRIportParmStruct
{
  qlong mPort;    // port number
  qulong mSpeed;    // port speed
  qulong mHandShake:2;  // 0 = No hand shake, 1 = xon/xoff, 2 = Hardware
  qulong mParity:2;  // 0 = No parity, 1 = odd, 2 = even
  qulong mDataBits:1;  // 0 = 7 data bits, 1 = 8 data bits
  qulong mStopBits:1;  // 0 = 1 stop bit, 1 = 2 stop bits
  qulong mReserved1;   // Reserved for future use.
  qulong mReserved2;   // Reserved for future use.
  str255 mPortName;    // Port Name.
  str255 mPortProfile; // Port Profile Name.
};
```

mLinesPerPage is used by text-based print jobs when mGeneratePages is true. It specifies the number of lines of text which fit on a single page.

mCharsPerLine is used by text-based print jobs when mGeneratePages and mRestrictPageWidth are both true. It specifies the maximum number of characters which fit on a line.

mRtimeout is used by the PRI_DEST_PORT device. It is a value which represents the duration in seconds that Omnis should wait before aborting the current port operation. Setting this value to zero will switch off timeouts.

mGeneratePages if true the formatting manager will generate page headers and footers for each page, otherwise only one page header for the first page is generated.

mSendFormFeed is used by some text-based devices. A form feed character is generated by these devices at the end of every page. mGeneratePages must be true.

mRestrictPageWidth is used by some text-based devices. If true and mGeneratePages is true, the width of a line is restricted by the number of mCharsPerLine characters.

mAppendFile is used by the PRI_DEST_FILE. If true the file specified by mTextFile is always appended to, regardless of whether the device is open or not. If it is false and the device is closed, the file is replaced.

mIsText if true forces the print manager to generate a text-based report. All the text based page formatting applies.

mStack is used by the PRI_DEST_PREVIEW and PRI_DEST_SCREEN devices. If true the window when opened will be stacked.

mCenter is used by the PRI_DEST_PREVIEW and PRI_DEST_SCREEN devices. If true the window when opened will be centered on screen.

mShowBounds is used by the PRI_DEST_SCREEN device. If true the window when opened will show page boundaries.

mOpenModal is used by the PRI_DEST_PREVIEW and PRI_DEST_SCREEN devices. If true the window when opened will be modal (execution stops until the window is closed).

mHide is used by the PRI_DEST_PREVIEW and PRI_DEST_SCREEN devices. If true the window will remain hidden until the print job is complete.

PRIdeviceInfoStruct

The device info structure describes internal and custom devices. It is used to register and change various properties of a device.

```
struct PRIdeviceInfoStruct
{
  // read only properties for PRIgetDeviceInfo (except on registration)
  qlong mID; // << unique ID (PRI_DEST_FST to PRI_DEST_LST)
  // if non-zero on registration, the standard
  // device is replaced by custom device
  HBITMAP mBitmap; // >> bitmap displayed in dialog (custom devices)
  void* mExtInfo; // >> info provided by custom devices
  qlong mExtCtrlID; // >> info provided by custom devices
  qbool mIsOpen:1; // << device is open
  qbool mIsText:1; // >> its a text based device
  qbool mCanOpenDirect:1; // >> device can be opened via PRIopenDevice
  qbool mCanKeepOpen:1; // >> if true device is opened on selection
  qbool mCanSendText:1; // >> if true it supports PRIsendTextToDevice
  qbool mCanSendData:1; // >> if true it supports PRIsendDataToDevice
  qbool mCanPage:1; // >> if false does not support paged reports
  qbool mKeepPages:1; // >> if true pages are buffered until close
  // read/write properties for PRIsetDeviceInfo
  str31 mName; // >> name shown in dialog
  qlong mIconID; // >> if non-zero shows the icon in dialog
  qbool mShowInDialog:1; // >> will appear in destination dialog
};
```

mID is a unique identifier in the range PRI_DEST_FST to PRI_DEST_LST. It is decided by the print manager when a device is registered. It is simply the order of registration. Internal devices are always registered in the same order, so their id is guaranteed. The id of a custom device may change if more custom devices are added.

mBitmap is provided by custom devices as the icon to be displayed in the destination dialog.

mExtInfo must be provided by custom devices when registering. Use

```
deviceInfo.mExtInfo = eci->mPrivate;
```

to supply this information.

mExtCtrlID must be provided by custom devices when registering. It is currently not used and must be set to zero.

mIsOpen if it is true, the device is currently open. This information is provided by the device manager on a call to PRIgetDeviceInfo only. It is ignored on registration.

mIsText if true tells the device manager that this device is a text only device.

mCanOpenDirect if true the device supports calls to PRIopenDevice.

mCanKeepOpen if true tells us that the device can be opened by Omnis automatically when picked via the destination dialog. It is recommended that custom devices always specify qfalse. It has only been provided for backward compatibility with Omnis Classic.

mCanSendText if true the device supports calls to PRIsendTextToDevice.

mCanSendData if true the device supports calls to PRIsendDataToDevice.

mCanPage if false the device does not support paged reports. This means that only one page header message is generated for the first page of the print job.

mKeepPages if true, the print manager will buffer all pages until the print job is complete and the device is closed. If false, as each page is ejected to the device, it is destroyed. Typically the device will then be closed by the print manager when the print job has been closed and the last page has been ejected.

mName on registration specifies the internal name of the device, and the name to be displayed in the destination dialog. Once a device is registered it is only used to refer to the name shown in the destination dialog.

mIconID if zero, the bitmap specified by mBitmap is shown in the destination dialog. If non-zero the icon specified by the id is shown instead.

mShowInDialog if true, the device will appear in the destination dialog.

PRIobjectStruct

The PRIobjectStruct structure specifies the properties of an object.

```
struct PRIobjectStruct
{
  // general properties
  qlong mIdent; // >> user identifier
  qlong mType; // >> the objects type (PRI_OBJ_xxx)
  qlong mUnique; // << unique id generate by print manager
  qpripos mPos; // <<>> report position of object
  qfldval mData; // >> objects data
  // image properties
  qdim mHorzDPI; // >> the images horizontal dots per inch
  qdim mVertDPI; // >> the images vertical dots per inch
  // fill properties
  qpat mFillPat; // >> fill pattern
  qcol mBackFillColor; // >> background fill color
  qcol mForeFillColor; // >> foreground fill color
  // border and line properties
  WNDborderStruct mBorder; // >> border style
  // text properties
  qshort mStyleIndex; // >> index into Omnis style table
  qshort mFontIndex; // >> index into Omnis font table
  GDItextSpecStruct* mTextSpec; // >> pointer to text style
  // other
  qbool mHorzSlide:1; // >> horizontal sliding
  qbool mHorzExtend:1; // >> grows horizontally to fit text
  qbool mVertExtend:1; // >> grows vertically to fit text
  qbool mFloatRight:1; // >> right edge can float
  qbool mFloatBottom:1; // >> bottom edge can float
  qbool mMultiLine:1; // >> paint as multi line text
  qbool mScreenUnits:1; // >> scale object to screen units
  qbool mAddEllipsis:1; // >> if text is too long ellipsis are added
  qbool mFitVertPage:1; // >> fit object on page vertically
  qbool mGrowSection:1; // >> grow section if object grows
  qbool mZeroEmpty:1; // >> zero numbers display empty
  qbool mJstText:1;    // >> Omnis styled text
  qbool mIsMultiLine:1; // << multi line text
  qbool mIsLastMultiLine:1; // << last row of multiline object
  qbool mLineHadCR:1;  // << row had CR character
  qbool mLineHadLF:1;  // << row had LF character
  qbool mLineWasWrap:1; // << row was wrapped
};
```

mIdent is a identifier for your use.

mType is the object type. This can be one of the following.

- **PRI_OBJ_BITMAP** Paints a HBITMAP. mData must specify the HBITMAP. mHorzExtend if false will stretch the bitmap to the horizontal bounding rectangle of the object. mVertExtend if false will stretch the bitmap to the vertical bounding rectangle. mScreenUnits if true will scale the bitmap to screen units. Otherwise the bitmap is drawn in device units. If mHorzExtend and mVertExtend are false, there should be no need to print the bitmap in screen units since the bitmap will be stretched to fit the bounding rect.

- **PRI_OBJ_BORDER** Paints a border. See hwnd module for a full listing of border styles. mBorder must specify the border properties. mFillPat must specify the pattern to be used for the fill inside of the border. mBackFillColor must specify the clear pixel color for the fill. mForeFillColor must specify the set pixel color for the fill. mScreenUnits if true indicates that the pen size and pattern size is in screen units. Otherwise qpridim units are assumed. On the Macintosh the standard gray shade patterns will not be scaled to screen units and will always print in device units.

- **PRI_OBJ_EXTERNAL** External objects are not painted by the print manager. When an external object requires painting, a message is send to the message procedure. See PM_PAINT_OBJECT.

- **PRI_OBJ_LINE** Paints a line. The left/top coordinates specify the starting point, and the right/bottom coordinates specify the ending point. mBorder.mLineStyle must specify the pen to be used to draw the line. mScreenUnits if true indicates that the pen size is in screen units. Otherwise qpridim units are assumed.

- **PRI_OBJ_OVAL** Paints an oval shape. Properties are as for PRI_OBJ_RECT.

- **PRI_OBJ_PGCNT** Special object for printing things like Page x out of y. If this object is added to the print job, all page ejection is halted and the pages are buffered until the last page has been prepared. The text given to this object should contain the two token characters PRI_OBJ_PGCNT_CNT and PRI_OBJ_PGCNT_PAGE. PRI_OBJ_PGCNT_CNT is the place holder for the total number of pages, and PRI_OBJ_PGCNT_PAGE is the place holder for the current page.

- **PRI_OBJ_PICTURE** Paints a picture as stored in an Omnis picture field. Properties are as for PRI_OBJ_BITMAP. mData must contain the picture data.

- **PRI_OBJ_PIXMAP** Paints a HPIXMAP. Properties are as for PRI_OBJ_BITMAP. mData must specify the HPIXMAP.

- **PRI_OBJ_RECT** Paints a simple rectangle. mBorder.mLineStyle must specify the pen to be used to draw the rectangle outline. mFillPat must specify the pattern to be used for the fill inside of the rectangle. mBackFillColor must specify the unset pixel color for the fill. mForeFillColor must specify the set pixel color for the fill. mScreenUnits if true indicates that the pen size and pattern size is in screen units. Otherwise qpridim units are assumed. On the Macintosh the standard gray shade patterns will not be scaled to screen units and will always print in device units.

- **PRI_OBJ_ROUNDRECT** Paints a rectangle with rounded edges. Properties are as for PRI_OBJ_RECT.

- **PRI_OBJ_TEXT** Paints a run of text. mData must contain character data. mFnt or mFontIndex or mStyleName must specify the font to be used. mJst must contain the text justification. mTextColor must contain the text color. mHorzSlide must be set to qtrue if horizontal sliding is required. mHorzExtend must be set to qtrue if the text is NOT to be clipped horizontally to the object's bounding rectangle. The bounding rectangle is grown to make the text fit horizontally. This will also grow the section it belongs to and has an effect on objects with floating properties. mVertExtend must be set to qtrue if the text is NOT to be clipped vertically to the object's bounding rectangle. The bounding rectangle is grown to make the text fit vertically. This will also grow the section it belongs to and has an effect on objects with floating properties. mMultiLine must be set to qtrue if the object is to draw the text on several lines. If the text is to wrap inside the bounding rectangle, mHorzExtend must be set to qfalse.

- **PRI_OBJ_COMMENT** This object has no appearance, and no special formatting will take place when added. It is a mechanism to insert comments for output devices. Currently none of the internal devices support comments, but some custom devices may do. The comment data must be of type text and the first word must be the device internal name followed by a device-specific comment identifier. The device name and identifier should be connected via an underscore. Following the identifier should be the actual comment or information.

  Example:

```
"HTML_OBJ http://www.omnis-software.com/"
```

mPos specifies the object's report position. If the object can extend horizontally or vertically or horizontally slide, etc., mPos is updated by the print manager prior to returning from PRIaddObject.

mData is dependent on the object type. Use EXTfldval::getFldVal() to set this member.

mHorzDPI specifies the horizontal dots per inch of a picture, bitmap or pixmap object. If set to zero, the print manager will assume screen resolution. This value is used, when the object can horizontally extend, to calculate the new width of the object based on the images horizontal size.

mVertDPI specifies the vertical dots per inch of a picture, bitmap or pixmap object. If set to zero, the print manager will assume screen resolution. This value is used, when the object can vertically extend, to calculate the new height of the object based on the images vertical size.

mFillPat is the background fill pattern. Not used for text objects. External objects may use this property.

mBackFillColor is the color in which all unset pixels of the fill pattern are drawn. Not used for text objects. External objects may use this property.

mForeFillColor is the color in which all set pixels of the fill pattern are drawn. Not used for text objects. External objects may use this property.

mBorder specifies the border properties and line style for background objects.

mStyleIndex is the index into the Omnis style table. If this property is NOT zero, it overrides the mFontIndex and mTextSpec properties. The print manager will store the style data for all platforms with the report so that a saved report can be printed from other platforms. External objects may use this property.

mFontIndex is the index into the report font table. If this property is NOT zero, it overrides the font specified by mTextSpec. The print manager stores the font information for all platforms at that index so that a saved report can be printed from other platforms. External objects may use this property.

mTextSpec specifies the text font, size, style, justification, and color.

mHorzSlide specifies the horizontal sliding property for text objects. Text objects will slide horizontally, if previous text objects with the same top coordinate have been horizontally sized to fit their text. The object is moved by the same amount the previous object(s) have been sized. Objects can be sized in both directions and therefore sliding can occur in either direction. (PRI_OBJ_TEXT only)

mHorzExtend specifies whether the object can extend horizontally to fit its text or image. If it is false, text is clipped and images are scaled.

mVertExtend specifies whether the object can extend vertically to fit its text or image. If it is false, text is clipped and images are scaled.

mFloatRight specifies whether an objects right edge will move when the section it belongs to grows horizontally.

mFloatBottom specifies whether an objects bottom edge will move when the section it belongs to grows vertically.

mMultiLine specifies whether the text objects contains more then one line of text (contains carriage return characters). If mHorzExtend is false, the text is also wrapped to fit the objects horizontal boundaries. (PRI_OBJ_TEXT only)

mScreenUnits specifies whether the object is to be drawn in device units, or if true in screen units.

mAddEllipsis if true and the object is a single line text object and mHorzExtend is false and the text does not fit within the specified boundaries, the text is cut short and ellipsis are appended. (PRI_OBJ_TEXT only)

mFitVertPage if true and the objects boundaries are global and do not fit entirely on the current page, the objects boundaries are moved so the object fits at the top of the global area of the next page.

mGrowSection if true and the object is inserted into a section, the section is grown if the object boundaries would appear outside the sections boundaries. This action will grow any objects whose mFloatRight or mFloatBottom properties are true.

mZeroEmpty if true zero number values will be drawn empty. (PRI_OBJ_TEXT only)

mJstText if true text may contain style characters. The print manager will use GDIdrawTextJst to draw the text.

mIsMultiLine this flag may be set by the formatting manager when it sends an object to an output device. When multi line text (mMultiLine) is added to a print job, the formatting manager separates the multi line object into one or more single line text objects. It does, however, store certain multi line details with the object so that output devices can re-assemble the multi line object if they require to do so. If mIsMultiLine is true, this indicates to an output device that this object is part of a serious of single row objects which originated from one multi line object.

mIsLastMultiLine this flag is set by the formatting manager if mIsMultiLine is true and it is the last of a serious of single line objects which originated from the same multi line object.

mLineHadCR is true if the original multi line text terminated this line with a CR character.

mLineHadLF is true if the original multi line text terminated this line with a LF character.

mLineWasWrap is true if the original multi line text terminated this line because of word wrapping.

PRIpageSetup

The PRIpageSetup structure then page formatting properties.

```
struct PRIpageSetup
{
  qshort mFields;     // <<>> specifies set properties
  qshort mOrientation; // <<>> paper orientation
  qshort mPaperSize;   // <<>> paper size (i.e. PRI_PAPER_A4)
  qpridim mPaperLength; // <<>> paper length in qpridims
  qpridim mPaperWidth; // <<>> paper width in qpridims
  qshort mHorzScale;  // <<>> horizontal scale in percent
  qshort mVertScale;  // <<>> vertical scale in percent
  qshort mCopies;    // <<>> number of copies to print
  PRIdrvInfo mDriverInfo; // << driver information
};
```

mFields specifies which of the properties are set. It can be a combination of the following values. Use the and operator '&' to test the values and the or operator '|' to set these values.

- **PRI_PS_ORIENT** mOrientation is set
- **PRI_PS_PAPER** mPaperSize is set
- **PRI_PS_PAPERL** mPaperLength is set
- **PRI_PS_PAPERW** mPaperWidth is set
- **PRI_PS_HSCALE** mHorzScale is set
- **PRI_PS_VSCALE** mVertScale is set
- **PRI_PS_COPIES** mCopies is set

mOrientation specifies the paper orientation. It can be one of the following:

- **PRI_OR_PORTRAIT** portrait orientation
- **PRI_OR_LANDSCAPE** landscape orientation

mPaperSize specifies the paper size. It can be one of the following:

- **PRI_PA_LETTER** Letter, 8 1/2 by 11 inches
- **PRI_PA_LEGAL** Legal, 8 1/2 by 14 inches
- **PRI_PA_A4** A4 Sheet 210 by 297 mm
- **PRI_PA_CSHEET** C Sheet, 17 by 22 inches
- **PRI_PA_DSHEET** D Sheet, 22 by 34 inches
- **PRI_PA_ESHEET** E Sheet, 34 by 44 inches
- **PRI_PA_LETTERSMALL** Letter small, 8 1/2 by 11 inches
- **PRI_PA_TABLOID** Tabloid, 11 by 17 inches
- **PRI_PA_LEDGER** Ledger, 17 by 11 inches
- **PRI_PA_STATEMENT** Statement, 5 1/2 by 8 1/2 inches
- **PRI_PA_EXECUTIVE** Executive 7 1/4 by 10 1/2 inches
- **PRI_PA_A3** A3 sheet, 297 by 420 mm

- **PRI_PA_A4** SMALL A4 small sheet, 210 by 297 mm
- **PRI_PA_A5** A5 sheet, 148 by 210 mm
- **PRI_PA_B4** B4 sheet, 250 by 354 mm
- **PRI_PA_B5** B5 sheet, 182 by 257 mm
- **PRI_PA_FOLIO** Folio, 8 1/2 by 13 inches
- **PRI_PA_QUARTO** Quarto, 215 by 275 mm
- **PRI_PA_10X14** 10 by 14 inch sheet
- **PRI_PA_11X17** 11 by 17 inch sheet
- **PRI_PA_NOTE** Note, 8 1/2 by 11 inches
- **PRI_PA_ENV_9** #9 Envelope, 3 7/8 by 8 7/8 inches
- **PRI_PA_ENV_10** #10 Envelope, 4 1/8 by 9 1/2 inches
- **PRI_PA_ENV_11** #11 Envelope, 4 1/2 by 10 3/8 inches
- **PRI_PA_ENV_12** #12 Envelope, 4 3/4 by 11 inches
- **PRI_PA_ENV_14** #14 Envelope, 5 by 11 1/2 inches
- **PRI_PA_ENV_DL** DL Envelope, 110 by 220 mm
- **PRI_PA_ENV_C5** C5 Envelope, 162 by 229 mm
- **PRI_PA_ENV_C3** C3 Envelope, 324 by 458 mm
- **PRI_PA_ENV_C4** C4 Envelope, 229 by 324 mm
- **PRI_PA_ENV_C6** C6 Envelope, 114 by 162 mm
- **PRI_PA_ENV_C65** C65 Envelope, 114 by 229 mm
- **PRI_PA_ENV_B4** B4 Envelope, 250 by 353 mm
- **PRI_PA_ENV_B5** B5 Envelope, 176 by 250 mm
- **PRI_PA_ENV_B6** B6 Envelope, 176 by 125 mm
- **PRI_PA_ENV_ITALY** Italy Envelope, 110 by 230 mm
- **PRI_PA_ENV_MONARCH** Monarch Envelope, 3 7/8 by 7 1/2 inches
- **PRI_PA_ENV_PERSONAL** 6 3/4 Envelope, 3 5/8 by 6 1/2 inches
- **PRI_PA_FANFOLD_US** Std Fanfold, 14 7/8 by 11 inches
- **PRI_PA_FANFOLD_STD_GER** German Std Fanfold, 8 1/2 by 12 inches
- **PRI_PA_FANFOLD_LGL_GER** German Legal Fanfold, 8 1/2 by 13 inches
- **PRI_PA_ISO_B4** B4 (ISO) 250 x 353 mm
- **PRI_PA_JAPANESE_POSTCARD** Japanese Postcard 100 x 148 mm
- **PRI_PA_9x11** 9 x 11 inches
- **PRI_PA_10x11** 10 x 11 inches
- **PRI_PA_15x11** 15 x 11 inches
- **PRI_PA_ENV_INVITE** Envelope Invite 220 x 220 mm
- **PRI_PA_LETTER_EXTRA** Letter extra 9 \275 x 12 inches
- **PRI_PA_LEGAL_EXTRA** Legal Extra 9 \275 x 15 inches

- **PRI_PA_TABLOID_EXTRA** Tabloid Extra 11.69 x 18 inches
- **PRI_PA_A4_EXTRA** A4 Extra 9.27 x 12.69 inches
- **PRI_PA_LETTER_TRANSVERSE** Letter Transverse 8 \275 x 11 inches
- **PRI_PA_A4_TRANSVERSE A4** Transverse 210 x 297 mm
- **PRI_PA_LETTER_EXTRA_TRANSVERSE** Let. Ex. Transv. 9 \275 x 12 inches
- **PRI_PA_A_PLUS** SuperA/A4 227 x 356 mm
- **PRI_PA_B_PLUS** SuperB/A3 305 x 487 mm
- **PRI_PA_LETTER_PLUS** 8.5 x 12.69 inches
- **PRI_PA_A4_PLUS** A4 Plus 210 x 330 mm
- **PRI_PA_A5_TRANSVERSE** A5 Transverse 148 x 210 mm
- **PRI_PA_B5_TRANSVERSE** B5 (JIS) Transverse 182 x 257 mm
- **PRI_PA_A3_EXTRA** A3 Extra 322 x 445 mm
- **PRI_PA_A5_EXTRA** A5 Extra 174 x 235 mm
- **PRI_PA_B5_EXTRA B5** (ISO) Extra 201 x 276 mm
- **PRI_PA_A2** A2 420 x 594 mm
- **PRI_PA_A3_TRANSVERSE** A3 Transverse 297 x 420 mm
- **PRI_PA_A3_EXTRA_TRANSVERSE** A3 Extra Transverse 322 x 445 mm
- **PRI_PA_IW_COMPUTER** ImageWriter Computer Paper 356 x 279 mm
- **PRI_PA_IW_INTER_FANFOLD** ImageWriter International Fanfold 210 x 305 mm
- **PRI_PA_CUSTOM** Custom paper

mPaperLength specifies the length of the paper specified by mPaperSize. Length is in qpridims.

mPaperWidth specifies the width of the paper specified by mPaperSize. Width is in qpridims.

mHorzScale specifies the factor by which the printed output is scaled horizontally. The value is in percent. A value of 100 means no scaling. The value can be in the range 25 (PRI_MIN_SCALE) to 400 (PRI_MAX_SCALE).

Note: In the current implementation, horizontal scaling and vertical scaling cannot be set to different values.

mVertScale specifies the factor by which the printed output is scaled vertically. The value is in percent. A value of 100 means no scaling. The value can be in the range 25 (PRI_MIN_SCALE) to 400 (PRI_MAX_SCALE).

Note: In the current implementation, horizontal scaling and vertical scaling cannot be set to different values.

mCopies specifies the number of copies to be printed.

mDriverInfo contains the complete driver information (platform specific).

Note: You must NOT manipulate the members of this structure directly, but always use the supported functions to change this information.

```
struct PRIdrvInfo
{
  qulong mSigniture;  // printer driver type one of the PRI_DRV_xxx
  void* mData;  // printer data
  qlong mHRes;  // printers horizontal Dots per inch
  qlong mVRes;  // printers vertical Dots per inch
};
```

on Mac OSX the **mData** member has been replaced with

```
PMPageFormat mPageFormat; // page format information
PMPrintSettings mPrintSettings; // job setup information
```

PRIpageStruct

The PRIpageStruct structure is used to set properties of a page. When a new page is generated, a message is sent to the message procedure requesting this structure to be filled in.

```
struct PRIpageStruct
{
  qlong mIdent; // << unique identifier of page
  qpripage mPage; // << current page
  qprirect mGlobalBounds; // << mLocalBounds in global coordinates
  qprirect mPaperBounds;  // << paper size
  qprirect mPrintBounds;  // << printable area local to paper
  qprirect mLocalBounds;  // <<>> global arae for the page local to paper
  qprirect mHeaderBounds; // <<>> page header bounderies local to paper
  qprirect mFooterBounds; // <<>> page footer bounderies local to paper
};
```

mIdent contains the unique identifier of the page. This will usually be identical to mPage.mVert.

mPage specifies the vertical and horizontal page number.

mGlobalBounds specifies the mLocalBounds in global coordinates (local to the top and left of the mLocalBounds of the first page of the print job).

mPaperBounds specifies the paper size in qpridims. The top and left are always zero.

mPrintBounds specifies the printable area local to the paper edge.

mLocalBounds specifies the global area of the page in coordinates local to the paper edge. By default this is equal to mPrintBounds.

mHeaderBounds specifies the boundaries of the page header local to the paper edge. The default is empty (see qprirect::isEmpty()).

mFooterBounds specifies the boundaries of the page footer local to the paper edge. The default is empty (see qprirect::isEmpty()).

PRIparmStruct

The PRIparmStruct structure is used to pass information to and receive information from the print manager when calling PRIstartJob.

```
struct PRIparmStruct
{
  qulong mVersion; // << The version of the print manager
  PRIjob mJob;    // << the print job
  qapp mApp;    // >> app used for style and font tables
  PRIdestParmStruct* mDestParms; // >> the destination info (can be null)
  str255* mDocName; // >> document name
  PRIprocClass* mProc; // >> pointer to message proc instance
  qbool mHorzPages:1; // >> generate horizontal pages
  qbool mHorzHeaders:1;// >> individual horizontal headers
  qbool mHorzFooters:1;// >> individual horizontal footers
  qbool mAutoEject:1; // >> automaticcaly sends pages to device
};
```

mVersion is returned by PRIstartJob and contains the version number of the print manager. The high word contains the major version and the low word contains the minor version.

mJob is returned by PRIstartJob and contains the current print job information. It is used to pass to the numerous other print manager functions.

mApp specifies the Omnis application/library to be used for retrieving style and font information. Use ECOgetApp() to get the app for your external component.

mDestParms contains the parameters for the output devices including the destination device.

mDocName specifies the document's name. The name will appear as the title in screen report and preview windows.

mProc must specify an instance to the PRIprocClass which is to receive the various print job messages.

mHorzPages if true enables horizontal pages. Horizontal pages will be generated when a global object crosses the right boundary of the global area of a page.

mHorzHeaders if true and mHorzPages is true, the print manager will generate header messages for each horizontal page. If false, only horizontal page 1 will generate a header message. In this case header objects can cross the right boundary of the page header and extend into subsequent horizontal pages.

mHorzFooters if true and mHorzPages is true, the print manager will generate footer messages for each horizontal page. If false, only horizontal page 1 will generate a footer message. In this case footer objects can cross the right boundary of the page footer and extend into subsequent horizontal pages.

mAutoEject if true will send completed pages to the device of the given destination. This means that once a new page has been generated, no more objects can be added to the previous page. If false, all pages are buffered by the print manager until PRIejectPage is called, or the print job is closed.

PRIprocClass

The PRIprocClass class is the class which must be sub classed in order to receive print manager messages.

```
class PRIprocClass
{
  public:
    virtual qlong PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )=0;
};
```

PRIsectionStruct

The PRIsectionStruct structure specifies the properties of a new section and is used when calling PRIcreateSection.

```
struct PRIsectionStruct
{
  qlong mIdent;  // >> user identifier
  qpripos mPos;  // >> sections position
  qbool mFloatRight:1;  // >> right edge floating
  qbool mFloatBottom:1; // >> bottom edge floating
};
```

mIdent is an identifier for your use. It gives you a mechanism by which you can identify a section once it has been created.

Note: The id must be unique.

mPos specifies the sections position.

mFloatRight if true it allows the section to size its right boundary when objects added to the section cross its right boundary.

mFloatBottom if true it allows the section to size its bottom boundary when objects added to the section cross its bottom boundary.

qpridim

The qpridim is the basic coordinate unit of the print manager. One qpridim is equal to 1/1000th of a millimeter.

```
typedef qlong qpridim;
```

qprierr

The print manager error type.

```
typedef qulong qprierr;
```

qpripage

The qpripage is a simple structure combining horizontal and vertical page numbers.

```
struct qpripage
{
  qshort mHorz; // horizontal page number
  qshort mVert; // vertical page number
  // For a full declaration of the class please refer to the source file PRI.HE
};
```

qpripos

The qpripos is the main coordinate class. The structure combines qprirect and qpripage with additional members to store a section id and coordinate mode.

```
struct qpripos: public qprirect, qpripage
{
  qlong mSectID; // section id if mMOde is ePosSection
  ePRIpos mMode; // the position mode (coordinate space)
  // For a full declaration of the class please refer to the source file PRI.HE
};
```

qprirect

The qprirect is a simple structure storing the left, top, right and bottom coordinates of a rectangular area. The coordinates are in units of a qpridim.

```
struct qprirect
{
  qpridim top;
  qpridim left;
  qpridim bottom;
  qpridim right;
  // For a full declaration of the class please refer to the source file PRI.HE
};
```

Messages (Printing)

The following messages may be sent to PRIprocClass::PriProc during a print job.

PM_ADD_FOOTER_OBJECTS

This message is generated when it is time to add objects to the page footer of the current page. It is sent prior to the PM_INIT_PAGE message of the next page or the end of the print job.

Parameters:

- **pageInfo** - lParam points to a PRIpageStruct with details of the current page.

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
```

```
    case PM_ADD_HEADER_OBJECTS:
    {
      PRIpageStruct* pageInfo = (PRIpageStruct*)lParam;
      // add your footer objects to ePosFooter
      return PRI_ERR_NONE;
    }
  }
```

PM_ADD_HEADER_OBJECTS

This message is generated when it is time to add objects to the page header of the current page. It is sent immediately after the PM_INIT_PAGE.

Parameters:

- **pageInfo** - lParam points to a PRIpageStruct with details of the current page.

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_ADD_HEADER_OBJECTS:
    {
      PRIpageStruct* pageInfo = (PRIpageStruct*)lParam;
      // add your header objects to ePosHeader
      return PRI_ERR_NONE;
    }
  }
}
```

PM_CLOSE

This message is generated when the print job is being destroyed. You may now delete the PRIprocClass instance you have created for this job.

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_CLOSE:
    {
      delete this;
      return PRI_ERR_NONE;
    }
  }
}
```

PM_INIT_PAGE

This message is generated when the print manager generates a new page. It allows you to decide on the position and sizes of page headers, footers, and the global area of the page. By default page headers and footers are set to empty (no headers or footers), and the global area is initialized to the printable area of the page. Once you have set the required information for the first page of your print job, all subsequent pages will inherit these settings. You will however receive a message for each vertical and horizontal page as they are generated.

You may change the mLocalBounds (global area), mHeaderBounds and the mFooterBounds of the PRIpageStruct.

Parameters:

- **pageInfo** - lParam points to a PRIpageStruct.
- **isPaged** - wParam specifies whether we are dealing with a paged or unpaged report (mGeneratePages).

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_INIT_PAGE:
    {
      PRIpageStruct* pageInfo = (PRIpageStruct*)lParam;
      qbool isPaged = (qbool)wParam;
      if ( pageInfo->mPage.mVert == 1 && pageInfo->mPage.mHorz == 1 )
      {
        // calculate page header area
        // Note: mLocalBounds by default is set to mPrintBounds
        pageInfo->mHeaderBounds = pageInfo->mLocalBounds;
        pageInfo->mHeaderBounds.bottom = pageInfo->mHeaderBounds.top + PRI_INCH - 1;
        // subtract header from local area
        pageInfo->mLocalBounds.top += PRI_INCH;
        if ( isPaged )
        {
          // calculate page footer area
          pageInfo->mFooterBounds = pageInfo->mPrintBounds;
          pageInfo->mFooterBounds.top = pageInfo->mFooterBounds.bottom - PRI_INCH + 1;
          // subtract footer from local area
          pageInfo->mLocalBounds.bottom -= PRI_INCH;
        }
      }
      else if ( !isPaged )
      {
        // set header to empty and size local area accordingly
        pageInfo->mHeaderBounds.setEmpty();
        pageInfo->mLocalBounds = pageInfo->mPrintBounds;
      }
      return PRI_ERR_NONE;
    }
  }
}
```

PM_OUT_PAGE

This message is generated when the user clicks on the print page button of the screen report or preview window.

Parameters:

- **page** - lParam specifies the vertical page number of the current visible page.

Usually you should simply call the default print manager procedure PRIdefOutputProc, unless you have a good reason to change the default behavior of this action.

See PM_OUT_PRINTER for example.

PM_OUT_PREVIEW

This message is generated when the user clicks on the preview button of the screen report window.

Parameters:

- **page** - lParam specifies the vertical page number of the current visible page in the screen report.

Usually you should simply call the default print manager procedure PRIdefOutputProc, unless you have a good reason to change the default behavior of this action.

See PM_OUT_PRINTER for example.

PM_OUT_PRINTER

This message is generated when the user clicks on the print to printer button of the screen report or preview window.

Usually you should simply call the default print manager procedure PRIdefOutputProc, unless you have a good reason to change the default behavior of this action.

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage, WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_OUT_PRINTER:
    case PM_OUT_PAGE:
    case PM_OUT_PREVIEW:
    case PM_OUT_DISK:
    {
      return PRIdefOutputProc( pJob, NULL, pMessage, lParam, 0, 0 );
    }
  }
}
```

PM_OUT_SAVE

This message is generated when the user clicks on the save to disk button of the screen report or preview window.

Usually you should simply call the default print manager procedure PRIdefOutputProc, unless you have a good reason to change the default behavior of this action.

See PM_OUT_PRINTER for example.

PM_PAINT_OBJECT

This message is generated when an external object (PRI_OBJ_EXTERNAL) requires painting.

Parameters:

- **dc** - wParam specifies the HDC into which the object is to be painted.
- **objInfo** - lParam points to a PRIobjectStruct with details of the object.

Example:

```
qprierr myPRIprocClass::PriProc( PRIjob pJob, UINT pMessage,
WPARAM wParam, LPARAM lParam )
{
  switch ( pMessage )
  {
    case PM_PAINT_OBJECT:
    {
      HDC theHdc = (HDC)wParam;
      PRIobjectStruct* objInfo = (PRIobjectStruct*)lParam;
      // paint your custom object
      return PRI_ERR_NONE;
    }
  }
}
```

Any of the normal GDI calls can be used to paint the object.

If the external object had a font style or font table index specified when it was added, the mFnt member of the objInfo will contain the qfnt to be used for painting text.

Messages (Custom devices)

The following messages are send to the message function of a custom device. The code examples given for each message, assume the following code surrounding the message example:

```
qprierr OMNISPRIPROC DeviceFunc( PRIjob pJob, void* pOutput, qulong pData, UINT pMessage, LPARAM lParam1, LPAR
{
  myOutputClass* myOutput = (myOutputClass*)pData;
  switch ( pMessage )
  {
    // message example
  }
  // all messages not dealt with must be send on to PRIdefOutputProc
  return PRIdefOutputProc( pJob, pOutput, pMessage, lParam1, lParam2, lParam3 );
}
```

PM_OUT_ADDPAGE

This message is sent when the print job ejects a page. Your action will largely depend on the type of device you are. For example. A device which sends the output would now generate a PM_OUT_DRAWPAGE message. A screen report device would simple increment the scroll range of the display window and invalidate any relevant area. When the screen report device receives an update it then would generate the appropriate PM_OUT_DRAWPAGE message.

Further more you will need to decide if you want to draw horizontal pages as individual pages, or if you want to only draw on a vertical page basis. If you want to draw all horizontal pages as one page, you must only generate a PM_OUT_DRAWPAGE when the horizontal page number is one. You must also adjust the clipping rectangles right edge when receiving a PM_OUT_ADJPOS message so all horizontal pages are included in the paint.

Parameters:

- **page** - lParam1 points to the page number (qpripage).

Example:

```
case PM_OUT_ADDPAGE
{
  qpripage* page = (qpripage*)lParam1;
  if ( page->mHorz == 1 )
  {
```

```
    return PRIdefOutputProc( pJob, pOutput, PM_OUT_DRAWPAGE, lParam1, lParam2, lParam3 );
  }
  return PRI_ERR_NONE;
}
```

PM_OUT_ADJPOS

This message is generated prior to the PRI_OUT_DRAWOBJECT message. It gives the device a last chance to manipulate the objects position and clipping prior to drawing. If the objects position does not intersect the clipping, the PM_OUT_DRAWOBJECT message is not generated.

As mentioned for the PM_OUT_ADDPAGE message, if the device draws all horizontal pages in one PM_DRAW_PAGE message, the clippings right edge must be altered to enclose all horizontal pages.

Parameters:

- **pageInfo** - lParam1 points to the PRIpageStruct.
- **objectPos** - lParam2 points to the qpripos of the object.
- **clipRect** - lParam3 points to the clipping (qprirect)

Example:

```
case PM_OUT_ADJPOS
{
  PRIpageStruct* pageInfo = (PRIpageStruct*)lParam1;
  qpripos* objectPos = (qpripos*)lParam2;
  qprirect* clipRect = (qprirect*)lParam3;
  // first we call the PRIdefOutputProc. This will convert the objects
  // position to the printable area (ePosPrintable). It also sets the
  // clipping to the bounding rectangle of the original coordinate space
  // the object belonged to. Both positions will be local to the printable area
  // on return.
  qprierr err = PRIdefOutputProc( pJob, pOutput, PM_OUT_ADJPOS, lParam1, lParam2, lParam3 );
  if ( !err )
  {
    // now we change the width of the clipping area to enclose all
    // horizontal pages
    cliprect->width( clipRect->width() * PRI_MAX_HORZ_PAGES );
  }
  return err;
}
```

PM_OUT_AFTER

This message is generated by the destination dialog when a custom control loses the focus. The custom device should update the parameter data from the data of the control.

Parameters:

- **ctrlID** - lParam1 specifies the controls id.
- **ctrlFldval** - lParam2 specifies the qfldval with the controls data.

Example:

```
case PM_OUT_AFTER
{
  qshort ctrlID = (qshort)lParam1;
  EXTfldval ctrlFldval( (qfldval)lParam2 );
  qprierr err = myOutput->setParmValue( ctrlID, ctrlFldval );
  return err;
}
```

PM_OUT_BROWSE

This message is generated when Omnis requires the device to let the user pick a destination file, etc. Custom devices only receive this message when a job is being redirected, or the device inherits from an internal device which itself implements the PM_OUT_BROWSE message. Custom devices which derive from such internal devices can simply send the message to the PRIdefOutputProc. Otherwise they should implement their own browse function if appropriate.

Parameters:

- **title** - points to string with the title of the dialog (may be NULL).
- **mask** - points to a string specifying one or more file masks (may be NULL).

Mostly for custom devices the title and mask parameters will be NULL. You may override the default title and mask by passing your own to the super device.

Example:

```
case PM_OUT_BROWSE
{
  str255 title("Please select the destination file.")
  return PRIdefOutputProc( pJob, pOutput, PM_OUT_BROWSE, (LPARAM)&title, lParam2, lParam3 );
}
```

PM_OUT_CLICK

This message is generated by the destination dialog when a custom control has been clicked.

Parameters:

- **ctrlID** - lParam1 specifies the controls id.
- **ctrlFldval** - lParam2 specifies the qfldval with the controls data.

Example:

```
case PM_OUT_CLICK:
{
  #define PARM1_ID             1
  #define CTRL_PARM1_RADIO_0   100
  #define CTRL_PARM1_RADIO_1   101
  #define CTRL_PARM1_RADIO_2   102
  #define CTRL_BROWSE_BUTT     1000
  qshort ctrlID = (qshort)lParam1;
  EXTfldval ctrlFldval( (qfldval)lParam2 );
  switch ( ctrlID )
  {
    case CTRL_PARM1_RADIO_0:
    case CTRL_PARM1_RADIO_1:
    case CTRL_PARM1_RADIO_2:
    {
```

```
      return myOutput->setParmValue( PARM1_ID, ctrlFldval );
    }
    case CTRL_BROWSE_BUTT:
    {
      return myOutput->browseDevice();
    }
  }
  return PRI_ERR_NONE;
}
```

PM_OUT_CLOSE

The print job which owns this device is closing it. This message will usually be received after a PM_OUT_CLOSEDEVICE, and it will be followed by a PM_OUT_DESTRUCT message.

Example:

```
case PM_OUT_CLOSE
{
  myOutput->mJob = NULL; // clear the job which opened us
  return PRI_ERR_NONE;
}
```

PM_OUT_CLOSEDEVICE

Request to close the device which was opened by the PM_OUT_OPENDEVICE message.

Example:

```
case PM_OUT_CLOSEDEVICE
{
  qprierr err = myOutput->closeTheDevice();
  return PRI_ERR_NONE;
}
```

PM_OUT_CONSTRUCT

Sent when the custom device is to construct an instance of the device. The device manager will always construct one instance on registration. More instances will be constructed when print jobs require them. If a second or subsequent instance is created, the pData parameter will point to the first instance from which various settings can be inherited. When the first instance is constructed, pData will be NULL.

Parameters:

- **instPtr** - lParam1 points to a qulong pointer for storing the instance pointer of the custom device.

Example:

```
case PM_OUT_CONSTRUCT:
{
  qulong* instPtr = (qulong*)lParam1;
  *instPtr = (qulong) new myOutputClass( myOutput );
  return ( *instPtr == NULL ? PRI_ERR_MEMORY : PRI_ERR_NONE );
}
```

PM_OUT_DCLICK

This message is generated by the destination dialog when a custom control has been double clicked.

Parameters:

- **ctrlID** - lParam1 specifies the controls id.
- **ctrlFldval** - lParam2 specifies the qfldval with the controls data.

PM_OUT_DESTRUCT

Sent when the custom device is to destroy an instance of the device.

Parameters:

- **instPtr** - lParam1 points to a qulong pointer containing the instance pointer of the custom device.

Example:

```
case PM_OUT_DESTRUCT
{
  qulong* instPtr = (qulong*)lParam1;
  if ( *instPtr ) delete ((myOutputClass*)*instPtr);
  return PRI_ERR_NONE;
}
```

PM_OUT_DRAWOBJECT

This message is generated for every object of a page by the basic internal device when it receives a PM_OUT_DRAWPAGE message. Prior to generating this message, it will also generate a PM_OUT_ADJPOS message, to give the destination device a chance to do some final adjustments to the position and clipping of the object. By default the internal device will NOT generate this message if the objects position does not intersect the clipping area.

When receiving this message, the device should paint the object or send the objects data to the output, etc. Some internal devices have default implementations for this message. To use the default behavior simply pass the message to the PRIdefOutputProc.

Parameters:

- **object** - lParam1 points to the PRIobjectStruct.
- **clipRect** - lParam2 points to the clipping (qprirect).

Example:

```
case PM_OUT_DRAWOBJECT
{
  PRIobjectStruct* object = (PRIobjectStruct*)lParam1;
  qprirect* clipRect = (qprirect*)lParam2;
  return myOutput->drawObject( object, clipRect );
}
```

PM_OUT_DRAWPAGE

This message is only generated by output devices. A custom device will only receive this message if it is derived from an internal device which generates this message. In order to receive PM_OUT_DRAWOBJECT messages for all the objects of the page to be drawn, this message must be send on to PRIdefOutputProc.

Parameters:

- **page** - lParam1 points to the page number (qpripage).

Example:

```
case PM_OUT_DRAWPAGE
{
  qpripage* page = (qpripage*)lParam1;
  return PRIdefOutputProc( pJob, pOutput, PM_OUT_DRAWPAGE, lParam1, lParam2, lParam3 );
}
```

PM_OUT_FLUSHDEVICE

Request to flush the device. This is important to some devices like disk file devices where a certain amount of buffering may be occurring.

Example:

```
case PM_OUT_FLUSHDEVICE
{
  qprierr err = myOutput->flushTheDevice();
  return err;
}
```

PM_OUT_GETDLGIDS

This message must be implemented to tell the print manager whether you want to display your own parameters in the destination dialog panes, inherit the parameters of the super device, or disable the panes.

Parameters:

- **parmPane** - lParam1 points to a rstrno for returning the mode of the parameters pane of the destination dialog.
- **pageSizePane** - lParam2 points to a rstrno for returning the mode of the page size pane of the destination dialog.

The return values for both parameters can be one of the following:

PRI_DLG_NONE disable the pane.

PRI_DLG_CUSTOM display your own controls. The device must implement the PM_OUT_GETPARMDLG or PM_OUT_GETPAGEDLG messages and return a valid dialog resource.

PRI_DLG_INHERIT inherit the pane from the super device.

Example:

```
case PM_OUT_GETDLGIDS
{
  rstrno* parmPane = (rstrno*)lParam1;
  rstrno* pageSizePane = (rstrno*)lParam2;
  *parmPane = PRI_DLG_CUSTOM;
  *pageSizePane = PRI_DLG_NONE;
  return PRI_ERR_NONE;
}
```

PM_OUT_GETEOL

This message is generated by the print manager or super device if it requires the correct end of line characters for the device. This message can be sent on to the super device if the inherited EOL characters are sufficient.

Parameters:

- **eolStr**- lParam1 points to a strxxx* which is to receive the EOL characters .

Example:

```
case PM_OUT_GETEOL
{
  strxxx* eolStr = (strxxx*)lParam1;
  eolStr[0] = 2;
  eolStr[1] = 13;
  eolStr[2] = 10;
  return PRI_ERR_NONE;
}
```

PM_OUT_GETERRTEXT

This message is generated when the custom device has returned a custom error code (see PRI_ERR_CUSTOM), and the print manager requires the error text of the custom error.

Parameters:

- **errorCode** - lParam1 the custom error code.
- **errorText** - lParam2 points to the str255 which is to receive the error text.

Example:

```
case PM_OUT_GETERRTEXT
{
  #define ERR_BASE_ID 1000
  qshort errorCode = (qshort)lParam1;
  str255* errorText = (str255*)lParam2;
  RESloadString( gInstLib, ERR_BASE_ID + errorCode, *errorText );
  return PRI_ERR_NONE;
}
```

PM_OUT_GETPAGEDLG

The custom device will receive this message if it shows its own parameters on the page size pane of the destination dialog. In order to do this the custom device must have a valid dialog resource and returned PRI_DLG_CUSTOM for the page size pane when it received the PM_OUT_GETDLGIDS message.

The device must return the dialog resource to be used.

Parameters:

- **dlgFldval** - lParam1 points to the qfldval for returning the dialog resource.

Example:

```
case PM_OUT_GETPAGEDLG
{
  EXTfldval dlgFldval( (qfldval)lParam1 );
  qHandle han = RESloadDialog( gInstLib, PAGE_DIALOG_ID );
  dlgFldval.setHandle( fftBinary, han, qfalse );
  // no need to free the handle. We told setHandle to take over the memory.
}
```

PM_OUT_GETPARM

Request to return the value of the specified device parameter. It may be called from Omnis notation or the destination dialog. The custom device should register constants for its device parameters for use with the Omnis notation $getparam. See ECM_GETCONSTNAME for more details on registering constants. If the request is sent by the destination dialog, the control ID will be specified. If you wish to distinguish between calls from notation and the destination dialog, you can give the dialog control a different id to the constant name, although both may refer to the same parameter.

Parameters:

- **parmID** - lParam1 specifies the parameter number or control id of the control linked to the parameter.
- **parmFldval** - lParam2 specifies the qfldval in which to return the parameters value.
- **pickListFldval** - lParam3 specifies the qfldval for returning a pick list text string for the parameters control, if the control is a list, combo box or drop list. The string should be a comma separated list of text.

Example:

```
case PM_OUT_GETPARM
{
  qshort parmID = (qshort)lParam1;
  EXTfldval parmFldval( (qfldval)lParam2 );
  qprierr err = myOutput->getParmValue( parmID, parmFldval );
  if ( !err && lParam3 )
  {
    EXTfldval pickListFldval( (qfldval)lParam3 );
    err = myOutput->getParmPickList( parmID, pickListFldval );
  }
  return err;
}
```

When this message is sent for the controls of your custom dialog pane, for any control which is not displaying parameter data you must return PRI_ERR_IGNORE. Otherwise the print manager will assume that you have specified data for the control. For example, your dialog resource may contain a push button for browsing which has received its text from the resource. If you do NOT return PRI_ERR_IGNORE, the button text will be replaced with the data in parmFldval.

PM_OUT_GETPARMDLG

The custom device will receive this message if it shows its own parameters on the parameters pane of the destination dialog. In order to do this the custom device must have a valid dialog resource and returned PRI_DLG_CUSTOM for the parameters pane when it received the PM_OUT_GETDLGIDS message.

The device must return the dialog resource to be used.

Parameters:

- **dlgFldval** - lParam1 points to the qfldval for returning the dialog resource.

Example:

```
case PM_OUT_GETPARMDLG
{
  EXTfldval dlgFldval( (qfldval)lParam1 );
  qHandle han = RESloadDialog( gInstLib, PARM_DIALOG_ID );
  dlgFldval.setHandle( fftBinary, han, qfalse );
  // no need to free the handle. We told setHandle to take over the memory.
}
```

PM_OUT_ISDEVICEOPEN

Return the open status of the device.

Parameters:

- **isOpen** - lParam1 points to a qbool for the return value.

Example:

```
case PM_OUT_ISDEVICEOPEN
{
  qbool* isOpen = (qbool*)lParam1;
  *isOpen = myOutput->isTheDeviceOpen();
  return PRI_ERR_NONE;
}
```

PM_OUT_KILL

This message can be sent to the PRIdefOutputProc, to tell the device manager to close the device, and destroy the device instance if it was instanciated by a print job.

PM_OUT_LOADPARMS

This message is sent when you are required to load your parameter class data from an Omnis data collection (CRB). This message is generated when:

- one or more parameters are being set or requested by the notation $getparam and $setparam.
- the device is about to be opened by the notation $open or by a print job.
- the destination dialog is about to display your custom pane.
- the parameters require validation (A PM_OUT_VALIDATEPARMS and PM_OUT_SAVEPARMS message will follow).
- the Destination dialog requires the device to browse (a PM_OUT_BROWSE message will follow).

Parameters:

- **parms** - lParam1 points to the Omnis data collection.

Example:

```
case PM_OUT_LOADPARMS
{
  #define CUR_VERSION 1
  #define PARM_VERSION_XN 1
  #define PARM_FIRST_XN 2
  #define PARM_LAST_XN 5
  EXTcrb crb( (qcrb)lParam1 );
  if ( crb.getLong( PARM_VERSION_XN ) == CUR_VERSION )
  {
    for ( qcrbindex parm = PARM_FIRST_XN ; parm <= PARM_LAST_XN ; parm++ )
    {
      myOutput->setParm( parm, crb.getDataRef( parm ) );
    }
  }
  else
  {
```

```
    // set parameters to default values
  }
  return PRI_ERR_NONE;
}
```

Note: It is always good practice to store a version number together with your parameters in the data collection. The data collection will be stored in the Omnis config file between sessions.

PM_OUT_OPEN

Sent when a print job has instanciated the device. This message will usually be received after a PM_OUT_CONSTRUCT message. And it will be followed by a PM_OUT_OPENDEVICE message. The device should remember that it was opened by a print job, and NOT allow calls to PM_OUT_SENDTEXT and PM_OUT_SENDDATA.

Parameters:

- **destParms** - lParam1 points to the destination parameters.

Example:

```
case PM_OUT_OPEN
{
  PRIdestParmStruct* destParms = (PRIdestParmStruct*)lParam1;
  myOutput->mJob = pJob; // remember the job which opened us
  return PRI_ERR_NONE;
}
```

PM_OUT_OPENDEVICE

Request to open the actual device. For different output devices it will mean different things. A device that writes to a file or port should open the file or port. A device that prints to a window on screen should open this window.

Parameters:

- **destParms** - lParam1 points to the destination parameters.

Example:

```
case PM_OUT_OPENDEVICE
{
  PRIdestParmStruct* destParms = (PRIdestParmStruct*)lParam1;
  qprierr err = myOutput->openTheDevice( destParms );
  return err;
}
```

PM_OUT_SAVEPARMS

This message is sent when you are required to store you parameter class data in an Omnis data collection (CRB).

Parameters:

- **parms** - lParam1 points to the Omnis data collection.

Example:

```
case PM_OUT_SAVEPARMS
{
  #define CUR_VERSION 1
  #define PARM_VERSION_XN 1
  #define PARM_FIRST_XN 2
  #define PARM_LAST_XN 5
  EXTcrb crb( (qcrb)lParam1 );
  // store version number with parameters
  crb.setLong( PARM_VERSION_XN, CUR_VERSION );
  for ( qcrbindex parm = PARM_FIRST_XN ; parm <= PARM_LAST_XN ; parm++ )
  {
    // call getDataRef with qtrue as the second parameter
    // this tells the collection that we are altering the data
    myOutput->getParm( parm, crb.getDataRef( parm, qtrue )
;
  }
  return PRI_ERR_NONE;
}
```

Note: It is always good practice to store a version number together with your parameters in the data collection. The data collection will be stored in the Omnis config file between sessions.

PM_OUT_SENDDATA

Send the given data to the device. This may be binary data of any sort. It is left to the developer or user whether the data is compatible with the device.

Parameters:

- **dataPtr** - lParam1 points to the data.
- **dataLen** - lParam2 specifies the data length in bytes.

Example:

```
case PM_OUT_SENDDATA
{
  qbyte* dataPtr = (qchar*)lParam1;
  qlong dataLen = (qlong)lParam2;
  qprierr err = PRI_ERR_NONE;
  if ( dataLen )
  {
    err = myOutput->sendDataToDevice( dataPtr, dataLen );
  }
  return err;
}
```

PM_OUT_SENDPAGE

This message is generated by the PRI_DEST_EXTTEXT and all other internal devices which derive from this device. The PRI_DEST_EXTTEXT device buffers the text of a complete page into one block of data. The text is positioned correctly by inserting spaces or extra return characters where there are gaps between objects. Once a page is complete, it generates this message so the text data can be written to the device.

Parameters:

- **pageInfo** - lParam1 points to the PRIpageStruct.
- **textFldval** - lParam2 points to the qfldval with the text.

Example:

```
case PM_OUT_SENDPAGE
{
  PRIpageStruct* pageInfo = (PRIpageStruct*)lParam1;
  EXTfldval textFldval( (qfldval)lParam2 );
  return myOutput->sendPage( pageInfo, textFldval );
}
```

PM_OUT_SENDTEXT

Send the given text to the device. All normal character conversion has already been applied to the text.

Parameters:

- **textPtr** - lParam1 points to the text.
- **textLen** - lParam2 specifies the text length.
- **newLine** - HIWORD(lParam3) specifies if a line feed is to occur.
- **formFeed** - LOWORD(lParam3) specifies if a form feed is to occur.

Example:

```
case PM_OUT_SENDTEXT
{
  qchar* textPtr = (qchar*)lParam1;
  qlong textLen = (qlong)lParam2;
  qbool newLine = (qbool)HIWORD(lParam3);
  qbool formFeed = (qbool)LOWORD(lParam3);
  qprierr err = PRI_ERR_NONE;
  if ( textLen )
  {
    err = myOutput->sendTextToDevice( textPtr, textLen );
  }
  if ( newLine && !err
  {
    err = myOutput->sendLineFeedToDevice();
  }
  if ( formFeed && !err )
  {
    err = myOutput->sendFormFeedToDevice();
  }
  return err;
}
```

PM_OUT_SETDATA

This message can be sent to the PRIdefOutputProc by the custom device to change the data of a custom control on the destination dialog.

Parameters:

- **ctrlID** - lParam1 specifies the controls id.
- **ctrlFldval** - lParam2 specifies the qfldval with the controls data.

Example:

```
qshort    ctrlID = 1
EXTfldval  ctrlFldval;
ctrlFldval.setLong(1);
return PRIdefOutputProc( pJob, pOutput, PM_OUT_SETDATA, (LPARAM)ctrlID, (LPARAM)ctrlFldval.getFldval(), 0 );
```

PM_OUT_SETPARM

This message is sent only by the notation $setparam. You should validate the data and return the error PRI_ERR_INVALID_PARM if the parameter is not valid.

Parameters:

- **parmID** - lParam1 specifies the parameter number.
- **parmFldval** - lParam2 specifies the qfldval specifying the new value for the parameter.

Example:

```
case PM_OUT_SETPARM
{
  qshort parmID = (qshort)lParam1;
  EXTfldval parmFldval( (qfldval)lParam2 );
  qprierr err = myOutput->setParmValue( parmID, parmFldval );
  return err;
}
```

PM_OUT_SET_HDC

This message can be sent to the PRI_DEST_EXTHDC super device of a custom device to set the drawing DC prior to calling the super device to draw.

Parameters:

- **theHDC**- lParam1 specifies the HDC for drawing.

Example:

```
// Our components WndProc has received a WM_PAINT message for our visual output device.
// first calculate the page we need to draw based on scroll position
// for simplicity of this example we assume it is page 1.
qpripage pg( 1, 1 );
qprierr err = PRI_ERR_NONE;
WNDpaintStruct paintStruct;
WNDbeginPaint( hwnd, &paintStruct );
  err = PRIdefOutputProc( pJob, pOutput, PM_OUT_SET_HDC, (LPARAM)paintStruct.hdc, 0, 0 );
  if ( !err ) err = PRIdefOutputProc( pJob, pOutput, PM_OUT_DRAWPAGE, (LPARAM)&pg, 0, 0 );
WNDendPaint( hwnd, &paintStruct );
```

PM_OUT_VALIDATEPARMS

This message is sent when you are required to validate the device parameters. You would have been sent a PM_OUT_LOADPARMS message prior to this call, so your class members which store the parameters should contain the correct values for validation. You should consider whether you need to prompt the user for one or more of your parameters if they are empty or invalid. For example, the internal File device and the custom Html device both put up a put file dialog if their file name parameter is empty.

You must also consider if you need to change any of the members of the PRIdestParmStruct which lParam1 points to. For example, the custom Html device always assigns qfalse to mGeneratePages. It does not support paged reports.

Parameters:

- **destParms** - lParam1 points to the PRIdestParmStruct.

Example:

```
case PM_OUT_VALIDATEPARMS
{
  PRIdestParmStruct* destParms = (PRIdestParmStruct*)lParam1;
  qprierr err = myOutput->validateParms( destParms );
  return err;
}
```

PM_OUT_ZOOM

This message can be sent to the PRI_DEST_EXTPREVIEW super device to toggle the zoom state on or off.

Parameters:

- **zoomOn** - lParam1 specifies if the zoom is to be turned on or off.

Example:

```
// turn zoom on
qprierr err = PRIdefOutputProc( pJob, pOutput, PM_OUT_ZOOM, 1, 0, 0 );
// turn zoom off
qprierr err = PRIdefOutputProc( pJob, pOutput, PM_OUT_ZOOM, 0, 0, 0 );
```

Functions

PRIaddObject

---

qprierr PRIaddObject(PRIjob pJob, PRIobjectStruct* pObject)

Adds an object to the specified print job.

Parameters:

- **pJob** - points to the print job.
- **pObject** - points to the PRIobjectStruct specifying the properties for the object. If the object has been moved or sized by the formatting manager, the mPos member will contain the updated position when the function returns.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIobjectStruct obj(qnil);
obj.mType = PRI_OBJ_RECT;
obj.mBorder.set( WND_BORD_BEVEL, 3, 3, 3 );
obj.mFillPat = patFill;
obj.mForeFillColor = GDI_COLOR_3DFACE;
obj.mScreenUnits = obj.mFloatBottom = qtrue;
qprierr err = PRIaddObject( theJob, &obj );
```

See also PRIcreateSection

PRIbrowseOutput

qbool PRIbrowseOutput(PRIdestParmStruct* pDestParms, strxxx* pTitle = NULL, strxxx* pMask = NULL)

Generates PM_OUT_BROWSE message for the output device specified by pDestParms->mDest. Only some devices like PRI_DEST_FILE support this message and will open a put file dialog.

Parameters:

- **pDestParms** - The destination parameters. You can use the return value of ECOgetDeviceParms() for this parameter to specify the Omnis global device parameters.
- **pTitle** - if specified will display the text as the title of the put file dialog.
- **pMask** - is specified will use the given mask(s) in the put file dialog.
- **return** - returns qtrue if the user has clicked ok.

Example:

```
qbool ok = PRIbrowseOutput( ECOgetDeviceParms( eci->mInstLocp ), NULL, NULL );
```

See also PRIopenDestinationDialog

PRIbuildPrinterList (Windows only)

qlong PRIbuildPrinterList(qfldval pList, qbool pShowPort)

Builds a list of installed printers and returns the row number of the current printer in the list.

Parameters:

- **pList** - destination qfldval which is to receive the list of printers.
- **pShowPort** - if true the port name through which the printer is installed is shown as part of the printers name.

Example:

```
EXTfldval fval; EXTqlist* lst;
qlong curPrinter = PRIbuildPrinterList( fval.getFldVal(), qfalse );
lst = fval.getList( qfalse );
// do something with the list
delete lst;
```

See also PRIopenChangePrinterDialog

PRIchangeJobPageSetup

qprierr PRIchangeJobPageSetup(PRIjob pJob, PRIpageSetup* pPageSetup, PRIpageStruct* pPageInfo)

This function must be called during the PM_INIT_PAGE message. It will change the page setup information for the current page and all subsequent pages. It allows you to print to different paper sizes or orientations or scaling etc, within the same print job. On return, pPageInfo will have been updated to reflect the new page setup information.

Parameters:

- **pJob** - specifies the active job.
- **pPageSetup** - specifies the new page setup information.
- **pPageInfo** - points to the page information of the current page. This information will be updated prior to the function returning.

Example:

```
case PM_INIT_PAGE:
{
  PRIpageStruct* pgInfo = (PRIpageStruct*)lParam1;
  if ( pgInfo->mPage.mVert == 2 )
  {
    // change page 2 onwards to landscape
    PRIpageSetup* pgSetup = NULL;
    qprierr err = PRIgetPageSetup( pJob, pgSetup );
    if ( !err )
    {
      if ( PRIsetPageSetupItem(pgSetup,PRI_PS_ORIENT,PRI_OR_LANDSCAPE) )
      {
        err = PRIchangeJobPageSetup( pJob, pgSetup, pgInfo );
      }
      PRIdestroyPageSetup( pgSetup );
    }
  }
  // initialize the page info
  return PRI_ERR_NONE;
}
```

See also PRIgetPageSetup, PRIsetPageSetup, PRIcopyPageSetup, PRIdestroyPageSetup

PRIclose

qprierr PRIclose()

Closes the print manager and destroys all print manager data, if the user count decrements to zero. See PRIopen for a full description on its usage.

Parameters:

- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIopen();
if ( !err )
{
  // use the print manager
  PRIclose();
}
```

See also PRIopen

PRIcloseDevice

qprierr PRIcloseDevice(qlong pDest)

You call PRIcloseDevice to close a device which you previously opened by calling PRIopenDevice.

Parameters:

- **pDest** - id of the device to be closed.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIopenDevice( PRI_DEST_PRINTER );
if ( !err )
{
  // send some text or data to device
  err = PRIcloseDevice( PRI_DEST_PRINTER );
}
```

See also PRIopenDevice, PRIsendTextToDevice, PRIsendDataToDevice, PRIflushDevice, PRIisDeviceOpen

PRIconvFromCM

qpridim PRIconvFromCM(qlong pCms, qlong pFraction)

Converts centimeters to qpridims.

Parameters:

- **pCms** - specifies the number of centimeters or fractions thereof.
- **pFraction** - specifies the fraction size of pCms. A value of 1 means that pCms specifies complete centimeters. A value of 1000 means that pCms specifies 1000th of a centimeter.
- **return** - returns the number of qpridims.

Example:

```
qpridim centiMeter = PRIconvFromCM( 1, 1 );
qpridim milliMeter = PRIconvFromCM( 1, 10 );
// centiMeter will be 10000 (1 cm), milliMeter will be 1000 (1 mm)
```

See also PRIconvToCM

PRIconvFromCMorINCH

void PRIconvFromCMorINCH(const qreal& pReal, qpridim& pPridim)

Converts the given value to qpridims. The given value is assumed to be centimeters if the Omnis preference $usecms is true. Otherwise the given value is assumed to be in inches.

Parameters:

- **pReal** - specifies the centimeters or inches.

- **pPridim** - the qpridims are returned in this parameter.

Example:

```
qpridim theResult;
qreal theValue = 1;
PRIconvFromCMorINCH( theValue, &theResult );
// if $usecms is true, theResult will be 10000 (1cm)
// if $usecms is false, theResult will be 25400 (1 inch)
```

See also PRIconvToCMorINCH

PRIconvFromDC

qpridim PRIconvFromDC(HDC pHdc, qdim pUnits, qbool pVert)

Converts device units of the given device to qpridims.

Parameters:

- **pHdc** - specifies the device
- **pUnits** - specifies the number of device units
- **pVert** - specifies if the given units are vertical or horizontal units
- **return** - returns the number of qpridims

Example:

```
qrect theDeviceRect( 0, 0, 15, 15 );
qpridim thePriWidth = PRIconvFromDC( theDC, theDeviceRect.width(), qfalse );
```

See also PRIconvToDC

PRIconvFromIN

qpridim PRIconvFromIN(qlong pInches, qlong pFraction)

Converts inches to qpridims.

Parameters:

- **pInches** - specifies the number of inches or fractions thereof.
- **pFraction** - specifies the fraction size of pInches. A value of 1 means that pInches specifies inches. A value of 1000 means that pInches specifies 1000th of an inch.
- **return** - returns the number of qpridims.

Example:

```
qlong theInches = 500;
qlong inFractionsOf = 1000;
qpridim theResult = PRIconvFromIN( theInches, inFractionsOf );
// theResult will be 12700 (1/2 inch)
```

See also PRIconvToIN

PRIconvFromScreen

qpridim PRIconvFromScreen(qdim pUnits, qbool pVert)

Converts screen units to qpridims.

Parameters:

- **pUnits** - specifies the number of screen units.
- **pVert** - specifies if the given units are vertical or horizontal units.
- **return** - returns the number of qpridims.

Example:

```
qrect theScreenRect( 0, 0, 15, 15 );
qpridim thePriWidth = PRIconvFromScreen( theScreenRect.width(), qfalse );
```

See also PRIconvToScreen

PRIconvPos

qprierr PRIvonvPos(PRIjob pJob, qpripos* pPos, ePRIpos pTo)

Converts the given report position from its current coordinate space to the coordinate space specified by pTo.

Parameters:

- **pJob** - specifies the job to be used in the conversion.
- **pPos** - points to the qpripos to be converted.
- **pTo** - specifies the new coordinate space for pPos. If pTo specifies ePosSection, the mIdent member of the pPos must specify the section id.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
Qprirect theRect( PRI_INCH, PRI_INCH, PRI_INCH * 3, PRI_MM * 5 );
qpripos thePos( ePosPaper, 1, 5, theRect ); // hpage 1, vpage 5
qprierr err = PRIconvPos( theJob, &thePos, ePosPrintable );
```

See also qpripos, ePRIpos

PRIconvRectFromDC

void PRIconvRectFromDC(HDC pHdc, qrect* pDeviceRect, qprirect* pPriRect)

Converts a device rectangle to a qprirect.

Parameters:

- **pHdc** - specifies the device.

- **pDeviceRect** - points to the device rectangle.
- **pPriRect** - points to the qprirect to receive the converted coordinates.

Example:

```
qrect theDeviceRect( 0, 0, 15, 15 );
qprirect thePriRect;
PRIconvRectFromDC( theDC, &theDeviceRect, &thePriRect );
```

See also PRIconvRectToDC

PRIconvRectFromScreen

---

void PRIconvRectFromScreen(qrect* pScreenRect, qprirect* pPriRect)

Converts a screen rectangle to a qprirect.

Parameters:

- **pScreenRect** - points to the screen rectangle.
- **pPriRect** - points to the qprirect to receive the converted coordinates.

Example:

```
qrect theScreenRect( 0, 0, 15, 15 );
qprirect thePriRect;
PRIconvRectFromScreen( &theScreenRect, &thePriRect );
```

See also PRIconvRectToScreen

PRIconvRectToDC

---

void PRIconvRectToDC(HDC pHdc, qprirect* pPriRect, qrect* pDeviceRect)

Converts a qprirect to a device rectangle.

Parameters:

- **pHdc** - specifies the device.
- **pPriRect** - points to the qprirect.
- **pDeviceRect** - points to the qrect to receive the converted coordinates.

Example:

```
Qprirect thePriRect( 0, 0, PRI_INCH*3, PRI_MM*5 );
qrect theDeviceRect;
PRIconvRectToDC( theDC, &thePriRect, &theDeviceRect );
```

See also PRIconvRectFromDC

PRIconvRectToScreen

void PRIconvRectToScreen(qprirect* pPriRect, qrect* pScreenRect)

Converts a qprirect to a screen rect.

Parameters:

- **pPriRect** - points to the qprirect.
- **pScreenRect** - points to the qrect to receive the converted coordinates.

Example:

```
Qprirect thePriRect( 0, 0, PRI_INCH*3, PRI_MM*5 );
qrect theScreenRect;
PRIconvRectToScreen( &thePriRect, &theDeviceRect );
```

See also PRIconvRectFromScreen

PRIconvToCM

qlong PRIconvToCM(qpridim pUnits, qlong pRetFraction)

Converts qpridims to centimeters or fractions thereof.

Parameters:

- **pUnits** - specifies the number of qpridims to convert.
- **pRetFraction** - specifies the fraction size to be returned. A value of 1 means return centimeters. A value of 1000 means return 1000th of a centimeter.
- **return** - returns centimeters or fractions thereof.

Example:

```
qpridim thePriDims = 100000; // = 10cms
qlong theResult = PRIconvToCM( thePriDims, 10 );
// the result will be 100
```

See also PRIconvFromCM

PRIconvToCMorINCH

void PRIconvToCMorINCH(qpridim pPridim, qreal& pReal)

Converts the given qpridims to centimeters or inches. If the Omnis preference $usecms is true, the qpridims are converted to centimeters, otherwise they are converted to inches.

Parameters:

- **pPridim** - specifies the qpridims to be converted.
- **pReal** - the centimeters or inches are returned in this parameter.

Example:

```
qpridim thePriDims = 100000; // 10cms
qreal theResult;
PRIconvToCMorINCH( thePriDims, theResult );
// if $usecms is true, theResult will be 10
// if $usecms is false, theResult will be 3.93700
```

See also PRIconvFromCMorINCH

PRIconvToDC

qdim PRIconvToDC(HDC pHdc, qpridim pUnits, qbool pVert)

Converts qpridims to device units of the given device.

Parameters:

- **pHdc** - specifies the device.
- **pUnits** - specifies the number of qpridims.
- **pVert** - specifies if the given units are vertical or horizontal units.
- **return** - returns the number of device units.

Example:

```
qpridim thePriDims = 25400; // 1 inch
qdim theResult = PRIconvToDC( theDC, thePriDims, qfalse );
// if the DC is a 600 dpi printer dc, theResult will be 600
```

See also PRIconvFromDC

PRIconvToIN

qlong PRIconvToIN(qpridim pUnits, qlong pRetFraction)

Converts qpridims to inches or fractions thereof.

Parameters:

- **pUnits** - specifies the number of qpridims to convert.
- **pRetFraction** - specifies the fraction size to be returned. A value of 1 means return inches. A value of 1000 means return 1000th of an inch.
- **return** - returns inches or fractions thereof.

Example:

```
qpridim thePriDims = 25400; // 1 inch
qlong theResult = PRIconvToIN( thePriDims, 1000 );
// theResult will be 1000 ( in 1000th of an inch )
```

See also PRIconvFromIN

PRIconvToScreen

qdim PRIconvToScreen(qpridim pUnits, qbool pVert)

Converts qpridims to screen units.

Parameters:

- **pUnits** - specifies the number of qpridims.
- **pVert** - specifies if the given units are vertical or horizontal units.
- **return** - returns screen units.

Example:

```
qpridim thePriDims = 25400; // 1 inch
qdim theResult = PRIconvToScreen( thePriDims, qtrue );
// if the vertical screen units are 96 dpi, theResult will be 96
```

See also PRIconvFromScreen

PRIcopyPageSetup

qprierr PRIcopyPageSetup(PRIpageSetup* pSrcPageSetup, PRIpageSetup*& pDestPageSetup)

Creates a copy of the page setup information of the given page setup.

IMPORTANT: PRIcopyPageSetup allocates memory for the page setup information. PRIdestroyPageSetup must be called when the page setup information is no longer required.

Parameters:

- **pSrcPageSetup** - points to the source page setup information which is to be copied.
- **pDestPageSetup** - reference of a page setup pointer which is to receive the pointer to the page setup copy.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageSetup* pageSetup, copyOfPageSetup;
qprierr err = PRIgetPageSetup( NULL, pageSetup );
if ( !err )
{
  err = PRIcopyPageSetup( pageSetup, copyOfPageSetup );
  if ( !err )
  {
    // delete once finished with
    PRIdestroyPageSetup( copyOfPageSetup );
  }
  // delete once finished with
  PRIdestroyPageSetup( pageSetup );
}
```

See also PRIgetPageSetup, PRIsetPageSetup, PRIdestroyPageSetup, PRIopenPageSetupDialog, PRIchangeJobPageSetup

PRIcreateSection

qprierr PRIcreateSection(PRIjob pJob, PRIsectionStruct* pSection)

Creates a new section. Objects can be added local to the section by specifying ePosSection as the position mode of the object. The section id must be specified in the objects position. The coordinates of objects added to a section, will be local to the sections position. Once you have finished with a section, you must call PRIdeleteSection to free the memory occupied by the section. The section is only used while adding objects to the section. Once all objects have been added to a section, the section is no longer required.

If a section is resized and you require the new position of the section, you can call PRIgetSectionInfo prior to deleting the section.

When a section is deleted, all objects belonging to the section will be floated if their floating properties are enabled, and their position is normalized, in other words they are converted to the coordinate space of the section, rather than being local to the section.

Parameters:

- **pJob** - points to the print job.
- **pSection** - points to the PRIsectionStruct specifying the properties for the new section.

return - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIsectionStruct secInfo(qnil);
secInfo.mIdent = 1001;
secInfo.mPos = qpripos(PRI_INCH,PRI_INCH,PRI_INCH*3,PRI_INCH*2);
secInfo.mFloatBottom = qtrue;
qprierr err = PRIcreateSection( theJob, &secInfo );
if ( !err )
{
  // add objects to section
  err = PRIdeleteSection( theJob, 1001 );
}
```

See also PRIaddObject, PRIdeleteSection, PRIgetSectionInfo, PRIsetSectionInfo, PRIgrowSection

PRIdefOutputProc

qprierr PRIdefOutputProc(PRIjob pJob, void* pOut, UINT pMessage, LPARAM lParam1, LPARAM lParam2, LPARAM lParam3)

PRIdeleteSection

qprierr PRIdeleteSection(PRIjob, pJob, qlong pSectionID)

Deletes the specified section and frees the memory it occupies. For a full description of using sections see PRIcreateSection.

Parameters:

- **pJob** - points to the print job.
- **pSectionID** - specifies the ID of the section to be deleted.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See PRIcreateSection

See also PRIcreateSection, PRIgetSectionInfo, PRIsetSectionInfo, PRIgrowSection

PRIdestroyPageSetup

qprierr PRIdestroyPageSetup(PRIpageSetup* pPageSetup)

Frees the memory occupied by the given page setup information.

WARNING: Any further reference to the page setup information will result in a crash.

Parameters:

- **pPageSetup** - points to the page setup information to be destroyed.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See PRIcopyPageSetup

See also PRIgetPageSetup, PRIsetPageSetup, PRIcopyPageSetup, PRIopenPageSetupDialog, PRIchangeJobPageSetup

PRIdisposeCustomProc

void PRIdisposeCustomProc(FARPROC pProc)

This function should be called to free the memory occupied by the FARPROC which was created when calling PRImakeCustomProc. This function must only be called when the custom message function is no longer required by the output manager (typically after the custom device has been unregistered).

Example:

See section 'A simple external output device'

See also PRImakeCustomProc, PRIregisterOutput, PRIunregisterOutput

PRIejectPage

qprierr PRIejectPage(PRIjob pJob, qlong pPage)

Ejects the specified page and all pages prior to pPage which have not been ejected. The page or pages are sent to the output device of the print job.

Once a page has been ejected, no more objects can be added to that page.

Note: PRIendpage is called for every page for which it hasn't been called already. PRIejectPage only needs to be called if automatic page ejection is disabled.

Parameters:

- **pJob** - points to the print job.
- **pPage** - specifies the page to be ejected. If it is zero, all pages are ejected.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIejectPage( theJob, 0 );
// will eject all pages
```

See also PRIparmStruct.mAutoEject

PRIendJob

qprierr PRIendJob(PRIjob pJob)

Closes the given print job, and once no more output devices are active, the print job will be destroyed. PRIendJob always calls PRIendPage and PRIejectPage to eject all pages before closing the print job.

Parameters:

- **pJob** - points to the print job to be closed. WARNING: Any further reference to this print job after calling this function may result in a crash.
- **return** - returns one of the PRI_ERR_xxx error constants. If a fatal error occurred while printing, this error will be returned by PRIendJob.

Example:

```
qprierr err = PRIendJob( theJob );
```

See also PRIstartJob, PRIkillJob

PRIendPage

qprierr PRIendPage(PRIjob pJob)

Closes the current page and ejects it if auto ejection is enabled.

The format manager will do the following:

send a print footer message for the current page.

check if any of the pages objects have crossed the page boundary. If one or more have, a new page is generated, and both print header and footer messages are sent for this page.

Parameters:

- **pJob** - points to the print job.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIendPage( theJob );
```

See also PRIstartPage, PRIgetPageInfo, PRIsetPageInfo

PRIextended

qbool PRIextended()

On the MAC OS it returns true if extended printing is available LaserWriter 8.4.1 or later.

On the Windows platform it always returns true.

PRIflattenDriverInfo

qprierr PRIflattenDriverInfo(PRIpageSetup* pPageSetup, qfldval pData)

Flattens the driver info specified by pPageSetup so it is suitable for storing on disk.

Parameters:

- **pPageSetup** - points to the page setup structure which contains the driver info.
- **pData** - qfldval which is to receive the data.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageSetup* pageSetup;
qprierr err = PRIgetPageSetup( NULL, pageSetup );
if ( !err )
{
  EXTfldval fval;
  err = PRIflattenDriverInfo( pageSetup, fval.getFldVal() );
  if ( !err )
  {
    // store driver info
  }
  PRIdestroyPageSetup( pageSetup );
}
```

See also PRIgetDriverSigniture, PRIgetFlattDriverInfoSize, PRIunflattenDriverInfo

PRIflushDevice

qprierr PRIflushDevice(qlong pDest)

Some devices may buffer the text or data transmitted to them. To make sure it has been sent, this function can be called.

Parameters:

- **pDest** - specifies the device.

Example:

qprierr err = PRIflushDevice(PRI_DEST_FILE);

See also PRIopenDevice, PRIcloseDevice, PRIsendTextToDevice, PRIsendDataToDevice, PRIisDeviceOpen

PRIgetDeviceEOLchars

qprierr PRIgetDeviceEOLchars(qlong pDest, str15* pEOL)

Returns the end of line characters for the specified device. Different devices may encode an end of line with a different set of characters.

Parameters:

- **pDest** - specifies the device.

- **pEOL** - points to the str15 which is to receive the EOL chars.

Example:

```
str15 eol;
qprierr err = PRIgetDeviceEOLchars( PRI_DEST_FILE, &eol );
```

PRIgetDeviceInfo

qprierr PRIgetDeviceInfo(qlong pDest, PRIdeviceInfoStruct* pInfo)

Fetches information about the specified device.

Parameters:

- **pDest** - specifies the device.
- **pInfo** - point to the PRIdeviceInfoStruct which is to receive the device's information.

Example:

```
PRIdeviceInfoStruct devInfo;
qprierr err = PRIgetDeviceInfo( PRI_DEST_PRINTER, &devInfo );
```

See also PRIdeviceInfoStruct, PRIsetDeviceInfo, PRIgetDeviceName

PRIgetDeviceName

qprierr PRIgetDeviceName(qlong pDest, str255* pName)

Returns the internal name of the specified device. The internal name is the name with which the device registered itself. The internal name can NOT be changed and must be unique.

Parameters:

- **pDest** - specifies the device.
- **pName** - points to the str255 which is to receive the device name.

Example:

```
qlong findHtml()
{
  str255 html( "Html" );
  str255 theName;
  for ( qlong cnt = PRI_DEST_CUSTOM_FST ; cnt <= PRI_DEST_CUSTOM_LST ; cnt++ )
  {
    qprierr err = PRIgetDeviceName( cnt, &theName );
    if ( !err && theName == html ) return cnt;
  }
  return 0;
}
```

See also PRIgetDeviceInfo, PRIsetDeviceInfo

PRIgetDriverSignature

qprierr PRIgetDriverSignature(PRIpageSetup* pPageSetup, qlong* pSignature)

Returns the signature of the driver info in pPageSetup. If pPageSetup is NULL, the print manager returns the signature for the driver info which is compatible with the current operating system.

When storing flattened driver information with a document on disk, you should also store the driver signature. Your document should be able to store multiple driver information (one for each signature).

Parameters:

- **pPageSetup** - points to the page setup for which to return the driver signature. If it is NULL the print manager returns the expected driver signature.
- **pSignature** - points to the qlong in which the signature is returned. It will be one of the following.
  PRI_DRV_MAC MAC classic printing
  PRI_DRV_MACE MAC classic printing with extended print record.
  PRI_DRV_CARBON MAC OSX printing
  PRI_DRV_WIN16 WIN 16 platform
  PRI_DRV_WIN32 WIN 32 platform
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageSetup* pageSetup;
qprierr err = PRIgetPageSetup( NULL, pageSetup );
if ( !err )
{
  qlong signature;
  err = PRIgetDriverSignature( pageSetup, &signature );
  if ( !err && signature == PRI_DRV_MACE )
  {
    // do something
  }
  PRIdestroyPageSetup( pageSetup );
}
```

See also PRIgetFlattDriverInfoSize, PRIflattenDriverInfo, PRIunflattenDriverInfo

PRIgetError

qprierr PRIgetError(PRIjob pJob)

Returns the last fatal error for the given print job.

Parameters:

- **pJob** - points to the print job.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIgetError( theJob );
if ( err ) PRIshowError( err );
```

See also PRIsetError, PRIshowError, PRIgetSysError, PRIgetErrorText

PRIgetErrorText

qret PRIgetErrorText(qprierr pErr, str255* pText)

Returns the error text and standard Omnis error code of the given print manager error.

Parameters:

- **pErr** - specifies the print manager error code.
- **pText** - points to the str255 which is to receive the error text.
- **return** - returns a standard Omnis error code.

Example:

```
qprierr err = PRIgetError( theJob );
if ( err )
{
  str255 errText;
  qret e = PRIgetErrorText( err, &errText );
}
```

See also PRIgetError, PRIsetError, PRIshowError, PRIgetSysError

PRIgetFlattDriverInfoSize

qprierr PRIgetFlattDriverInfoSize(PRIpageSetup* pPageSetup, qlong* pSize)

Returns the size of the driver info stored in pPageSetup once it is converted to a flat format which is suitable for storing on disk. This function should be called before calling PRIflattenDriverInfo so a buffer of the correct size can be allocated.

Parameters:

- **pPageSetup** - points to the page setup structure which contains the driver info.
- **pSize** - points to the qlong which receives the required buffer size.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See PRIflattenDriverInfo

See also PRIgetDriverSigniture, PRIflattenDriverInfo, PRIunflattenDriverInfo

PRIgetPageInfo

qprierr PRIgetPageInfo(PRIjob pJob, PRIpageStruct* pPage)

Fetches information about the specified or current page.

Parameters:

- **pJob** - points to the print job.

- **pPage** - points to the PRIpageStruct which is to receive the information. The mPage member must specify the vertical and horizontal page for which to retrieve the page information. If both are specified as zero, the default job page information is retrieved. If the specified page does not exist, the information of the last page is retrieved.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageStruct pageInfo;
qprierr err = PRIgetPageInfo( theJob, &pageInfo );
if ( !err )
{
  // use info
}
```

See also PRIsetPageInfo

PRIgetPageSetup

PRIgetPageSetup()

Fetches a copy of the page setup information of the given print job, or the default printer if pJob is null.

Note: PRIgetPageSetup allocates memory for the page setup information. PRIdestroyPageSetup must be called when the page setup information is no longer required.

Parameters:

- **pJob** - points to the print job.
- **pPageSetup** - points to the pointer of a PRIpageSetup structure which is to receive the page setup information.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageSetup* pageSetup;
qprierr err = PRIgetPageSetup( theJob, pageSetup );
if ( !err )
{
  // use page setup
  PRIdestroyPageSetup( pageSetup );
}
```

See also PRIsetPageSetup, PRIcopyPageSetup, PRIdestroyPageSetup, PRIopenPageSetupDialog, PRIchangeJobPageSetup

PRIgetPageSetupItem

qlong PRIgetPageSetupItem(PRIpageSetup*, qlong pItem) qlong PRIgetPageSetupItem(qcrb pCrb, qlong pItem)

Returns the value of the specified page setup item from the page setup data or data collection.

Parameters:

- **pPageSetup** - points to the page setup data.

OR

- **pCrb** - points to the Omnis data collection storing the page setup data.
- **pItem** - specifies the item to be returned. This is one of the PRI_PS_xxx defines. See PRIpageSetup structure for full details.
- **return** - returns the value of the item.

Example:

```
qlong copies = PRIgetPageSetupItem( theCrb, PRI_PS_COPIES );
```

See also PRIsetPageSetupItem, PRIpageSetupToCRB, PRIpageSetupFromCRB

PRIgetPaperDimensions

qprierr PRIgetPaperDimensions(PRIjob pJob, const qpripage* pPage, qprirect* pPaperBounds, qprirect* pPrintBounds) qprierr PRIgetPaperDimensions(PRIpageSetup* pPageSetup, qprirect* pPaperBounds, qprirect* pPrintBounds)

Fetches the paper boundaries of the given print job and page number, or the given page setup.

Parameters:

- **pJob** or **pPageSetup** - points to the print job or page setup from which to retrieve the paper dimensions.
- **pPage** - specifies the page for which to retrieve the paper dimensions.
- **pPaperBounds** - points to the qprirect structure which is to receive the paper coordinates. The top and left is always zero.
- **pPrintBounds** - points to the qprirect structure which is to receive the coordinates of the printable area local to pPaperBounds.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprirect paperBounds, printBounds;
qpripage page(1,1);
qprierr err = PRIgetPaperDimensions( theJob, &page, &paperBounds, &printBounds );
```

See also PRIgetPageSetup

PRIgetParmStruct (v4.2)

PRIparmStruct* PRIgetParmStruct(PRIjob pJob)

Returns the parameter structure associated with a print job. The user process should not free the memory returned by this call. struct PRIparmStruct is defined in PRI.HE

- **pJob** - points to the print job.

PRIgetSectionInfo

qprierr PRIgetSectionInfo(PRIjob pJob, PRIsectionStruct* pSection)

Fetches information about the specified section.

Parameters:

- **pJob** - points to the print job.
- **pSection** - points to the PRIsectionStruct which is to receive the information. mIdent must specify the id of the section for which the info is to be fetched.
- **return** - returns one of the PRI_ERR_xxx error constants. If PRI_ERR_INVALID_SECTION is returned, the section was not found.

Example:

```
PRIsectionStruct sectInfo;
sectInfo.mIdent = 1001;
qprierr err = PRIgetSectionInfo( theJob, &sectInfo );
```

See also PRIcreateSection, PRIdeleteSection, PRIsetSectionInfo, PRIgrowSection

PRIgetSysError

qlong PRIgetSysError(qprierr pError)

Returns the system error code which is embedded in the pError code if pError has the PRI_ERR_SYSERROR bit set. This function can be used to return the system error code when a system error has occurred.

Parameters:

- **pError** - specifies the print manager error value.
- **return** - returns the system error code.

Example:

```
if ( err & PRI_ERR_SYSERROR )
{
  qlong sysErr = PRIgetSysError( err );
}
```

See also PRIgetError, PRIsetError, PRIshowError, PRIgetErrorText

PRIgrowSection

qprierr PRIgrowSection(PRIjob pJob, qlong pSectionID, qpridim pHorz, qpridim pVert)

Grows the specified section by the specified amounts. Any objects belonging to the section who's floating properties are enabled will be effected by the growth of the section.

Parameters:

- **pJob** - points to the print job.

- **pSectionID** - identifies the section to be grown.
- **pHorz** - specifies the horizontal amount by which the section is to grow.
- **pVert** - specifies the vertical amount by which the section is to grow.

Example:

```
qprierr err = PRIgrowSection( theJob, 1001, 0, PRI_CM );
```

See also PRIcreateSection, PRIdeleteSection, PRIgetSectionInfo, PRIsetSectionInfo

PRIinitDestinationParms

void PRIinitDestinationParms(PRIdestParmStruct* pDestParms)

Initializes the destination parameters to default values.

Parameters:

- **pDestParms** - the destination parameters to be initialized.

Example:

```
PRIdestParmStruct destParms;
PRIinitDestinationParms( &destParms );
```

See also PRIvalidateDest

PRIisDeviceOpen

qbool PRIisDeviceOpen(qlong pDest)

Returns true if the specified device has been opened by PRIopenDevice.

Parameters:

- **pDest** - specifies the device.

Example:

```
if ( PRIisDeviceOpen( PRI_DEST_PRINTER ) )
{
  qprierr err = PRIcloseDevice( PRI_DEST_PRINTER );
}
```

See also PRIopenDevice, PRIcloseDevice, PRIsendTextToDevice, PRIsendDataToDevice, PRIflushDevice

PRIkillJob

qprierr PRIkillJob(PRIjob pJob)

Closes the given print job, all active output devices, and destroys the print job.

Parameters:

- **pJob** - points to the print job to be closed. WARNING: Any further reference to this print job after calling this function may result in a crash.
- **return** - returns one of the PRI_ERR_xxx error constants. If a fatal error occurred while printing, this error will be returned by PRIkillJob.

Example:

```
qprierr err = PRIgetError( theJob );
if ( err )
{
  PRIshowError( err );
  PRIkillJob( theJob );
}
```

See also PRIstartJob, PRIendJob

PRIloadJob

qprierr PRIloadJob(PRIparmStruct* pParms, strxxx* pFile) qprierr PRIloadJob(PRIparmStruct* pParms, qHandle pRepData)

Loads a report from file or memory, which was previously printed to disk or a memory handle and sends it to the destination specified by pParms.

Parameters:

- **pParms** - points to the parameter structure which specifies the print job properties. For PRIloadJob, only the destination and destination related information needs to be set.
- **pFile** - specifies the full path and name of the file.

OR

- **pRepData** - the handle which holds the report data.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIopen();
if ( !err )
{
  PRIparmStruct parms(qnil);
  parms.mDestParms = ECOgetDeviceParms( eci->mInstLocp );
  qlong savedDest = parms.mDestParms->mDest;
  parms.mDestParms->mDest = PRI_DEST_PRINTER;
  err = PRIloadJob( &parms, fileName );
  PRIclose();
  parms.mDestParms->mDest = savedDest;
}
```

See also PRIparmStruct , PRIstartJob

PRImakeCustomProc

HINSTANCE pInstance)

This function is called to create a FARPROC of your custom output device message function. When registering a custom output device the device manager must be given a FARPROC pointer to your custom device message function.

Example:

See section 'A simple external output device'

See also PRIdisposeCustomProc, PRIregisterOutput

PRInormPos

qprierr PRInormPos(PRIjob pJob, qpripos* pPos)

If the given position is local to a section, it converts the report position to the coordinate space of the section.

Parameters:

- **pJob** - specifies the print job.
- **pPos** - specifies the position to be normalized.

Example:

```
qprirect theRect( 10000, 0, 50000, 8000 );
qpripos thePos( 1001, theRect ); // 1001 = section ident
qprierr err = PRInormPos( theJob, &thePos );
```

See also PRIcreateSection

PRIopen

qprierr PRIopen()

Opens and initializes the print manager, if it isn't open already. You must always call PRIopen() before and PRIclose() after calls to the print manager. A good rule of thumb is to call PRIopen and PRIclose on a per function basis, that is any function which uses the print manager calls PRIopen on entry, and PRIclose on exit. Failing to do so may cause crashes.

Parameters:

- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr myFunction()
{
  qprierr err = PRIopen();
  if ( !err )
  {
    // use the print manager
    PRIclose();
  }
  return err;
}
```

See also PRIclose

PRIopenChangePrinterDialog (Windows only)

qbool PRIopenChangePrinterDialog()

Opens the Omnis change printer dialog.

Parameters:

- **return** - returns true if the user has changed the printer.

Example:

```
if ( PRIopenChangePrinterDialog() )
{
  // print
}
```

See also PRIbuildPrinterList, PRIopenPageSetupDialog, PRIopenJobSetupDialog

PRIopenDestinationDialog

qprierr PRIopenDestinationDialog(PRIdestParmStruct* pDestParms,qbool pShowParms, qbool* pChanged)

Opens the standard Omnis destination dialog.

Parameters:

- **pDestParms** - points to the device parameters to be used. You should call ECOgetDeviceParms to retrieve the standard global set of device parameters used by Omnis as the value for this parameter.
- **pShowParms** - if true, the parameters pane is shown when the dialog opens.
- **pChanged** - points to the Boolean which is to receive the result of the dialog. It will be set to qtrue if the user changed the destination or parameters.

Example:

```
qbool changed;
PRIdestParmStruct* destParms = ECOgetDeviceParms( eci->mInstLocp );
qprierr err = PRIopenDestinationDialog(destParms,qfalse,&changed);
if ( changed )
{
  // do something
}
```

See also PRIopenPageSetupDialog , PRIopenChangePrinterDialog

PRIopenDevice

qprierr PRIopenDevice(PRIdestParmStruct* pDestParms)

Opens the device with the device parameters specified by pDestParms. Once a device has been opened, PRIsendTextToDevice and PRIsendDataToDevice can be used to transmit text or data to the device. Not all output devices can be opened in this way, and can only be opened by a print job. When finished with the device, you MUST call PRIcloseDevice.

Parameters:

- **pDestParms** - specifies the device and the parameters of the device to be opened.

Example:

```
qprierr myPrint()
{
  PRIdestParmStruct* destParms = ECOgetDeviceParms(eci->mInstLocp);
  destParms->mDest = PRI_DEST_PRINTER;
  qprierr err1 = PRIopenDevice( destParms );
  qprierr err2 = PRI_ERR_NONE;
  if ( !err1 )
  {
    // print 1 or more jobs, or send text to device
    err2 = PRIcloseDevice( PRI_DEST_PRINTER );
  }
  return err1 ? err1 : err2;
}
```

See also PRIcloseDevice, PRIsendTextToDevice, PRIsendDataToDevice, PRIflushDevice, PRIisDeviceOpen

PRIopenGetFileDialog

qbool PRIopenGetFileDialog(str255* pName)

This function opens the standard get file dialog for saved report files (reports printed to the PRI_DEST_DISK destination).

Parameters:

- **pName** - points to the str255 which is to receive the full path and file name of the file picked by the user.
- **return** - returns qtrue if the user has clicked ok.

Example:

```
qprierr err = PRIopen();
if ( !err )
{
  PRIdestParmStruct* destParms = ECOgetDeviceParms(eci->mInstLocp);
  str255 fname;
  if ( PRIopenGetFileDialog( fname ) )
  {
    destParms->mDest = PRI_DEST_PREVIEW;
    err = PRIloadJob( destParms, fname );
  }
  PRIclose();
}
if ( err ) PRIshowError( err );
```

See also PRI_DEST_DISK

PRIopenJobSetupDialog

qprierr PRIopenJobSetupDialog(PRIjob pJob, qbool* pOk)

Opens the job setup dialog. This function can only be called for an active print job, and prior to the first page being generated. It should normally be called immediately after the call to PRIstartJob.

Parameters:

- **pJob** - specifies the active print job for which to open the job setup dialog.
- **pOk** - points to the Boolean which is to receive the result of the dialog. If the user clicked on Ok, it will be set to qtrue, otherwise it will be qfalse.

Example:

```
qbool ok;
qprierr err = PRIopenJobSetupDialog( theJob, &ok );
if ( ok )
{
  // print
}
```

See also PRIopenPageSetupDialog, PRIopenChangePrinterDialog

PRIopenPageSetupDialog

---

qprierr PRIopenPageSetupDialog(PRIpageSetup* pPageSetup, qbool* pChanged)

Opens the operating system page setup window.

Parameters:

- **pPageSetup** - points to the page setup information which is to be used with the dialog. If pPageSetup is NULL, the print managers global page setup is used with the dialog. PRIgetPageSetup can be called to fetch the global page setup.
- **pChanged** - points to the Boolean which is set to true if the user has changed the page setup.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qbool changed;
qprierr err = PRIopenPageSetupDialog( NULL, &changed );
if ( changed )
{
  // do something
}
```

See also PRIgetPageSetup, PRIsetPageSetup, PRIcopyPageSetup, PRIdestroyPageSetup, PRIopenChangePrinterDialog, PRIopenJobSetupDialog

PRIpageSetupFromCRB

---

qprierr PRIpageSetupFromCRB(qcrb pCrb, PRIpageSetup*& pPageSetup)

Retrieves the page setup for the running operating system from an Omnis data collection. If valid driver information for the current operating system cannot be found in the data collection, the function will create default driver info. When you have finished with the page setup, you must call PRIdestroyPageSetup to free the memory occupied by the page setup.

Parameters:

- **pCrb** - points to the data collection which may hold page setup information for various operating systems.

- **pPageSetup** - points to the page setup structure which is to receive the page setup info. You must set the signature member of the driver info. Usually you should specify PRI_DRV_BEST which will retrieve the driver information most suitable for the current operating system, but you may specify any signature compatible with the current operating system.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See PRIpageSetupToCRB

See also PRIpageSetupToCRB, PRIdestroyPageSetup, EXTcrb class

PRIpageSetupToCRB

qprierr PRIpageSetupToCRB(qcrb pCrb, PRIpageSetup* pPageSetup)

Stores/adds the given page setup information to the given Omnis data collection.

Note: The same data collection can store page setup data from more then one platform.

Parameters:

- **pCrb** - points to the data collection to which the page setup information is to be added.
- **pPageSetup** - points to the page setup structure which specifies the page setup information to be stored.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageSetup* pageSetup;
qprierr err = PRIgetPageSetup( NULL, pageSetup );
if ( !err )
{
  EXTcrb crb;
  err = PRIpageSetupToCRB( crb.crb(), pageSetup );
  if ( !err )
  {
    PRIdestroyPageSetup( pageSetup );
    qlong size = crb.getFlatSize();
    qchar* buffer = MEMmalloc( size );
    size = crb.flatten( buffer, size );
    // store flat page setup
    // read flat page setup
    if ( crb.unflatten( buffer, size ) )
    {
      err = PRIpageSetupFromCRB( crb.crb(), pageSetup );
      if ( !err )
      {
        // destroy when finished with page setup
        PRIdestroyPageSetup( pageSetup );
      }
    }
  }
  PRIdestroyPageSetup( pageSetup );
}
```

See also PRIpageSetupFromCRB, PRIdestroyPageSetup, EXTcrb class

PRIredirectJob

qprierr PRIredirectJob(PRIjob pJob, PRIdestParmStruct * pDestParms)

PRIredirectJob allows you to send an existing job to an alternative report destination.

Parameters:

- **pJob** - points to the print job.
- **pDestParms** - specifies the alternative destination and destination parameters.

Example:

```
// make a copy of the global device parameters (we don't want to effect the global parameters )
PRIdestParmStruct destParms = *ECOgetDeviceParms( eci->mLocLocp );
// print page one to printer
destParms.mDest = PRI_DEST_PRINTER;
destParms.mPages = str255("1");
qprierr err = PRIredirectJob( theJob, &destParms );
```

See also PRIdestParmStruct, PRIstartJob

PRIregisterOutput

qprierr PRIregisterOutput(PRIdeviceInfoStruct* pInfo, qlong pSuperClass, FARPROC pProc)

This function registers the specified custom output device with the output manager.

Parameters:

- **pInfo** - Points to the custom device information. On return the mID member will be set to the runtime id of the registered device.
- **pSuperClass** - Specifies one of the PRI_DEST_EXTxxx defines. This is the internal device from which the custom device wishes to inherit from.
- **pProc** - Specifies the custom message function.

Example:

See section 'A simple external output device'

See also PRIdeviceInfoStruct, PRIunregisterOutput, PRImakeCustomProc, PRIdisposeCustomProc

PRIsendDataToDevice

qprierr PRIsendDataToDevice(qlong pDest, qbyte* pData, qlong pDataLen)

Sends raw data to the specified device. The device must have been opened by calling PRIopenDevice.

Parameters:

- **pDest** - id of the device.
- **pData** - points to the data to be transmitted.

- **pDataLen** - specifies the length of the data to be transmitted.

Example:

```
qprierr mySendDataToFile( EXTfldval& pData )
{
  qprierr err = PRIopen();
  if ( err ) return err;
  // open the printer device (copy device parameters so we don't alter them )
  PRIdestParmStruct destParms = *ECOgetDeviceParms( eci->mLocLocp );
  destParms.mDest = PRI_DEST_FILE;
  destParms.mTextFile = str255("C:\MYFILE.TXT");
  err = PRIopenDevice( &destParms );
  if ( !err )
  {
    qHandle han = pData.getHandle( qfalse );
    qHandlePtr hp(han,0);
    err = PRIsendDataToDevice( PRI_DEST_FILE, &hp[0], hp.dataLen() );
    qprierr err2 = PRIcloseDevice( PRI_DEST_FILE );
    if ( !err ) err = err2;
  }
  PRIclose();
  return err;
}
```

See also PRIopenDevice, PRIcloseDevice, PRIsendTextToDevice, PRIflushDevice, PRIisDeviceOpen

PRIsendTextToDevice

qbool pNewLine, qbool pFormFeed)

Sends the given text to the specified device. The device must have been opened by calling PRIopenDevice.

Parameters:

- **pDest** - id of the device.
- **pText** - points to the text to be transmitted.
- **pTextLen** - specifies the length of the text to be transmitted.
- **pNewLine** - if true, the device will transmit new line character or characters after the text.
- **pFormFeed** - if true, the device will transmit a form feed character.

Example:

```
qprierr mySendTextToPrinter( EXTfldval& pText )
{
  qprierr err = PRIopen();
  if ( err ) return err;
  // open the printer device (copy device parameters so we don't alter them )
  PRIdestParmStruct destParms = *ECOgetDeviceParms( eci->mLocLocp );
  destParms.mDest = PRI_DEST_PRINTER;
  err = PRIopenDevice( &destParms );
  if ( !err )
  {
    qHandle han = pText.getHandle( qfalse );
```

```
    qHandlePtr hp(han,0);
    err = PRIsendTextToDevice( PRI_DEST_PRINTER, &hp[0], hp.dataLen(),qfalse, qfalse );
    // Note: sending text does not check for line feed characters inside the data.
    // If the text contains these, the data must be separated and send line by line
    qprierr err2 = PRIcloseDevice( PRI_DEST_PRINTER );
    if ( !err ) err = err2;
  }
  PRIclose();
}
```

See also PRIopenDevice, PRIcloseDevice, PRIsendDataToDevice, PRIflushDevice, PRIisDeviceOpen

PRIsetDeviceInfo

qprierr PRIsetDeviceInfo(qlong pDest, PRIdeviceInfoStruct* pInfo)

Updates the display name, icon id and visibility of the device from the information specified by pInfo.

Parameters:

- **pDest** - specifies the device.
- **pInfo** - point to the PRIdeviceInfoStruct which specifies the new information.

Example:

```
PRIdeviceInfoStruct devInfo;
qprierr err = PRIgetDeviceInfo( PRI_DEST_PRINTER, &devInfo );
if ( !err )
{
  devInfo.mName = str31("Laser Writer");
  err = PRIsetDeviceInfo( PRI_DEST_PRINTER, &devInfo );
}
```

See also PRIdeviceInfoStruct, PRIgetDeviceInfo, PRIgetDeviceName

PRIsetError

void PRIsetError(PRIjob pJob, qprierr pErr)

Stores the specified error code in the given print job. If the error code is non-zero, any further calls to print manager functions for the given job will return without action. Once a print job has recorded an error the job can only be closed by calling PRIendJob or PRIkillJob. To clear an error in a print job you can specify PRI_ERR_NONE.

Parameters:

- **pJob** - points to the print job in which to store the error.
- **pErr** - specifies the error code to be stored.

Example:

```
qprierr err = PRIgetError( theJob );
if ( err )
{
  PRIshowError( err );
  PRIsetError( theJob, PRI_ERR_NONE );
}
```

See also PRIgetError, PRIgetSysError, PRIshowError, PRIgetErrorText

PRIsetPageInfo

qprierr PRIsetPageInfo(PRIjob pJob, PRIpageStruct* pPage)

Sets the page information for the specified page. This function can only be called while the specified page is the current page i.e. PRIendPage has not been called for the page, otherwise a page closed error will be returned.

Parameters:

- **pJob** - points to the print job.
- **pPage** - points to the PRIpageStruct which specifies the information. The mPage member must specify the vertical and horizontal page for which to set the page information. Only the local, header and footer boundaries can be changed using this function. On return the global bounds will have been updated to reflect the changed information.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
PRIpageStruct pageInfo;
pageInfo.mPage.mVert = currentPage;
pageInfo.mPage.mHorz = 1;
qprierr err = PRIgetPageInfo( theJob, &pageInfo );
if ( !err )
{
pageInfo.mHeaderBounds.bottom += PRI_INCH;
pageInfo.mLocalBounds.top += PRI_INCH;
err = PRIsetPageInfo( theJob, &pageInfo );
}
```

See also PRIgetPageInfo

PRIsetPageSetup

qprierr PRIsetPageSetup(PRIpageSetup* pPageSetup)

Sets the page setup information of the default printer.

Parameters:

- **pPageSetup** - points to the PRIpageSetup structure containing the page setup information.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
// save the global page setup
PRIpageSetup *savedPageSetup, *newPageSetup;
PRIgetPageSetup( NULL, savedPageSetup );
// set the page setup from one stored with document
qprierr err = PRIpageSetupFromCRB( theCrb, newPageSetup );
if ( !err )
{
  PRIsetPageSetup( newPageSetup );
  PRIdestroyPageSetup( newPageSetup );
  // print
}
```

```
// restore saved page setup
if ( savedPageSetup )
{
  PRIsetPageSetup( savedPageSetup );
  PRIdestroyPageSetup( savedPageSetup );
}
```

See also PRIgetPageSetup, PRIcopyPageSetup, PRIdestroyPageSetup, PRIopenPageSetupDialog

PRIsetPageSetupItem

---

qbool PRIsetPageSetupItem(PRIpageSetup* pPageSetup, qlong pItem, qlong pLngValue)

---

qbool PRIsetPageSetupItem(qcrb pCrb, qlong pItem, qlong pLngValue)

Sets the specified page setup item to the given value in the page setup data or data collection. Changing one item may effect other items if they are related.

Parameters:

- **pPageSetup** - points to the page setup data.

OR

- **pCrb** - points to the Omnis data collection storing the page setup data.
- **pItem** - specifies the item to be changed. This is one of the PRI_PS_xxx defines. See PRIpageSetup structure for full details.

pLngValue - specifies the new value for the item.

Example:

```
if ( PRIsetPageSetupItem( myPageSetup, PRI_PS_PAPER, PRI_PA_A4 ) )
{
qprirect paperBounds, printBounds;
err = PRIgetPaperDimensions( myPageSetup, &paperBounds, &printBounds );
}
```

See also PRIpageSetup, PRIgetPageSetupItem

PRIsetSectionInfo

---

qprierr PRIsetSectionInfo(PRIjob pJob, PRIsectionStruct* pSectionStruct)

Sets information of the specified section.

Parameters:

- **pJob** - points to the print job.
- **pSection** - points to the PRIsectionStruct which contains the new section information. mIdent must specify the id of the section for which the info is to be altered.
- **return** - returns one of the PRI_ERR_xxx error constants. If PRI_ERR_INVALID_SECTION is returned, the section was not found.

Example:

```
PRIsectionStruct sectInfo;
sectInfo.mIdent = 1012;
qprierr err = PRIgetSectionInfo( theJob, &sectInfo );
if ( !err )
{
  sectInfo.mPos.offset( 0, PRI_INCH );
  err = PRIsetSectionInfo( theJob, &sectInfo );
}
```

See also PRIcreateSection, PRIdeleteSection, PRIgetSectionInfo, PRIgrowSection

PRIshowError

void PRIshowError(qprierr pErr)

Opens an error message box.

Parameters:

- **pErr** - specifies the error code for which to display an error message.

Example:

```
qprierr err = PRIgetError( theJob );
if ( err )
{
  PRIshowError( err );
  PRIsetError( theJob, PRI_ERR_NONE );
}
```

See also PRIgetErrorText, PRIgetError, PRIsetError, PRIgetSysError

PRIstartJob

qprierr PRIstartJob(PRIparmStruct* pParms)

Starts a new print job.

Parameters:

- **pParms** - points to a PRIparmStruct which specifies the print job properties. If successful , some information will be returned in this structure. See PRIparmStruct for more information.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See section ‘A simple print job’

See also PRIparmStruct, PRIendJob, PRIkillJob, PRIloadJob, PRIredirectJob

PRIstartPage

qprierr PRIstartPage(PRIjob pJob)

Generates a new page. PRIstartPage will always call PRIendPage for the previous page before generating the new page. When a new page is generated the PM_INIT_PAGE and PM_ADD_HEADER_OBJECTS are generated.

Parameters:

- **pJob** - points to the print job.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
qprierr err = PRIstartPage( theJob );
```

See also PRIendPage, PRIejectPage

PRItextHeight

qpridim PRItextHeight(PRIjob pJob, qfnt* pFnt)

Returns the height of the given font. If the jobs output device is text based, the mLineHeight of the jobs destination parameters is returned.

Parameters:

- **pJob** - points to the print job.
- **pFnt** - specifies the font.

Example:

```
qfnt fnt = fntSystem;
qpridim height = PRItextHeight( theJob, &fnt );
```

See also PRItextWidth

PRItextWidth

qpridim PRItextWidth(PRIjob pJob, qchar* pAdd, qlong pLen, GDItextSpecStruct* pTextSpec)

Returns the width of the given text. If the jobs output device is text based, the mCharWidth of the jobs destination parameters multiplied by pLen is returned.

Parameters:

- **pJob** - points to the print job.
- **pAdd** - points to the text data.
- **pLen** - specifies the text length.
- **pTextSpec** - specifies the text style information.

Example:

```
str255 text("Some text");
GDItextSpecStruct tspec( fntSystem, styBold );
qpridim width = PRItextWidth( theJob, &text[1], text[0], &tspec );
```

See also PRItextHeight

PRItextWidthEx

qpridim PRItextWidthEx (PRIjob pJob, qchar* pAdd, qlong pLen, GDItextSpecStruct* pTextSpec, qbool pJstText)

Returns the width of the given styled text when pJstText is true. Same as PRItextWidth except that it has one more parameter (qbool pJstText) which is true when the supplied text string is to be treated as styled text (so it can contain styled text escapes).

Parameters:

- **pJob** - points to the print job.
- **pAdd** - points to the text data.
- **pLen** - specifies the text length.
- **pTextSpec** - specifies the text style information.
- **pJstText** - true when supplied text string is styled text.

See also PRItextWidth

PRIunflattenDriverInfo

qprierr PRIunflattenDriverInfo(PRIpageSetup* pPageSetup, qfldval pData)

Expands the driver data stored in pData and applies it to pPageSetup.

Parameters:

- **pPageSetup** - points to the page setup structure which is to receive the expanded driver info.
- **pData** – contains the flattened driver data.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

```
EXTfldval fval;
// load the driver data into fval
qprierr err = PRIunflattenDriverInfo( myPageSetup, fval.getFldVal() );
// See also PRIflattenDriverInfo
```

See also PRIgetDriverSigniture, PRIgetFlattDriverInfoSize, PRIflattenDriverInfo

PRIunregisterOutput

qprierr PRIunregisterOutput(qlong pID)

Calling this function will remove the specified custom device from the list of installed output devices.

Parameters:

- **pID** - specifies the id of the custom device to be removed.
- **return** - returns one of the PRI_ERR_xxx error constants.

Example:

See section ‘A simple external output device’

See also PRIdeviceInfoStruct, PRIregisterOutput, PRImakeCustomProc, PRIdisposeCustomProc

PRIvalidateDest

qprierr PRIvalidateDest(PRIdestParmStruct* pDestParms)

Validates the destination and destination parameters.

Parameters:

- **pDestParms** - the destination parameters to be validated.

Example:

```
// get a copy of global device parameters
PRIdestParmStruct destParms = *ECOgetDeviceParms( eci->mLocLocp );
destParms.mDest = PRI_DEST_FILE;
qprierr err = PRIvalidateDest( &destParms );
// if the file path and name are empty, the device will prompt the user,
// otherwise it is left alone.
```

See also PRIinitDestinationParms

Chapter 14—DAM API Reference

Creating your own non-visual DAMs

Introduction

This document details the APIs used to write non-visual DAM components and it assumes familiarity with the earlier sections of this manual. For general documentation on the use of Omnis Studio Object DAMs, please refer to the Omnis Programming Manual.

Omnis Studio DAMs (Data Access Modules) are specialised, non-visual external components that allow the Omnis Studio development tool to access data sources through a consistent programming interface. The non-visual DAMs replace the old DAM external libraries. This was required primarily to allow an object-oriented approach to the DAM interface to be adopted and secondly to allow database access in a multi-threaded environment as part of the Omnis Web Server.

The objects provided by a DAM represent the session, statement and worker objects. These objects provide properties and methods to the application to allow it to manage communication with the external database server. Using the non-visual approach, an application creates object variables of a particular DAM class that are instantiated when the object is created by the component. There are a group of common operations that apply to all DAM objects (those provide by the “base class”) and a set of DAM specific operations based on the type of object (those provided by the “implementation” or “derived object”).

The primary object is the *session object,* internally known as **tqfDAMbaseObj**, which manages the connection for a particular DAM type and is used to pass commands to the database server. Every DAM has a unique DAM type associated with it. For example if an application requires a Sybase session then it will use a SybaseDAM object. The tqfDAMbaseObj object is wrapped within a container object called **tqfDAMObjCont** so that it is recognised by the component library.

The application uses the session object to create a *statement object,* internally known as **tqfDAMStatementObj** in order to issue SQL statements. A statement object can be used for all types of statements, e.g. SQL, PL/SQL cursors and remote procedures. There can be multiple statement objects which share a common context provided by a single session object. The tqfDAMStatementObj object is wrapped within a container object called **tqfDAMStatementCont**.

For use with the multi-threaded Omnis Web Server, we created the concept of a *session pool*; a pre-defined array of session objects sharing the same DAM type. An application can request session objects from this pool. Each object in the pool is connected with the same attributes when the pool is created. This has an advantage in performance since the connection is immediately available when a client requests an object.

The multi-threaded nature of session objects, session pools and statement objects are part of the underlying implementation and DAM developers do not need to implement any specific functionality, although they should be aware of the need to write thread safe code.

Additionally, there are 7 main support classes designed to encapsulate specific tasks relating to data manipulation:

DAMrpcDefn	**DAMrpcDefn** is the Remote Procedure Call definition class. This class encapsulates the data for the database server procedures and their parameters. This is used by the derived session object.
DAMErrorInfo	**DAMErrorInfo** is the error handling class. This stores and reports any errors whether internally or remotely and is used by both the derived session and statement objects.
DAMCharMapTable	**DAMCharMapTable** is the character mapping class that is used to manipulate 8-bit characters moved between the application and the database. Its creation is based on a character map file and is used by the derived session object.
DAMTypeTable	**DAMTypeTable** contains the internal mapping between Omnis data types and SQL C types. This object is created during the construction of the derived session object and is generally standard among databases.
DAMData	**DAMData** encapsulates the DAMParam objects relating to a single SQL statement. This performs functions that relate to all of the parameters. It is used by the derived session object.
DAMParam	**DAMParam** encapsulates the data and information for a single Omnis bind variable. This includes both in and out parameters or a return value from a procedure. These parameters are contained within a DAMData object.
DAMunicodeText	**DAMunicodeText** is a utility class encapsulating several Unicode conversion functions provided by the component library. It can be used to create and concatenate text in various Unicode encodings.

There are also two additional classes designed to support asynchronous worker objects:

StatementWorker	An interface object that can be used to $init() and $start() worker background tasks. These callback into the Omnis core on completion.
StatementWorkerDelegate	Delegate to StatementWorker, this class performs the actual work of the worker object and can be spawned (and detached) multiple times. StatementWorker can only interface with one StatementWorkerDelegate at a time.

Asynchronous worker objects may be considered an extension to the basic DAM functionality and are discussed later.

The main base objects, tqfDAMbaseObj and tqfDAMStatementObj, need to be sub classed by your implementation and any pure virtual functions defined by the base class must be implemented.

The remainder of this document introduces you to a *generic DAM object* which you can use as the starting point for your own implementation. The developer guide addresses the techniques involved in developing and testing your DAM and describes the sequencing of the various processes involved and how they relate to your derived objects.

The reference section describes the API's for the classes shown above and gives examples on how to write the pure virtual functions in your tqfDAMbaseObj and tqfDAMStatementObj implementations.

Figure 1. UML class diagram showing relationship between DAM object classes.

Developer Guide

This section discusses in greater detail the processes involved in developing a custom DAM.

Key concepts to be aware of whilst developing your DAM are:

Cross platform development

Working with the Omnis component library and DAM library, it is possible to write cross-platform code easily provided that your database server provides the appropriate clientware for each platform. In essence, you maintain one source file (and associated header

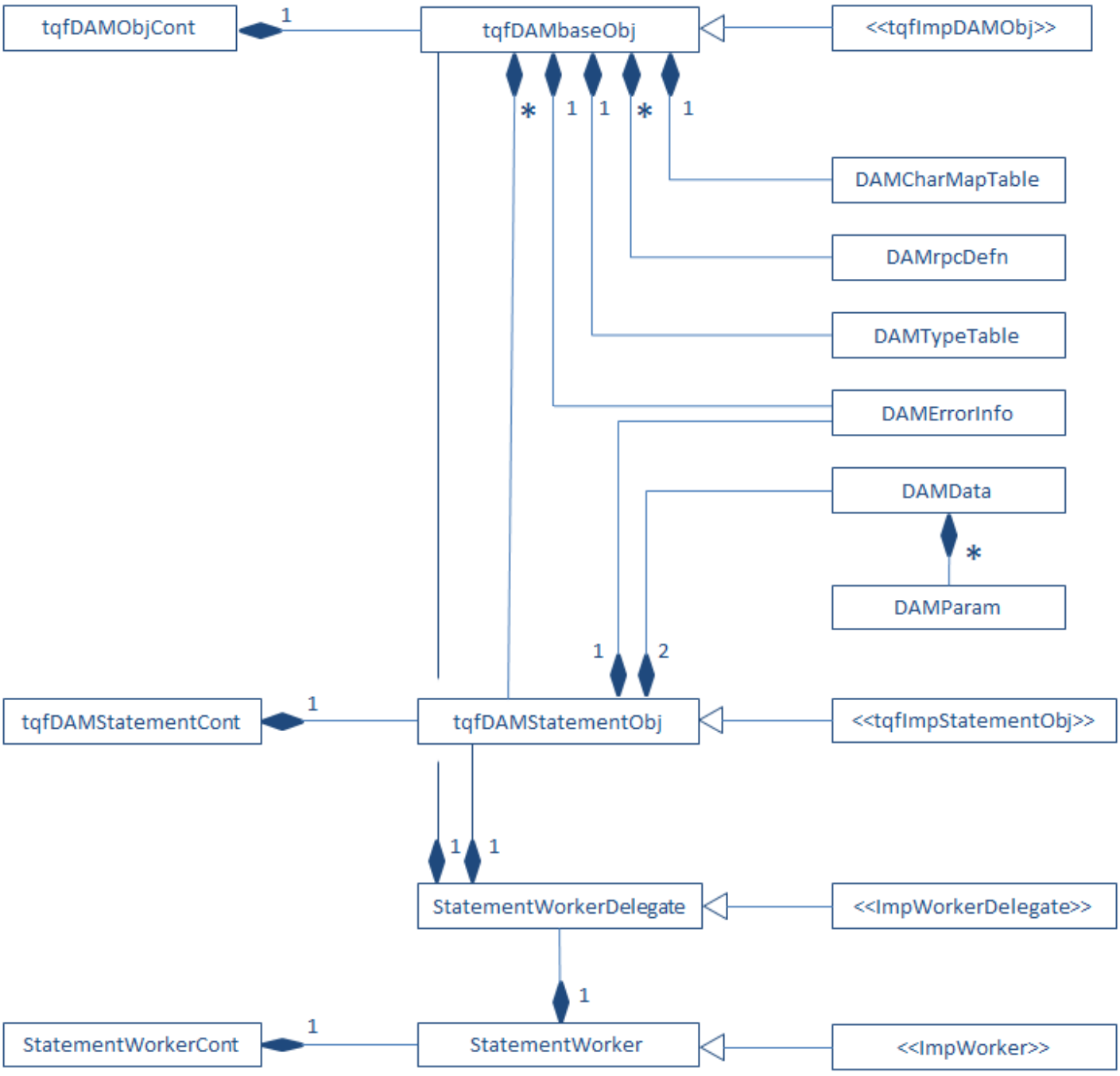
tqfDAMObjCont
1
tqfDAMbaseObj
<<tqfImpDAMObj>>
*
1
1
*
1
DAMCharMapTable
DAMrpcDefn
DAMTypeTable
DAMErrorInfo
DAMData
*
DAMParam
1
2
tqfDAMStatementCont
1
tqfDAMStatementObj
<<tqfImpStatementObj>>
1
1
StatementWorkerDelegate
<<ImpWorkerDelegate>>
1
StatementWorkerCont
1
StatementWorker
<<ImpWorker>>

Figure 3:

file) for your session object, one source file (and associated header file) for your statement object and a file containing resource strings used by your implementation. The same files are used on each platform you will be developing for.

The Omnis component library is designed with cross platform development in mind. By using the component library and not relying on platform-specific SDKs, you insure your code against issues such as missing API calls, different function arguments and behavior.

Unicode

If you are writing a DAM for use with Studio 5 or later, your component must support Unicode, although the base class supports both Unicode and non-Unicode targets, for use with Studio 4.3 and earlier for example.

Even if your clientware does not support Unicode, there are several Unicode conversion functions and classes built into the component and DAM libraries which readily convert between Omnis' UTF-32 character set and non-Unicode, UTF-8 or UTF-16. The DAMunicode-Text class can also be overridden allowing you to provide additional functionality if required.

Debugging and Maintainability

To simplify the process of writing a new component and to avoid having to start from scratch, a template DAM is provided, available for download from the Omnis website. The template DAM is supplied with project files for all supported platforms:

- Windows: Microsoft Visual C++ 2008 & 2013 projects
- Macintosh: XCode 8.1 project plus support files
- Linux: gcc 4.2 compatible makefile (should also work with other versions)

This generic DAM is the recommended starting point for any new development. It requires no clientware and contains stubs for all of the derived virtual methods. Debug messages built into most of the derived methods should assist with debugging as you develop your DAM.

Once you have established a build environment for your DAM, you should compile it and test it using Omnis Studio. (Ensure that you build a *Debug* target in order to do this!)

Make sure the component loads, that you can see it in the Interface Manager, that any constants have been registered (F9), that derived test methods and properties are visible within the session and statement objects. You should also be able to hit breakpoints set inside your derived C++ methods.

Moving forward, you should adhere to good programming practice by keeping your methods as compact as possible (breaking complex processes down if necessary). Use naming and formatting conventions to make your code easily readable and understandable (by other people). Also, try to minimize the time that your code is in a non-compiling state. Keeping your code in a usable state makes it easy to test changes to your DAM and the effect they might have on existing behaviour.

As a first step, you may first wish to rename your source files, target name, classes, structures, and constants to better suit your target database. Hint: you can change the DAM name that appears in Omnis by editing resource string 1000.

Study the DAM API carefully to ensure that you do not duplicate processes which the API already provides or does for you implicitly. The component and DAM libraries provide several classes for handling strings and converting data, some of which are alluded to in the generic code.

Folder Structure

As with other types of external component, DAMs need to be linked with the Omnis component library (COMPLIB). In addition to the component library, DAMs must also be linked with the DAM base library (DAMLIB) which contains additional header files your implementation will need to refer to. Any client API libraries specific to the target database will also need to be linked to the project.

The COMPLIB and DAMLIB SDKs can be downloaded from the Omnis website. For further information on building an external component please review the Omnis External Components Tutorial.

The recommended folder structure used when developing your DAM is shown below.

complib contains the static component library and associated header files. *damlib* contains the static DAM library and associated header files. The Unicode component library will be named similar to omnisu.lib (or omnisu64.lib). Likewise, the Unicode DAM library will be named damobju.lib (or damobju64.lib)

The template project looks for header files and libraries in ..\complib and ..\damlib as well as in the current folder by default.

Your custom session and statement source files are placed in the custom DAM folder (renamed as appropriate). You may also wish to place your client-specific header files and static libraries inside this folder.

Figure 4:

Mac OSX

For Mac OSX, the XCode project uses a folder structure as shown.

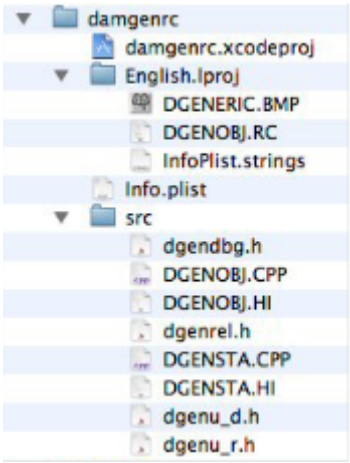

Figure 5:

Note that resource & localisation files are stored in the *English.lproj* sub-folder. Source files, including prefix header files (one for each build target) are stored in the *src* sub-folder.

The *info.plist* file contains bundle resource information.

The *InfoPlist.strings* file is a simple XML file which contains the bundle name (DAM name). These two files will need to be modified later. See the Renaming Files section for details.

Building the DAM

When the folder structure has been setup and the component & DAM base libraries have been built, you should be able to navigate to the custom DAM folder and build the template/generic DAM.

Once built and ready for testing, the DAM should be placed in the xcomp folder of the Omnis Studio development tree. When Omnis starts the DAM will be loaded along with all the other external components. If it loads successfully it will appear in the $root.$extobjects group in the Notation Inspector. If your DAM object does not appear in this group then an error message will appear in the Omnis trace log. Initially, this may be because the new component may have initialisation problems or may not be recognised as a viable Omnis component.

Later on, when you add external client libraries it is possible that one or more dynamic dependencies could not be resolved on the client machine; i.e. a client library could not be located.

Please note that Unicode components require the Unicode version of Omnis Studio and vice-versa.

Windows

For Windows platforms, this involves opening the .vcproj file (VC++ 2008) or .vcxproj file (VC++2013), selecting the required build configuration and selecting *Build*. If you want to debug the component, you will need to edit the project settings, change the build location to the Omnis\xcomp folder and add the omnis.exe as the debug executable. You can then start Omnis (F5) and verify that the custom DAM loads, e.g. by inspecting the $root.$components node and/or the Omnis trace log.

Note that as with other platforms, you can also set a breakpoint inside the GenericObjProc to trap the ECM_CONNECT message; If your component is viable, the Omnis will attempt to load it on startup.

Mac OSX

For Mac OS X, a separate Omnis resource compiler is required. This should also be downloaded from the Omnis website and installed into the /Developer/Tools folder before you attempt to build the DAM. See the *Omnis Resource Compiler* section for more details.

Next, copy the DGENOBJ.RC and DGENERIC.BMP files into the English.lproj folder (XCode looks for resource files at this location).

Double-click the xcodeproj file to start XCode, select the required target and configuration, then press *Build*. As above, you will need the change the build location for the Unicode Debug target if you want to debug the component. To debug, the DAM needs to be built into the Omnis.app/Contents/MacOS/xcomp folder. You will also need to add the Omnis.app as a *New Custom Executable*. Following this, you can use the *Build and Go* feature to start debugging.

If the component fails to load on starting Omnis, you can verify the integrity of the component by navigating to Omnis.app/Contents/MacOS/xco right-clicking on the component and selecting *Show Package Contents*. The *Contents* folder should be appear as shown.

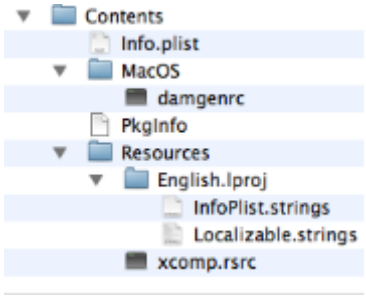

Figure 6:

Note the resource files which should be copied into the component package during the build process. Localizable.strings and xcomp.rsrc are generated by the Omnis resource compiler from the .RC files. xcomp.rsrc in particular must be present in order for Omnis to recognise the package as an Omnis external component. Clean the project, verify that the Omnis resource compiler is working and check your build phases if this file is missing.

Linux

For Linux, the gcc command line compiler is used to build the DAM. You will also need to download the Omnis resource compiler and install this for example into ~/omnisrcdir. See the Omnis Resource Compiler section for more details.

The Linux build requires certain environment variables to be setup before building can take place:

- **V4DIR** - This variable points to the location of certain additional Omnis core header files required to build components.
- **PATH** - This variable should be appended with the path to the Omnis resource compiler.
- **OMNISRC** - This variable should also be set to the folder containing the resource compiler.
- **LD_LIBRARY_PATH** - This variable should be set to the location of the omnisxi.so file.
- **omunicode** - This variable should be set to 1 to build the Unicode targets, 0 to build non-Unicode targets. Note: this is lower-case.

A *setenv* script is provided to simply this process, although it will require editing initially to set the values of the environment variables. It can subsequently be run to setup the build environment:

`. ~/setenv` (note the use of dot-space-tilde)

Whilst your source files are shared across Windows, Mac and Linux there is an important point to note regarding the Linux file format. Specifically, Linux requires simple linefeed characters at the end of each line of text whereas Windows and Macintish use linefeed and carriage return characters.

For this reason, a *DOS2UNIX* utility is provided for use on Windows. DOS2UNIX provides simple one-way conversion of text files, stripping out carriage return characters (as well as unsupported #pragmas in the source code). You should therefore *take a copy* of your source files before processing them using DOS2UNIX and *retain your original files* for making any code changes.

Failure to process line endings in your .cpp, .he and .rc files may cause the resulting binary file to be non-viable and/or resource strings may fail to load.

To install DOS2UNIX, place the executable file in the Windows\System32 folder.

Before building the DAM, the damobj.rc file must be manually copied into the project folder:

```
cp ../damlib/damobj.rc .
```

Additionally, omnisxi.so (X-Windows Interface library) should be copied from the Studio folder:

```
cp $V4DIR/omnisxi.so .
```

Failure to observe these steps will result in build errors.

The supplied makefiles require lower case filenames. Use the *lc* utility provided to lower-case your source file names before proceeding:

```
lc -f
```

The lc utility should be placed in /usr/bin or /usr/local/bin

Once the environment has been setup and your files are in place, you can build the Release target by entering:

`rm -rf releaseuni` (or rm -rf release, depending on the value of omunicode) `make Release`

or the Debug target by entering:

`rm -rf debuguni` (or rm –rf debug) `make Debug`

To test the DAM, you should copy the built product into the Omnis/xcomp folder:

cp debuguni/damgenrc.so /usr/local/Omnis Software/studio100/xcomp

You can then startup Omnis using the gdb debugger (if installed) by entering:

```
cd /usr/local/Omnis Software/studio100
gdb omnis
```

Alternatively, you can use your debugger of choice (e.g. Totalview), use the DAM interface's debugging mechanism or seed your code with fprintf() statements to assist with debugging.

Omnis Resource Compiler

This section applies to Mac OSX and Linux platforms only which require a separate build utility in order to compile Omnis resource strings. Omnis resource compilers for Mac OSX and Linux are provided as part of the Component SDK. Microsoft Visual Studio, does not require a separate resource compiler.

If you have already followed the "Getting Started with Generic" section in the External Components tutorial, it is likely that your resource compiler is already setup, in which case this section may be ignored.

Mac OSX

Once downloaded, the Mac resource compiler should be placed in to the /Developer/Tools folder before starting Xcode. When building the component, omnisrc and the name of your .RC file should appear in the detailed build results. As a sanity check, output similar to the following should be displayed inside the *Report Navigator → Build (All Messages):*

```
gXcomp: 1
COCOA VERSION 1.0CountResources('OCTY') = 64
```

Linux

For Linux, the resource compiler should be placed in a folder to which you have execute privileges, for example: /usr/local/omnisrc or ~/omnisrc. Two environment variables are also required:

- OMNISRC – used internally by the resource compiler
- PATH – used by the system to locate the resource compiler

You can test availability of the resource compiler before building your component using the following terminal commands:

```
. ~/setenv (note the use of dot-space-tilde)
omnisrc
```

The Linux omnisrc.ini file requires modification before it can be used by the compiler. Specifically, the Setup→Template attribute should contain the path to the omnisrc.tpl file.

Setup→IncludeDirs should be set to the folder containing Studio core header files. E.g.

```
[Setup]
Template=/home/user/omnisrcdir/omnisrc.tpl
IncludeDirs=/home/user/corefiles
```

If the resource compiler is correctly configured, output similar to the following should be generated when you build the component:

```
Compiling dgenobj.rc
dgenobj.rc successfully compiled.
String objects : 190
Bitmap objects : 1
Cursor objects : 0
Dialog objects : 0
User-Defined data objects : 0
RCDATA objects : 0
gcc -g -fPIC -I/home/user/corefiles -I../damlib -c omnisrc.c -o debuguni/omnisrc.o
```

If the resource compiler fails during a build, there is possibly a problem with one or more missing environment variables, the omnisrc.ini or omnisrc.tpl files are incorrect, the format of one of the .RC files (unexpected carriage returns or missing quotation marks). Ensure also, that **damobj.rc** can be located. Recall that for Linux, this file needs to be copied into the project folder.

Renaming Files

When you are ready to expand on the generic code you should rename the project makefile, source files, target names and auxiliary files appropriate to your intended application.

Historically, Omnis DAMs use an 8.3 filename convention, e.g. DAMORA8, DAMSYBSE & DAMGENRC. Adherence to this convention is nolonger a requirement of modern compilers although you should avoid spaces in filenames.

The C++ sources, header files, resource file and bitmap file should be renamed first. You then need to make any corrections to your project/makefile in order to recognise the renamed files.

Edit your resource file and change the name of the component (1000), the exported session name (2000) and the name of the DAM's entry point (31000). The bitmap filename must match the value of resource string 1. In the source file for the session object, rename the entry point function/message handler from GenericObjProc to reflect the name of your DAM, retaining the ObjProc suffix. This name must match resource string 31000.

Windows

Rename the DGENERIC.DEF file, then edit this file and change the exported entry point name to match the name of your message handler/ObjProc. You will need to edit the Linker→Input→Module Definition File with the new file name

Mac OSX

For all build targets, rename the Packaging→Product Name project setting, retaining the dam... prefix. Rename the four prefix header files, then for each target, edit the Language→Prefix Header project setting with the new filename.

Edit the Info.plist file and change the value of CFBundleExecutable to match the product name.

Edit the Infoplist.strings file using TextEdit and edit the CFBundleName key value with the new DAM name. Perform a Build→Clean All before attempting to build your select target.

Linux

Edit the makefile and change the PROJ_NAME, objects, DEPENDENCIES and RCFILE to correspond with your renamed files, (retaining the .o, .he and .rc filename extensions).

Remove any intermediate files before attempting to build (e.g. rm -rf debuguni).

Implementation Notes

Once you have carried-out any renaming and house-keeping on the generic DAM and you have a working component, you are ready to start expanding the various stub functions to provide the functionality you want.

At this point you may wish to link in any external client libraries your implementation will require, even if you are not calling any of their functions for the time being. This verifies that the libraries are compatible with the compiler and hopefully avoids any nasty surprises later on.

Try to pigeon-hole any API calls provided by the external API into the various functional areas provided by the DAM interface, perhaps allocating them to their appropriate derived methods. Certain API calls will lend themselves sensibly to the DAM's session object; others will be more applicable to the statement object. You should also note any custom properties and/or methods you may need to add to the session or statement objects for functionality not provided by the DAM API. This includes properties which provide additional parameters for pre-defined base class methods, for example; to the $logon() method.

Avoid the temptation to write in API calls and custom properties immediately as this will undoubtedly break the code. Instead, add them as stubs or comments or add them as you come to develop each section of code. Analysing the functions provided by the client API allows you to anticipate any shortcomings in either the DAM library or the client interface which your code will need to overcome. You may also be able to identify API calls which your DAM will not need.

Other points to bear in mind before commencing include:

Is your database server 'relational' (i.e. *rows & columns*) in nature? The DAM API is designed with relational database management systems (RDMS) in mind. Use with non-relational database systems may require careful examination of the requirements of the database versus those of the DAM API.

Does your database support direct execution of SQL statements? (dAllowsDirectExec())

Does your database/clientware support multiple transactions (multiple statement objects)?

Does your clientware support place markers in place of bind variables?

Will your database require support for/conversion of specialised data types?

Does your database support remote procedures/function calls?

Is information about tables/columns/indexes/primary keys readily available for use by the meta-data calls? What (if any) extra information will your meta-data methods need to return via their DamInfoRow columns?

Does your database/clientware support chunking of result data? (dGetNextChunk())

Does your database support deferring of input parameter data? (dSendParameter())

Suggested Order of Implementation

When you start to implement the code for your DAM, you should first concentrate on initializing the session object (the session object constructor). The next step is to open a connection to the database by implementing the dLogon() method.

Once you have a working connection (and at all stages of development), check in Omnis that the session properties are correct and that your DAM is working as expected. You may wish to refer back to the Omnis Programming Manual periodically to ensure that your DAM conforms to the documentation. If you are maintaining both Unicode and non-Unicode targets, you should also check that the $logon() method succeeds using both DAMs (in both versions of Omnis Studio).

From here, you may wish to implement dLogOff() and add one or more session properties which return additional information about the connection. Useful next steps might be to implement error reporting in the session object, which you can test by entering erroneous logon credentials for example. You can also implement dColText() in the session object, which will allow you to test $createnames() and $coltext() in Omnis.

The next major step in development is to implement the statement object; concentrating first on the statement object constructor and dClearStatement(). Give consideration at this point to transaction support; if and how your DAM will manage multiple statement objects and how transactions on each will be isolated. You may wish to go back to the session object and implement dBegin(), dCommit() and dRollback() for manual transaction mode before proceeding further.

In the statement object, you can now implement dPrepare(), dExecute() and dExecDirect() and related methods to issue simple SQL statements to the database. The statement object has its own error handler, so you may wish to implement this next.

Once simple queries are working, you might concentrate on the fetching mechanism; describing and returning result sets to Omnis; i.e. dSetResults(), dDescribeParam(), dFetch() and dConvertParam().

Next, implement the meta-data methods; dTables(), dColumns() and dIndexes() plus their associated fetch methods dTablesGet...(), dColumnsGet...(), dIndexesGet...().

Next, implement input binding and/or inlining of parameters into your SQL text; dBindParameter(), dSetInputBufferValue(), dProcessExecuteParams() and dSendParameter(). Ensure that your bind marker gets substituted for the Omnis bind variables and that dBindParameter() is called either at prepare or execute time as appropriate.

Once the main derived virtual functions have been implemented, you can add additional functionality, such as support for RPC calls ($rpc()) plus any custom functionality noted earlier from the client API.

Naming Conventions Used

The DAM SDK, like the component SDK adheres to certain naming conventions and styles laid down by Omnis programmers. Additionally, this document uses conventions to differentiate between code written in Omnis (Omnis code) and code written in C++:

tqfDAMObject *tqf* is prepended to base class session and statement objects as well as their container objecrs. The historical significance of the letters tqf is unknown.

DAMObject *DAM* is prepended to other object classes provided by the DAM SDK and also to callbacks provided by the Omnis core executable.

cConstant *c* is used to denote enumerations or hard-coded constants. Such constants are used to identify resource strings, property and method IDs as well as implementation-specific constants.

pValue *p* denotes a method parameter, either a static value, a pointer or a value passed by reference.

mAttribute *m* denotes a private, protected or public member of an object class. You will not have direct access to private members of base class objects. Only the derived sub-class will have access to protected base class members.

kConstant *k* denotes either an Omnis constant value as seen via the Omnis catalog (F9) or a constant defined in the component or DAM SDK, usually defined using the #define directive. *k* Constants usually have corresponding values/meanings in Omnis.

dMethodName() *d* denotes a derived pure or non-pure virtual object method. Pure virtual methods defined by the base class must be implemented by your DAM. Implementation of non-pure virtual methods is optional. Note also the use of parentheses which are used to differentiate method names from properties.

$attribute *$* denotes an Omnis object property, visible when your DAM is running inside Omnis Studio.

$method() Denotes an Omnis object method, visible when your DAM is running inside Omnis Studio.

Resources

Omnis DAMs use bespoke resource strings with resource numbers ranging from 1000 to 32000. The DAM API reserves certain resources and ranges of resource IDs for specific uses as described below.

1	Bitmap Resource file name.This is 16x16 pixel (icon sized) image that will be associated with the DAM component. Not strictly relevant for non-visual components. A generic bitmap is supplied.
1000	Component Name (RES_LIB_NAME) The DAM's message handler returns this string when sent the ECM_GETCOMPLIBINFO message. The name typically conforms to "CUSTOMDAM"

2000 to 200x	Advertised Object Names The component library requests the names of any objects provided by the component via the ECM_GETOBJECT message sent to the message handler. 2000 contains the name that will represent the session object provided by your DAM and typically conforms to: “CUSTOMSESS:Custom Session”. Since statement objects may not be instantiated directly, resource 2001 is empty by default. This prevents Omnis developers from creating object variables of subtype *statement object*.
2100 to 5999	Available for use by the implementation (with the exception of 4000-4002) It is advisable to subdivide this range to allow for the following categories: - General strings required by the implementation, e.g. version, timezone, datetime format, server data types & misc. error messages - Columns names assigned to lists, e.g. when returning DamInfo via $tables() & $columns() - DAM-specific constants that you wish to register with the Omnis core - Custom session object methods - Custom session object properties - Custom session object method parameters - Custom statement object methods - Custom statement object properties - Custom statement object method parameters See the template resource file for a suggested implementation.
4000	This must be set to the SQL text required in order to create a database column which accepts null values. If the database creates nullable columns by default, the string can be empty. Example: “NULL”
4001	This must be set to the SQL text required in order to create a database column which does not accept null values. Example: “NOT NULL”
4002	This must be set to the SQL text required in order to specify that a database column is a primary key. If the database does not support primary key columns, the string can be empty. Example: “PRIMARY KEY”
6000 to 6099	Reserved for use by base class session instance methods.
7000 to 7099	Reserved for use by base class session object properties.
8000 to 8099	Reserved for use by base class session method parameters.
9000 to 9199	Reserved for use by base class statement instance methods.
10000 to 10099	Reserved for use by base class statement object properties.
11000 to 11099	Reserved for use by base class statement method parameters.
12000 to 12999	Reserved for use by the base class.
13000 to 30999	Not currently used.
31000	DAM Entry Point (EXTFUNCPROCRCID) The DAM component exports a single entry point (procedure) with a name similar to *customDAMObjProc*. The name of the entry point (aka *Function Proc,* aka *Message Handler*) must be assigned to Resource String 31000 inside your .RC file. The component library uses this string to register the entry point for your component. Windows developers will also note that there is a .DEF file which additionally exports this procedure name
31020	ORFC Version (ORFC_VER_RESID) Required for WebClient components, resource number 31020 is reserved for the ORFC version string, used by the Omnis webserver when dynamically updating components. This takes the form “VER M m” where M represents the component’s major version number and m represents the product’s minor version number. is a place marker for use on Linux platforms. Used by ECOreturnVersion().
32000	Thread Safe Resources (UTSR) Components must assign resource number 32000 to the string “UTSR” in order to use thread-safe resources.

By inspecting the resource file supplied with the base class (DAMOBJ.RC), you may also make use of other gaps in resource ranges. This should be carried out judiciously however as future releases of the base class may result in base class resources conflicting with your own.

Adding Custom Attributes

Adding Constants

Adding constants for use with your DAM involves:

- Adding resource strings containing descriptions of the new constants

- Handling the ECM_GETCONSTNAME message to register the constants

Resource strings for constants should be in the following format:

(See External Component documentation on ECM_GETCONSTNAME for more details)

“[group~]name:num:charvalue:desc” For example:

```
2354 "Transaction Status~kPgSqlTranIdle:0:kPgSqlTranIdle:Transaction status idle"
2355 "kPgSqlTranActive:1:kPgSqlTranActive:Command in progress"
2356 "kPgSqlTranInTran:2:kPgSqlTranInTran:Idle, within a transaction block"
2357 "kPgSqlTranInError:3:kPgSqlTranInError:Idle, within a failed transaction"
2358 "kPgSqlTranUnknown:4:kPgSqlTranUnknown:Transaction status not known"
2359 "Connection Status~kPgSqlConnectionOK:0:kPgSqlConnectionOK:Successful connection"
2360 "kPgSqlConnectionBad:1:kPgSqlConnectionBad:The connection failed"
2361 "kPgSqlConnectionStarted:2:kPgSqlConnectionStarted:Waiting for connection to be made"
```

The resource numbers for the first and last constants to be defined are returned using the ECOreturnConstants() method. Thus, you need to amend the relevant constants each time a constant is added. When the component has loaded, constants returned via the ECM_GETCONSTNAME message can be viewed in the Omnis Catalog (F9).

Adding Properties

Adding a custom property to the session or statement object involves the following:

- Add a resource string describing the new property
- Add a constant to the relevant header file, identifying the resource number
- Amend the ECOproperty structure of and session/statement object to include a description of the new property
- Add support for the property to the dPropertyCanAssign(), dSetProperty() and dGetProperty() derived methods

Property resource strings should be in the following format:

(See External Component documentation on ECM_GETPROPNAME for more details)

“propName:Description” For example:

```
2505 "$datesecdp:The number of decimal places used for SQL Server date columns"
```

Note that all property names start with a dollar character. The description field appears in the Interface Manager and as a tooltip in the Property Inspector. Note also that the resource file groups session properties and statement properties separately.

Your session/statement header file should include a constant for any custom properties. The constant added for each property should be defined with the property’s resource number, for example:

```
const cSessionDateSecDp  =   2505;
```

The ECOproperty structure is used by the ECM_GETPROPNAME message in the DAM’s message handler to register any custom properties defined by the session & statement objects. Each entry in the structure identifies the property’s resource number, its data type and other flags including whether it is read-only.

When attempting to assign a value to a property from Omnis, the base class’ propertySupport() method is called. This in turn calls the derived dPropertyCanAssign() method with the property’s resource number. dPropertyCanAssign() must return qtrue in order to permit assignment, even if the ECOproperty structure did not use EXTD_FLAG_EDITRONLY in its declaration.

Pending the result of dPropertyCanAssign(), dSetProperty() is called, which sets an internal property (usually a private member of the object). dSetProperty() does not need to be implemented for read-only properties.

When Omnis interrogates the value of a property, the base class calls dGetProperty() which uses the supplied resource number to identify the property and returns the value of the appropriate private member.

Implementation Note

It is important to bear in mind that object properties may be interrogated frequently and repeatedly, for instance in situations where the Property Inspector, SQL Browser or Interface Manager are being used. For performance reasons, it is advisable to avoid making client calls to the database or other time-consuming operations when retrieving property values. If necessary, such operations should be performed elsewhere and the results cached via private class members.

Adding Methods

Adding a custom method to the derived session/statement object involves the following:

- Add a resource string with a description of the new method
- Add further resource strings for each method parameter
- Add constants to the relevant header file, identifying the resource numbers
- Amend the ECOmethodEvent structure of the object to include a description of the new method and its parameters
- Append dMethodCall() with the implementation for the custom method

Method resource strings should be in the format required by the ECM_GETMETHODNAME message:

“methodName:Description” For example:

```
omnis 2705 "$serverstatus:Returns information about the database server's current status"
```

Parameters are normally grouped separately in the resource file, for example:

```
5000 "Parameter:The server parameter to be queried" //$parameterstatus
5001 "Fieldlength:The field length to which this mapping applies" //$addenumtype
```

Note that method names also start with a dollar character. The description field appears in the Interface Manager and as a tooltip in the Property Inspector.

Your session/statement header file should include a constant group for any custom methods. The constant added for each method should be defined with the method’s resource number, for example:

```
const cSessionServerStatus = 2705;

const cSessionParamParam = 5000,
      cSessionParamFieldLen = 5001;
```

When the DAM’s message handler receives an ECM_GETMETHODNAME message, it returns the relevant ECOMethodEvent structures for the specified object.

Each ECOMethodEvent structure defines one method’s resource number, return type, number of parameters plus any extended flags. If the method requires parameters, these are also returned via a pointer to an ECOparam struct. As with ECOproperty structures, ECOMethodEvent and ECOparam structures are normally located near the top of the relevant source file, after the #include section.

When the DAM’s message handler receives an ECM_METHODCALL message, it calls the base class’ methodCall() method, which in turn calls the derived dMethodCall() handler.

Except where the derived methods have a trivial implementation, it is normally better practice to implement each custom method as a separate C++ function and call in to these. This keeps your dMethodCall() handler compact and easier to maintain.

Return Values

Please note that the return value of the dMethodCall() handler does not constitute the return value of the custom method; it indicates whether or not the implementation successfully handled the requested method.

To return a value from a custom method, the dMethodCall() *rtnVal* and *hasRtnVal* parameters are used. It is recommended to adopt a consistent approach towards custom method return values; i.e. by using a Boolean value to indicate the success or failure of any given method, your method handler might look similar to this:

```
qbool tqfODBCDAMObj::dMethodCall(EXTCompInfo* pEci, EXTfldval &rtnVal, qbool &hasRtnVal)
{
  qlong funcId = ECOgetId(pEci);
  qbool status = qfalse, rtnStatus = qtrue;
  switch (funcId)
  {
    //…
```

```
    case cODBCSessionSetConfigOption:
    {
      status = setConfigOption( pEci );
      break;
    }
    default:
    {
      rtnStatus = qfalse;
      break;
    }
  }
  hasRtnVal = qtrue;
  rtnVal.setBool(status? preBoolTrue : preBoolFalse); //indicates whether (funcId) succeeded
  return rtnStatus; //indicates whether dmethodCall() succeeded
}
```

Adding Method Parameters

Resource strings for method parameters are also returned via the ECM_GETMETHODNAME message (as part of the ECOMethodEvent structure) and take the form: “paramName:Description”

For example:

```
8107 "Option:Constant specifying the required attribute"
8108 "Attribute:Character value containing attribute"
```

The derived object should define an ECOparam structure for each method where one or more parameters are required, for example:

```
ECOparam ODBCObjParamSetOption[] =
{
  8107, fftConstant, 0, 0, //attrib type
  8108, fftCharacter, EXTD_FLAG_PARAMOPT, 0 //attrib value
};
```

Each entry in the structure defines a parameter’s resource number, its datatype and other flags, for example whether the method needs to write back to the parameter. ECOparam is discussed in greater detail in the External Component documentation.

```
ECOmethodEvent ODBCObjFuncs[] =
{
  DAM_OBJ_FUNCS,
  cODBCSessionSetConfigOption,cODBCSessionSetConfigOption, fftBoolean, 2, ODBCObjParamSetOption, 0, 0
};
```

The above example registers a custom method containing 2 parameters and with a return type of Boolean.

DAM Type Table

The base class DAMTypeTable object is used to store information relating to each Omnis data type. The type table contains one entry for each *eTypeEntry* enum: kTypeCharacter, kTypeBoolean, kTypeDateTime, kTypeDate, kTypeTime, kTypeSequence, kTypeNumber, kTypeShortNumber, kTypeFloat, kTypeInteger, kTypeShortInteger, kTypePicture, kTypeBinary, kTypeList, kTypeRow, kTypeObject, kTypeItemRef and kTypeUnknown.

The DAMTypeTable class provides conversion between Omnis data types and *C-Types*, which are database-independent data types. It uses a private *map2TypeTable()* method to derive the type table index from any given Omnis data type and sub-type.

Each table entry contains a DAMMap structure which describes:

- sendType – how variables of this type should be sent to the database. One of kDefault, kInline, kConvert or kDeferInline.
- cType – the implementation-independent datatype.

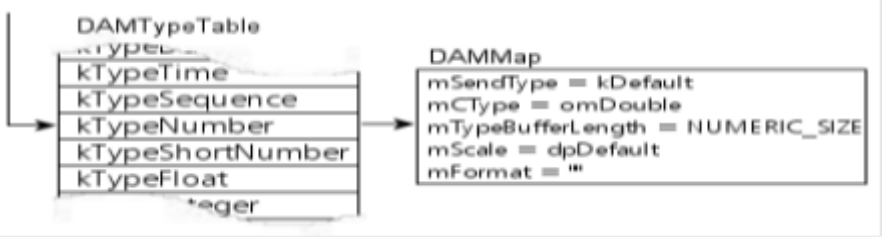

Figure 7:

- bufferLength – the default buffer length that parameters of this type will require.
- scale – the default numeric scale (decimal places) used for this type.
- format – the date/time format string to be used when formatting datetime values.

The base class provides default values for the C-Type, length, scale, sendType and format attributes of each entry. The DAMTypeTable is designed to be overridden by the implementation, which is then free to make any modifications to this table, for example during construction of the derived type table object, or inside the derived session object's construct method or at any point up until the connection has been established (dLogon()). The contents of this table then usually remain constant for the life of the session.

The DAM type table is used by the base statement object's setupInputVars() method to establish the default C-Type, scale, sendType and buffer lengths for each input bind parameter (mInputParams).

The DAM type table is also used by DAMParam::setBufferValue() when assigning datetime values, it uses the type entry's format attribute to format date and time values correctly.

enum eTypeEntry is described further in the Constants and Enumerations section.

The DAMTypeTable class public methods are described in the Support Classes section.

Omnis Object Methods

This section summarises the main methods provided by the DAM interface and shows you the protected virtual functions in the context of the Omnis methods which call them. The order in which various sub-methods are called is also shown. Calls to derived methods are shown in bold and there are brief explanatory notes where applicable. For a fuller description of the base class methods used, please refer to the Private Methods section for the relevant object.

For descriptions explaining the implementation requirements for each of the derived methods, please refer to the Protected Virtual Methods section for the object.

Key:	
□method()	Where □ appears before a method name, this indicates that the method is called inside a loop, multiple times
method()□	Where □ appears after the method name, this indicates that the method contains a loop
←DAMcallback()	Indicates a callback to the Omnis Studio core executable
dMethod()	Indicates a derived virtual method; a call to your DAM implementation

Session Object

$logon()

```
tqfDAMbaseObj::callLogon()
    logon()
        logoff()
    ←DAMregisterSession()
    dLogon()
    dSetBindMarker()
    ←DAMunregisterSession()
```

logoff() is only called if the session is currently logged on (mState=kSessionLoggedOn)

DAMunregisterSession() is called if dLogon() or dSetBindMarker() fails.

$logoff()

```
tqfDAMbaseObj::logoff()
    dropStatements()
        mStatements→ drop()→dDropStatement()
    dLogoff()
    ←DAMunregisterSession()
```

drop() causes dDropStatement() to be called for each derived statement object

$clear()

Clears a session object.

```
tqfDAMbaseObj::clear()
    getOmnisSeparators()
    clearStatements()
        mStatements→ clearAll()
```

getOmnisSeparators() evaluates $root.$prefs.$separators() each time a session object is cleared in case Omnis decimal and thousand separators have changed in the interim.

$makeschema()

Makes a schema class.

```
objectinst::makeSchema()
    newStatement()
    statementobj→doSQL(kSOBJcolumns)
        dColumns()
    statementobj→doSQL(kSOBJfetch)
        dFetch()
        dColumnsGet…()
```

makeSchema() is implemented by the Omnis Studio core. It invokes a $columns() call on an internal statement object and also fetches the complete result set.

$newstatement() $newstatementref()

Creates a statement object.

```
tqfDAMbaseObj::newStatement()
    dGetStatementId()
    dNewStatement()
```

If dNewStatement() is successful, newStatement() assigns a statement name to the newly created object, assigns the object's session pointer and copies the session object's mStripSpaces property into the statement object.

If the user calls $newstatement(), newStatement() returns the object instance. If the user calls $newstatementref(), newStatement() returns a pointer to the newly created statement object.

$begin()

Begin a transaction.

```
tqfDAMbaseObj::begin()
    dUsesAutoBegin()
    dBegin()
```

The session must be logged on and the transaction mode must be set to kTranManual in order for begin() to succeed. mAllowsTransactions must be qtrue and dUsesAutoBegin() must return qfalse in order for dBegin() to be called.

$commit()

Commit a transaction.

```
tqfDAMbaseObj::commit()
    dCommit()
```

The session must be logged on and the transaction mode must be set to kTranManual in order for commit() to succeed. mAllowsTransactions must be qtrue in order for dCommit() to be called.

$rollback

Rollback a transaction.

```
tqfDAMbaseObj::rollback()
    dRollback()
```

The session must be logged on and the transaction mode must be set to kTranManual in order for rollback() to succeed. mAllowsTransactions must be qtrue in order for dRollback() to be called.

$nextnativeerror()

Get the next native error.

```
tqfDAMbaseObj::setNativeError()
  dGetNativeError()
```

This call fails if the session object's mErrorInfo.getNativeErrorPending() returns qfalse.

If dGetNativeError() succeeds, the mErrorInfo native error code and native error text attributes are set from the returned values. mErrorPending on the error object is also set to qtrue. *Native* errors are those generated by your database API.

$rpcdefine()

Define parameters for remote procedure call.

```
tqfDAMbaseObj::rpcDefine()
    findRpc()
    DAMrpcDefn::DAMrpcDefn()
```

This call fails if parameters 1 or 2 are missing or if parameter 2 is not a list variable.

If a case-sensitive search on the RPC name succeeds, the existing definition is modified. Otherwise, a new DAMrpcDefn object is created. rpcDefine() then builds a list of parameters as supplied

$coltext()

Get SQL text for a table column.

```
tqfDAMbaseObj::coltext()
    dColText()
```

If coltext() is called with a predefined EXTparamTypeInfo structure, this is passed to dColText().

Otherwise the ffttype of the parameter is determined from the EXTparamInfo. If the parameter has a binary type (includes lists, item refs, pictures and objects), the length supplied to dColtext() is set to mBlobSize

$createnames()

Get createnames clause for a CREATE TABLE statement.

```
tqfDAMbaseObj::createnames()
    dColText()
```

$createnames() calls dColText() for each column in the supplied list. For binary columns, the length supplied to dColText() is set to mBlobsize.

$insertnames()

Get insertnames clause for an INSERT statement.

```
tqfDAMbaseObj::insertnames()
    selectnames()
```

insertnames() calls selectnames() to get a list of column names. The VALUES clause is generated based on the column names of the supplied list/row variable and column names. No call to the derived session object is required.

$selectnames()

Get selectnames clause for a SELECT statement.

```
tqfDAMbaseObj::selectnames()
```

selectnames() generates a comma-separated list based on the column names of the supplied list/row. No call to the derived session object is required.

$updatenames()

Get updatenames clause for an UPDATE statement.

```
tqfDAMbaseObj::updatenames()
```

updatenames() generates the SET clause for an UPDATE statement based on the column definitions of the supplied list/row. No call to the derived session object is required.

$wherenames()

Get wherenames clause for a SQL statement.

```
tqfDAMbaseObj::wherenames()
```

wherenames() generates the WHERE clause for a SQL statement based on the supplied parameters. No call to the derived session object is required.

Statement Object

$clear()

Clearing a statement object.

```
tqfDAMStatementObj::clearAll()
    dClearStatement()
    dClearParams(mInputParams)
    dClearParams(mOutputParams)
    clear()
      clearResults()
      mInputParams.dropParams()
      mOutputParams.dropParams()
```

clearResults() causes mRowsFetched , mBatchRow, mRowsAffected , mColumnCount , mRowCount, mNoRows and mResultsPending to be reset to default values. clear() also drops input and output parameters, resets mRpcReturnValue, sets mState to kStateClear and frees mTableRef if it is owned by the object.

drop()

Drop a statement object.

```
tqfDAMStatementObj::drop()
    clearAll()
        dDropStatement()
```

Dropping a statement object is carried out implicitly by the base class. There is no Omnis method to invoke this directly. drop() is called for each statement object when a session object is about to be destructed.

$columns()

Describing table columns.

```
tqfDAMStatementObj::columns()
    prepareForMetaDataMethod()
        clearAll()
    dColumns()
    readyForFetch()
        dSetResults()
    describe()
         dDescribeParam()
        dProcessOutputColumns()
```

prepareForMetaDataMethod() ensures that the statement is in a suitable state to execute a new SQL statement. This involves checking for errors and clearing the statement. dColumns() is responsible for executing the SQL statement required to describe the table. readyForFetch() calls dSetResults() and commits the transaction (in kTranAutomatic only) on success. describe() calls dDescribeParam() for each column of the pending result set. dProcessOutputColumns() allows the derived statement object to make any additional adjustments, e.g. to mOutputParams. columns() stops executing and returns qfalse if any of these methods fail.

$execdirect()

Direct execution of a SQL statement.

```
tqfDAMStatementObj::execdirect()
    mSession→allowsDirectExec()
        dAllowsDirectExec()
    execdirect()    or prepareExec()
    describe()
```

If dAllowsDirectExec() returns qtrue, execdirect() is called. Otherwise prepareExec() is called.

```
tqfDAMStatementObj::execdirect()
    clearAll()
        dClearStatement()
        dClearParams(mInputParams)
        dClearParams(mOutputParams)
    setStatementText()
        ←DAMPrepareBindVariables()
        mSession→charMapOut()
            dAllowsCharConversion(kCharMapOut)
    setupInputVars(kBindExecute)
    mSession→getTypeTable()
    DAMParam::setBuffers(1)
    dProcessPrepareParams(mInputParams)
    ←DAMgetBindVariableValues()
    setBufferValues()
         →DAMParam::setBuffervalue()
             charMapOut()
             dAllowsCharConversion(kCharMapOut)
          dSetInputBufferValue()   (only if sendType=kConvert)
        dProcessExecuteParams(mInputParams)
    dProcessInlines(mInputParams)
    bindInputBuffers(kBindExecute)
         dBindParameter()
    dExecDirect()
    sendInputValues()
         dSendParameter()
    readyForFetch()
        dSetResults()
```

getTypeTable() invokes the derived session object's dGetTypeTable() method.

```
tqfDAMStatementObj::prepareExec()
    prepare()    ( see $prepare() )
    execute()    ( see $execute() )

tqfDAMStatementObj::describe()
     dDescribeParam()
    dProcessOutputColumns()
```

Output character mapping of the SQL statement is only performed if $unicode is kFalse.

setupInputVars() initialises the buffer type, subtype, C-type, scale, sendType, offset, bindType, parameterType and field length for each input paramter. setBuffers() allocates the data buffer and data length buffer for each parameter. The interface does not support array binding of input variables. Hence setBuffers() is passed 1 for the batch size parameter.

After calling execdirect() or prepareExec(); describe() is called. This popluates mOutputParams with descriptions of the result set columns. dSetResults() must set mColumnCount to the number of columns in the result set.

$prepare()

Prepare a SQL Statement.

```
tqfDAMStatementObj::prepare()  (Prepare SQL statement)
    clearAll()
        dClearStatement()
        dClearParams(mInputParams)
        dClearParams(mOutputParams)
    setStatementText()
        ←DAMPrepareBindVariables()
        mSession→charMapOut()
            dAllowsCharConversion(kCharMapOut)
    setupInputVars(kBindPrepare)
        mSession→getTypeTable()
        DAMParam::setBuffers(1)
    dProcessPrepareParams(mInputParams)
    dPrepare()
    bindInputBuffers(kBindPrepare)
         dBindParameter()
```

getTypeTable() invokes the derived session object's dGetTypeTable() method.

$execute()

Execute a SQL statement.

```
tqfDAMStatementObj::execute()
    ←DAMgetBindVariableValues()
    setBufferValues()
         →DAMParam::setBuffervalue()
             charMapOut()
             dAllowsCharConversion(kCharMapOut)
          dSetInputBufferValue() (only if sendType=kConvert)
        dProcessExecuteParams(mInputParams)
    dProcessInlines()
    bindInputBuffers(kBindExecute)
         dBindParameter()
    dExecute()
    sendInputValues()
         dSendParameter()
    readyForFetch()
        dSetResults()
```

$fetch()

Fetching results.

```
tqfDAMStatementObj::fetch()
    dPreFetch()
    defineList()
    bindOutputBuffers()
        mOutputParms.setBuffers()
         dBindColulmn()
    fetchRows()
         →dFetch()
             dCanAddRow()
          addRow()
              mTableRef→insertRow()
              setOmnisColumnData()*
    dPostFetch()
    dMoreResults()
```

```
    dSetResults()           (only called if fetchStatus returns kFetchFinished)
    describe()
```

*Called only if mAddRowMode is eDAMaddRowNormal. For other modes (metadata), addTableRow(), addColumnRow(), addIndexRow(), addRpcProcedureRow() or addRpcParameterRow() is called instead.

fetch() attempts to describe another result set when fetchRows() returns kFetchFinished (or when dPostFetch() augments fetchStatus).

```
tqfDAMStatementObj::setOmnisColumnData()     (Set the contents of an Omnis field value)

         →dGetNextChunk()
          DAMParam::writeNextChunk() (until chunkState=kChunkFinished)
        mSession→convertCharacterDataFromEncoding() (character data only)
        mSession→charMapIn() (non-Unicode character data only)
            dAllowsCharConversion(kCharMapIn)

OR

        dConvertParam()
        mSession→charMapIn() (non-Unicode character data only)
            dAllowsCharConversion(kCharMapIn)
        DAMParam::setOmnisVal()
        setFldVal() OR convFldVal() (only called if paramLen != DAM_PARAM_IS_NULL)
            mSession→convertCharacterDataFromEncoding()*
```

If pParam→getChunk() is qtrue, setOmnisColumnData() fetches all data chunks and writes them to the parameter's chunk handle. It then (conditionally) converts and maps character data.

For non-chunked data, dConvertParam() is called, followed by setOmnisVal(), which either sets an Omnis field value from the contents of the parameter buffer or converts the data where the data type of the parameter does not match the data type of the field value. If the parameter value is NULL, setOmnisVal() sets the field value to NULL implicitly.

*setFldVal() calls convertCharacterDataFromEncoding() when the parameter's C-Type=omChar.

```
tqfDAMStatementObj::addTableRow()  (Extract one row of a $tables() result set)
    dTablesGetOwner()
    dTablesGetTableName()
    dTablesGetTableType()
    dTablesGetDesription()
    dTablesGetDamInfo()

tqfDAMStatementObj::addColumnRow() (Extract one row of a $columns() result set)
    dColumnsGetDatabaseOrCatalog()
    dColumnsGetOwner()
    dColumnsGetColumnName()
    dColumnsGetOmnisDataTypeText()
    addOmnisTypeAndSubType()
        dColumnsGetOmnisDataType()
        dColumnsGetOmnisDataSubType()
    dColumnsGetSqlDataType()
    dColumnsGetLength()
    dColumnsGetScale()
    dColumnsGetNull()
    dColumnsGetIndex()
    dColumnsGetPrimaryKey()
    dColumnsGetDescription()
    dColumnsGetDamInfo()

tqfDAMStatementObj::addIndexRow()     (Extract one row of a $indexes() result set)
```

```
    dIndexesGetDatabaseOrCatalog()
    dIndexesGetOwner()
    dIndexesGetColumnName()
    dIndexesGetIndexName()
    dIndexesGetUnique()
    dIndexesGetColumnPosition()
    dIndexesGetDamInfo()

tqfDAMStatementObj::addRpcProcedureRow()   (Extract one row of a $rpcprocedures() result set)
    dRpcProceduresGetDatabaseOrCatalog()
    dRpcProceduresGetOwner()
    dRpcProceduresGetProcedureName()
    dRpcProceduresGetDamInfo()

tqfDAMStatementObj::addRpcParameterRow()  (Extract one row of a $rpcparameters() result set)
    dRpcParametersGetOmnisDataTypeText()
    dRpcParametersGetDatabaseOrCatalog()
    dRpcParametersGetParameterName()
    dRpcParametersGetSqlDataType()
    dRpcParametersGetScale()
    dRpcParametersGetLength()
    dRpcParametersGetPassType()
    addOmnisTypeAndSubType()
        dRpcParametersGetOmnisDataType()
        dRpcParametersGetOmnisDataSubType()
    dRpcParametersGetDamInfo()
```

$fetchinto()

Fetching directly into Omnis variables.

```
tqfDAMStatementObj::callFetchInto()
    clear_mTableRef()
     mTableRef→addCol()
    fetch()
    clear_mTableRef()
```

callFetchInto() calls fetch() with rowCount=1. Data is then copied from the mTableRef columns into the supplied Omnis fields. mTableRef is subsequently released.

$fetchtofile()

Fetching data into a file.

```
tqfDAMStatementObj::callFetchToFile()
    clear_mTableRef()
    fetch()
    writeListToFile()
    clear_mTableRef()
```

callFetchToFile() calls fetch() as per a normal fetch. The contents of mTableRef are then written to an external file. mTableRef is subsequently released.

$indexes()

Describe table Indexes.

```
tqfDAMStatementObj::indexes()
    prepareForMetaDataMethod()
        clearAll()
    dIndexes()
    readyForFetch(eDAMaddRowIndexes)
    describe()
```

$nextnativeerror()

Get next native error.

```
nextnativeerror()
    mErrorInfo.getNativeErrorPending()
    setNativeError()
        dGetNativeError()
```

$nextnativeerror() invokes setNativeError() if an error is pending on mErrorInfo. Otherwise setError(kDAMNoPendingError) is called.

$results()

Describe columns of a result set.

```
tqfDAMStatementObj::results()
defineMetaDataList()
     →dResults()
          ←DAMgetDataTypeText()
          dGetColumnInfoForResults()
      addResultsRow()
```

dResults() and dGetColumnInfoForResults() are called once for each column in the pending result set.

$rpc()

Call a remote procedure.

```
tqfDAMStatementObj::rpc()
    clearAll()
    dRpc()
    readyForFetch()
    describe()
```

$rpcparameters()

Describe procedure parameters.

```
tqfDAMStatementObj::rpcParameters()
    prepareForMetaDataMethod()
        clearAll()
    dRpcParameters()
    readyForFetch()
    describe()
```

$rpcprocedures()

List remote procedures.

```
tqfDAMStatementObj::rpcProcedures()
    prepareForMetaDataMethod()
        clearAll()
    dRpcProcedures()
    readyForFetch()
    describe()
```

$tables()

List server tables/views.

```
tqfDAMStatementObj::tables()
    prepareForMetaDataMethod()
        clearAll()
    dTables()
    readyForFetch()
    describe()
```

StatementWorker Object

$init()

Initialise statement worker object.

```
StatementWorker::methodInit()
    mDelegate→init()
        clearParams()
        clearResultRow()
        mErrorInfo.clearError()
        dInit()
```

$run()

Run statement worker on main thread.

```
StatementWorker::methodRun()
    mDelegate→run()
        clearResultRow()
        dRun()
    mDelegate→pushWorkerCallbackFromRunMethod()
```

$start()

Run statement worker on background thread.

```
StatementWorker::methodStart()
    mDelegate->start()
        clearResultRow()
        ECOcreateThread()→

→mDelegate→run()
        clearResultRow()
        dRun()
   mDelegate→pushWorkerCallbackFromRunMethod()
```

$cancel()

Cancel statement on background thread.

```
StatementWorker::methodCancel()
    mDelegate->cancel()
        dCancel()
```

$sessionref()

```
StatementWorker::methodSessionRef()  Return an object reference to the internal session object
    mDelegate->getSessionObj() (session)
    mDelegate->damGetSessionObjId()
    session->addRef()
```

methodSessionRef() makes no call to the derived worker object.

Base Classes

tqfDAMbaseObj

The DAM base object controls the session environment from which statement objects are created. It contains attributes and methods which are common to a given connection. This includes the logon parameters, the transaction mode and any default statement properties such as the size of large object data chunks. From Omnis, an instance of a new object is created either explicitly via the $new() method or wherever an object variable of subtype *MyDAMObj* it is first used. This causes the object to initialise (construct) with default property values. When the object goes out of scope or is closed explicitly then depending on the session state any statements are closed and the connection is logged off. Internally, tqfDAMbaseObj encapsulates a database session or connection handle. Its purpose is to be sub-classed for use by the DAM implementation.

Protected Members

Type	Name	Description
DAMErrorInfo	mErrorInfo	Stores error information for the session object
eTranMode	mTransactionMode	The transaction mode associated with the session. It can be one of kTranAutomatic, kTranManual or kTranServer. See Constants for further information.
qlong	mAllowsTransactions	Should be set to true if the database supports manual transactions.
str80	mStatementBindMarker	The symbol expected by the database in place of bind variables, e.g. "?"
str255	mSessionName	The name of the session object used by the application. See $logon()
qlong	mDebugLevel	The level of internal debugging information that should be generated. See $debuglevel
qbool	mIsUnicode	qtrue if the DAM is operating in Unicode mode. See $unicode
preconst	mCodePage	The ANSI codepage used to interpret non-Unicode character data. See $codepage
qbool	mFetch64BitInts	Enables/disables fetching of 64-bit integer values as numbers
qlong64	mBytesReceived	The number of bytes received through session object since logon
qlong64	mBytesSent	The number of bytes sent through session object since logon
qlong	mDebugSize	The maximum size of debug file before truncation occurs

Private Methods

Although you may not call private methods directly, these are referred to in the Omnis Methods section and in the damobj.he header file so are described here in overview.

tqfDAMbaseObj::callLogon()
qbool tqfDAMbaseObj::callLogon(EXTCompInfo* pEci)

callLogon() is invoked by calling $logon() from Omnis. It validates the supplied parameters then calls the logon() public method, returning qtrue if the logon succeeds.

tqfDAMbaseObj::addBindVariable()
qbool tqfDAMbaseObj::addBindVariable(str255 &pRowName,str255 &pColName, EXTfldval &pRtnBindText)

addBindVariable() is called from the insertnames(), updatenames(), wherenames() public methods. Its function is to create a string in the following format: @[pRowName.pColName] which is written into pRtnBindText.

tqfDAMbaseObj::dropStatements()
void tqfDAMbaseObj::dropStatements()

dropStatements() calls drop() on each object in the session object's linked-list of statement objects. It is used by the logoff() public method.

tqfDAMbaseObj::commitClearStatements()
void tqfDAMbaseObj::commitClearStatements()

commitClearStatements() is by during commit() ($commit()). If dGetCommitMode() returns kCommitDelete; tranClearStatements() is called. If the commit mode is kCommitClose- tranCloseStatements() is called. For kCommitPreserve, no action is performed.

tqfDAMbaseObj::rollbackClearStatements()
void tqfDAMbaseObj::rollbackClearStatements()

rollbackClearStatements() is called by rollback() ($rollback()). If dGetRollbackMode() returns kRollbackDelete- tranClearStatements() is called. If the commit mode is kRollbackClose- tranCloseStatements() is called. For kRollbackPreserve, no action is performed.

tqfDAMbaseObj::tranCloseStatements()
void tqfDAMbaseObj::tranCloseStatements()

tranCloseStatements() calls close() on each object in the session object's linked-list of statements (mStatements). Invoked indirectly via $commit() and $rollback() in Omnis.

tqfDAMbaseObj::tranClearStatements()
void tqfDAMbaseObj::tranClearStatements()

tranClearStatements() calls clearTran() on each object in the session object's linked-list of statements (mStatements). Invoked indirectly via $commit() and $rollback() in Omnis.

tqfDAMbaseObj::clearError()
void tqfDAMbaseObj::clearError()

clearError() calls mErrorInfo.clearError() followed by dClearNativeError(). It is invoked by methodCall() and propertySupport() (ECM_SETPROPERTY).

tqfDAMbaseObj::setMapTable()
qbool tqfDAMbaseObj::setMapTable(EXTfldval &pFldVal)

setMapTable() is invoked by calling $maptable() from Omnis. setMapTable() terminates if mUnicode=qtrue. Otherwise, two new DAMCharMapTable objects are allocated and the supplied filename (plus either a *.IN* or *.OUT* suffix) is assigned to each object. If there are no errors, mMapIn and mMapOut are freed (if they are already allocated) then assigned to the newly created objects.

tqfDAMbaseObj::setCharMap()
qbool tqfDAMbaseObj::setCharMap(qlong pCharMap)

setCharMap() is invoked by assigning a value to the $charmap property in Omnis and sets mCharMap based on pCharMap. setCharMap() fails if mIsUnicode=qtrue.

tqfDAMbaseObj::rpcDefine()
qbool tqfDAMbaseObj::rpcDefine(EXTCompInfo *pEci)

rpcDefine() is invoked by calling $rpcdefine() from Omnis. This method receives two parameters. The first is the name of the RPC to define, and the second is the list defining its parameters. This list requires at least 4 columns, and these 4 columns must contain information such as that returned as the result set of $rpcparameters(). rpcDefine() terminates if either of these parameters cannot be validated. Otherwise, findRpc() is called to determine whether a definition for the RPC already exists, in which case it is replaced. If the RPC is not found, a new DAMrpcDefn is created and assigned to the linked list if RPC definitions (mRpcs). Next, for each parameter mapOmCons2fftType() is called and the results stored in the RPC definition's parameter array.

tqfDAMbaseObj::needsBlobLength()
qbool tqfDAMbaseObj::needsBlobLength(ffttype pOmnisType)

needsblobLength() is called by coltext() and createnames() to determine whether mBlobsize should be substituted for the column length. It returns qtrue if the supplied ffttype is fftBinary, fftList, fftRow, fftObject, fftPicture or fftItemref, qfalse otherwise.

tqfDAMbaseObj::getLibStripSpaces()
qbool tqfDAMbaseObj::getLibStripSpaces(locptype *pLocp)

getLibStripSpaces() is called from the session object constructor. It evaluates the notation string "$clib.$prefs.$sqlstripspaces" (read from DAMOBJ.RC) and is used to set mStripSpaces, from which all spawned statement objects take their default values. qfalse is returned if the notation cannot be evaluated.

tqfDAMbaseObj::mapOmCons2fftType()
void tqfDAMbaseObj::mapOmCons2fftType(qlong pConsType, qlong pConsSubType, ffttype &pParmType, qshort &pSubType)

mapOmCons2fftType() is called by rpcDefine() and is used to convert the data type and sub-type constants found in the parameter list into ffttypes and sub-types. Affects lists, rows and datetime types.

tqfDAMbaseObj::getErrorText()
qbool tqfDAMbaseObj::getErrorText(str255 &pErrorText)

getErrorText() is invoked when the object's $errortext property is inspected. It reads an internal error message from the Omnis core resources. The RESgetOmnisDAT() callback is used to obtain a handle on the core's resource strings. The statement object's mErrorInfo→getErrorCode() is used to derive the resource number for the error text.

tqfDAMbaseObj::endDebug()
qbool tqfDAMbaseObj::endDebug()

endDebug() is invoked from Omnis by assigning $debugfile to an empty string. Otherwise, endDebug() is called when the session object is destructed. endDebug() closes the debug file (mDebugFilePtr) if open and frees the debug text buffer (mDebugBuffer)

tqfDAMbaseObj::startDebug()
qbool tqfDAMbaseObj::startDebug(str255 &pFileName)

startDebug() is invoked from Omnis when a non-empty string is assigned to $debugfile. It first calls endDebug() to close any existing debug file before attempting to open the specified filename, or the standard error device (stderr). mDebugBuffer is then allocated, into which debug messages will be cached before being written to mDebugFilePtr. See debugMsg()

tqfDAMbaseObj::convertToCharacterSet()
qbool tqfDAMbaseObj::convertToCharacterSet(csettype pCharSet, qbyte *pData, qlong pDataLen, eCharMapMode pDir)

convertToCharacterSet() is called from charMapOut() and charMapIn() when mCharMap=kSessionCharMapTable or kSessionCharMapOmnis and only when dAllowsConversion(pDir) permits character set conversion.

convertToCharacterSet() uses one of two static mapping tables; one which maps MacRoman extended characters to ISO8859/Latin1 characters (mactowin), and another which maps Latin1 extended characters to Mac Roman characters (wintomac). When output character mapping, convertToCharacterSet() is called before any custom character mapping. When input character mapping, this is called after any custom character mapping. For Macintosh targets, when pCharSet is csetOdata, no conversion is performed.

8-bit character mapping (and hence this function) is not called if mUnicode=qtrue.

tqfDAMbaseObj::insertQuotes()
void tqfDAMbaseObj**::**insertQuotes(str255 &pString)

insertQuotes() is intended to be supplied with a string conforming to *database.owner.colname* and converts this into *"database"."owner"."colna* i.e. it inserts quotes around dot-delimited identifiers. It is called from selectnames() and wherenames() and only when mQuotedIdentifier ($quotedidentifier) is qtrue (kTrue).

Public Methods

tqfDAMbaseObj::tqfDAMbaseObj()
tqfDAMbaseObj::tqfDAMbaseObj(EXTCompInfo *pEci)

The DAM base class constructor initialises the members of the base class.

- **pEci** – The external component information class that contains information required to construct the base object.

tqfDAMbaseObj::~tqfDAMbaseObj()
tqfDAMbaseObj::~tqfDAMbaseObj()

The base class destructor logs off from any session and removes(unlinks) any associated statement objects. This does NOT destroy any statement objects.

tqfDAMbaseObj::addRef()
void tqfDAMbaseObj::addRef()

This method increments the number of references to this object instance.

tqfDAMbaseObj::releaseRef()
void tqfDAMbaseObj::releaseRef()

This decrements the reference count of this object instance. If the reference count reaches zero then the base class is destroyed.

tqfDAMbaseObj:: getOmnisSerialNumber()
qbool getOmnisSerialNumber(str80 &pSerialNumber)

Provides implementation with access to the Omnis serial number. Returns qtrue on success.

- **pSerialNumber** – string variable to be populated with the serial number

tqfDAMbaseObj:: methodCall()
qbool tqfDAMbaseObj::methodCall(EXTCompInfo* pEci)

The external component library calls methodCall() every time a session object function is invoked within Omnis Studio. The pEci parameter will contain the information about the invoked DAM function and the base class will call the appropriate method based on this information. Returns qtrue on success.

- **pEci** - The external component information class that contains information regarding the method to be called.

tqfDAMbaseObj:: propertySupport()
qlong tqfDAMbaseObj::propertySupport(LPARAM pMessage, WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

This method supports the session object properties. It will determine whether a property is read only as well as read and write the value of the property if permitted. Returns qtrue on success.

- **pMessage** - The type of property request. This can be ECM_PROPERTYCANASSIGN, ECM_SETPROPERTY or ECM_GETPROPERTY
- **wParam** - This is set to ECM_WPARAM_PROPBUTTON when pMessage is ECM_SETPROPERTY and if the Property Manager popup button was pressed to set the property. For example, a file name property may wish to use a file open dialog if the popup button was pressed. Otherwise it is not used.
- **lParam** - Not used.
- **eci** - The external component information class that contains information regarding the property to be processed.

tqfDAMbaseObj:: logon()
qbool tqfDAMbaseObj::logon(EXTCompInfo *pEci, str255& pHostName, str255& pUserName, str255& pPassWord, str255 &pSessionName)

This method registers the session with the $sessions group if necessary and then calls the database specific logon implementation. If this is successful then the connection properties are updated in the session object. Returns qtrue on success.

- **pEci** - The external component information class that contains information regarding logon method.

- **pHostName** – The hostname required to connect to the database.
- **pUserName** – The username for the database connection
- **pPassWord** – The password for the database connection.
- **pSessionName** – An optional parameter that if used will register the session as pSessionName in the $sessions group.

tqfDAMbaseObj::logoff()
qbool tqfDAMbaseObj::logoff()

This method drops any statements associated with the session before unregistering with $sessions and finally dropping the database connection.

tqfDAMbaseObj::coltext()
qbool tqfDAMbaseObj::coltext(EXTParamInfo *pParam, str255 &pRtnString, EXTparamTypeInfo *pInfo)

This method returns a text string that contains the equivalent SQL Data type text for an Omnis variable or column of a list. Returns qtrue on success.

- **pParam** – Contains information about the Omnis variable passed as a parameter to $coltext()
- **pRtnString** – An out parameter that will contain the SQL datatype text for the Omnis variable.
- **pInfo** – A structure that contains information regarding the column of a list. Used by the core during list.$createnames() methods.

tqfDAMbaseObj::selectnames()
qbool tqfDAMbaseObj::selectnames(EXTParamInfo* pTableDefParam,EXTParamInfo* pTableNameParam,EXTfldval &pRtnString)

This method returns a text string defining the columns to be used in a SQL SELECT statement.

- **pTableDefParam**- A parameter info structure containing a list or row variable containing the row definition.
- **pTableNameParam** – A parameter info structure containing the name of the table.
- **pRtnString** - An out parameter that will contain a text string defining the columns to be used in a SQL SELECT statement.

tqfDAMbaseObj::insertnames()
qbool tqfDAMbaseObj::insertnames(EXTParamInfo* pRowDefParam,EXTfldval &pRtnString)

This method returns a text string containing a comma delimited list specifying the columns to be used in SQL INSERT statement.

- **pRowDefParam** – A list or row variable containing the row definition.
- **pRtnString** - An out parameter that will contain a text string defining the columns to be used in a SQL INSERT statement.

tqfDAMbaseObj::updatenames()
qbool tqfDAMbaseObj::updatenames(EXTParamInfo* pRowDefParam,EXTfldval &pRtnString)

This method returns a text string which is a comma delimited list specifying the columns to be used in SQL UPDATE statement.

- **pRowDefParam** – A list or row variable containing the row definition.
- **pRtnString** - An out parameter that will contain a text string defining the columns to be used in a SQL UPDATE statement.

tqfDAMbaseObj::wherenames()
qbool tqfDAMbaseObj::wherenames(EXTParamInfo* pRowDefParam,EXTParamInfo* pTableNameParam, EXTParamInfo* pComparisonParam, EXTParamInfo* pOperatorParam,EXTfldval &pRtnString)

This method returns a text string that is to be used in SQL WHERE clause.

- **pRowDefParam** – A list or row variable containing the row definition.
- **pTableNameParam** – The name of the table.
- **pComparisonParam** – The comparison predicate to be used e.g. '='
- **pOperatorParam** – The operator to be used between columns e.g. 'AND'
- **pRtnString** - An out parameter that will contain a text string defining the columns to be used in a SQL WHERE clause.

tqfDAMbaseObj::createnames()
qbool tqfDAMbaseObj::createnames(EXTParamInfo* pTableDefParam,EXTfldval &pRtnString, EXTParamInfo* pNullInfo, EXTParamInfo* pPrimK

This method returns a text string defining the columns and types to be used in a SQL CREATE TABLE statement.

- **pTableDefParam** – A list or row variable containing the row definition.
- **pRtnString** - An out parameter that will contain a text string defining the columns and types to be used in a SQL CREATE TABLE statement.
- **pNullInfo** – An optional boolean parameter which is used to append NULL/NOT NULL information to each datatype.
- **pPrimKeys** – An optional Boolean parameter which is used to append PRIMARY KEY information to the required datatype.

tqfDAMbaseObj::newStatement()
qobjinst tqfDAMbaseObj::newStatement(str255& pStatementName, EXTCompInfo* pEci)

This method dynamically generates a new statement object instance.

- **pStatementName** – The name for the new statement object.
- **pEci** – Information about the statement object to be created.

tqfDAMbaseObj::removeStatements()
void tqfDAMbaseObj::removeStatements()

This method removes the session reference from the session's statements and removes statements from session's list of statements. Called internally during destruction of the session object.

tqfDAMbaseObj::setStatements()
void tqfDAMbaseObj::setStatements(tqfDAMStatementObjPtr pStatementList)

This sets a list of Statement objects in the Session object. Called internally during destruction of the session object.

- **pStatementList** – A list of Statement Objects

tqfDAMbaseObj::getStatements()
tqfDAMStatementObjPtr tqfDAMbaseObj::getStatements()

This returns a list of statement objects belonging to the session object.

Example:

```
tqfMyStatementObjPtr currStatement = (tqfMyStatementObjPtr)mSession->getStatements()
while(currStatement != NULL)
{
  // do something
  currStatement = (tqfMyStatementObjPtr)(currStatement->getNext());
}
```

tqfDAMbaseObj::setState()
void tqfDAMbaseObj::setState(eSessionState pState)

This method sets the state of the session. See Session State Constants. The base class handles the session state by default but this method allows the implementation to override for instance if a connection break has been detected.

- **pState** – The state of the session. Can be kSessionLoggedOn or kSessionLoggedOff.

tqfDAMbaseObj::getState()
eSessionState tqfDAMbaseObj::getState()

This method returns the current state of the session. See Session State Constants. Normally used to check whether the session is logged on before performing a session or statement operation.

tqfDAMbaseObj::getTypeTable()
DAMTypeTablePtr tqfDAMbaseObj::getTypeTable()

This returns the database specific type conversion table for the session. Not normally needed by the implementation as the DAMType-Table is handled by the base class.

tqfDAMbaseObj::setTranMode()
qbool tqfDAMbaseObj::setTranMode(qlong pTranMode)

This method sets the transaction mode for the session.

- **pTranMode** – The transaction mode. Can be kTranAutomatic, kTranManual or kTranServer.

tqfDAMbaseObj::getTranMode()
eTranMode tqfDAMbaseObj::getTranMode()

This method returns the transaction mode for the current session. See Transaction Mode Constants.

tqfDAMbaseObj::commit()
qbool tqfDAMbaseObj::commit()

This commits all transactions associated with the session.

tqfDAMbaseObj::rollback()
qbool tqfDAMbaseObj::rollback()

This rolls back all transactions associated with the session.

tqfDAMbaseObj::begin()
qbool tqfDAMbaseObj::begin()

This begins a new transaction. Only supported manual transaction mode, and if transactions are supported, and $autobegin is kFalse.

tqfDAMbaseObj::needsChunk()
qbool tqfDAMbaseObj::needsChunk(qlong pDataLength)

This method determines whether chunking is required based on the data length parameter. If the data length is greater than the lob threshold then chunking is required.

- **pDataLength** – The length of the data.

tqfDAMbaseObj::setLobThreshold()
qbool tqfDAMbaseObj::setLobThreshold(qlong pLobThreshold)

This method sets the data length threshold at which chunking should occur.

- **pLobThreshold** – The length of the data before chunking should occur.

tqfDAMbaseObj::getLobThreshold()
qlong tqfDAMbaseObj::getLobThreshold()

This method returns the current Large OBject threshold set in the session.

tqfDAMbaseObj::setLobChunkSize()
qbool tqfDAMbaseObj::setLobChunkSize(qlong pChunkSize)

This method sets the Large OBject chunk size in the session.

- **pChunkSize** – The size of each chunk

tqfDAMbaseObj::getLobChunkSize()

qlong tqfDAMbaseObj::getLobChunkSize()

This method returns the current Large OBject chunk size set in the session.

tqfDAMbaseObj::getStatementBindMarker()

void tqfDAMbaseObj::getStatementBindMarker(str80 &pBindMarker)

This method returns the bind marker used as a placeholder when preparing statements with bound variables.

- **pBindMarker** – The bind marker

tqfDAMbaseObj::allowsDirectExec()

qbool tqfDAMbaseObj::allowsDirectExec()

This method returns qtrue if the database allows direct execution of a SQL Statement. Called by the base class to determine whether the implementation supports direct execution. See dAllowsDirectExec()

tqfDAMbaseObj::charMapOut()

qbool charMapOut(qbyte *pData, qlong pDataLen, eSessionEncoding pEncoding)

This method converts the character string in the buffer ready to be sent to the server.

8-bit character mapping is only applied to non-Unicode data, i.e. it is not applied if the DAM is operating in Unicode Mode. If pEncoding is not kSessionEncodingAnsi, the data is converted to 8-bit data, character-mapped, then converted back to pEncoding; i.e. the data is assumed to contain 8-bit data encoded as Unicode.

- **pData** – The buffer containing the string to be converted.
- **pDataLen** – The length of the data in the buffer.
- **pEncoding** – The Unicode encoding of the data

tqfDAMbaseObj::charMapIn()

qbool charMapIn(qbyte *pData, qlong pDataLen, eSessionEncoding pEncoding)

This method converts the character string in the buffer ready to be transferred to Omnis.

8-bit character mapping is only applied to non-Unicode data, i.e. it is not applied if the DAM is operating in Unicode Mode. If pEncoding is not kSessionEncodingAnsi, the data is converted to 8-bit data, character-mapped, then converted back to pEncoding; i.e. the data is assumed to contain 8-bit data encoded as Unicode.

- **pData** – The buffer containing the string to be converted.
- **pDataLen** – The length of the data in the buffer.
- **pEncoding** – The Unicode encoding of the data

tqfDAMbaseObj::getCharMap()
eSessionCharMap tqfDAMbaseObj::getCharMap()

This method returns the character map (type) for the session. See Session Character Map Constants.

tqfDAMbaseObj::setBlobSize()
qbool tqfDAMbaseObj::setBlobSize(qlong pBlobSize)

This method sets the maximum length of binary data for insertion into database columns.

pBlobSize must be greater than zero and less than MAX_BINARY_LEN

- **pBlobSize** – The maximum size of a Binary Large Object.

tqfDAMbaseObj::getBlobSize()
qlong tqfDAMbaseObj::getBlobSize()

This method returns the current session BLOB size.

tqfDAMbaseObj::setDefaultDate()
qbool tqfDAMbaseObj::setDefaultDate(datestamptype pDefDate)

This method sets the default date property in the Session.

- **pDefDate** – The default date

tqfDAMbaseObj::getDefaultDate()
datestamptype tqfDAMbaseObj::getDefaultDate()

This method returns the session's default date property.

tqfDAMbaseObj::makeValidDate()
void tqfDAMbaseObj::makeValidDate(datestamptype &pDate)

This method checks the supplied date and substitutes any missing date components from the session's default date property (mDefaultDate) so that the resulting date is suitable for insertion into the database.

- **pDate** – Datestamp variable containing the date/time to be processed

tqfDAMbaseObj::findRpc()
DAMrpcDefn *tqfDAMbaseObj::findRpc(strxxx &pRpcName)

This method locates a Remote Procedure Call by name, returning its definition structure. The RPC must already be registered with the session object using the $rpcdefine() method from Omnis.

- **pRpcName** – The name of the RPC to be located (case-sensitive)

tqfDAMbaseObj::getStripSpaces()
qbool tqfDAMbaseObj::getStripSpaces()

This returns the setting for the session strip-spaces property (mStripSpaces). qtrue indicates that spaces will be stripped from strings being returned to Omnis. This affects the default value assigned to new statements spawned from the session object. Implementation statement objects should therefore not use this property directly. Instead use tqfDAMStatementObj::getStripSpaces().

tqfDAMbaseObj::setStripSpaces()
void tqfDAMbaseObj::setStripSpaces(qbool pDoStrip)

This method sets the session strip-spaces property to the flag specified. qtrue indicates that spaces will be stripped from strings being returned to Omnis. Affects only the default value assigned to new statements. Implementation statement objects should normally use tqfDAMStatementObj::setStripSpaces() instead.

- **pDoStrip** – Strip spaces flag

tqfDAMbaseObj::setCompInfo()
void tqfDAMbaseObj::setCompInfo(EXTCompInfo *pCompInfo)

This method stores the current external component information pointer in the session. Not normally useful to/needed by the implementation.

- **pCompInfo** – The current external component information object.

tqfDAMbaseObj::getCompInfo()
EXTCompInfo *tqfDAMbaseObj::getCompInfo()

This method returns the current external component information object.

tqfDAMbaseObj::setDecSeparator()
qbool tqfDAMbaseObj::setDecSeparator(str15 &pDecSep)

This sets the session object's decimal separator property. Accepts a single character.

- **pDecSep** – The new decimal separator

tqfDAMbaseObj::getDecSeparator()
void tqfDAMbaseObj::getDecSeparator(str15 &pDecSep)

This returns the session object's decimal separator property.

- **pDecSep** – The decimal separator

tqfDAMbaseObj::setThouSeparator()
qbool tqfDAMbaseObj::setThouSeparator(str15 &pThouSep)

This sets the session object's thousand separator property. Accepts a single character.

- **pThouSep** – The new thousand separator

tqfDAMbaseObj::getThouSeparator()
void tqfDAMbaseObj::getThouSeparator(str15 &pThouSep)

This returns the session object's thousand separator property.

- **pThouSep** – The thousand separator

tqfDAMbaseObj::convertDecimalSep()
qbool tqfDAMbaseObj::convertDecimalSep(str255 &pDecimalString)

This method converts the thousand and decimal separators in the decimal string from Omnis separators ($root.$prefs.$separators) to the session object's decimal and thousand separators.

- **pDecimalString** – String variable containing the number.

tqfDAMbaseObj::allowsTransactions()
qlong tqfDAMbaseObj::allowsTransactions()

This method returns the session object's mAllowsTransactions property. May be cBoolUnknown, qfalse or qtrue.

tqfDAMbaseObj::getProcessFolder() Mac OSX only
static qbool tqfDAMbaseObj::getProcessFolder(str255 &pProcessFolder, qbool pMainFolderOnly = qfalse)

Returns the HFS (colon delimited) path to the folder containing the Omnis executable.

If pFolderNameOnly is selected, the path is stripped to the folder containing the Omnis tree.

- **pProcessFolder** - (output) The path to the Omnis process folder.
- **pMainFolderOnly** - If qtrue, the folder containing *Omnis.app* is returned. Otherwise the folder containing the executable is returned (usually the *MacOS* folder).

tqfDAMbaseObj::setEnvFile() Mac OSX only
void tqfDAMbaseObj::setEnvFile()

Initialises an internal path to the DAM-specific '*.ini*' file used to set additional environment variables on OSX. This file resides in the Omnis.app:contents:Macos:xcomp:ini folder and the file name must conform to "damname.ini" where damname is the string assigned to resource number 1000 (cRESDamName). The implementation does not normally need to call this method.

tqfDAMbaseObj::getEnvFile() Mac OSX only
void tqfDAMbaseObj::getEnvFile(str255 &pEnvFile)

This method returns the path to the '*.ini*' file used to set additional environment variables on OSX.

- **pEnvFile** – The name of the environment file.

tqfDAMbaseObj::setEnvVars() Mac OSX only
qbool tqfDAMbaseObj::setEnvVars()

This method reads the environment variables from the DAM's environment file and sets them according to the database implementation. Environment variables set in this way exist in the context of the Omnis process only.

tqfDAMbaseObj::getHostName()
void tqfDAMbaseObj::getHostName(str255 &pHostName)

This method returns the hostname that was previously passed to the $logon() method, or empty if the logon failed.

- **pHostName** – The output parameter to return the hostname into.

tqfDAMbaseObj::getUserName()
void tqfDAMbaseObj::getUserName(str255 &pUserName)

This method returns the username that was previously passed to the $logon() method, or empty if the logon failed.

- **pUserName** – The output parameter to return the username into.

tqfDAMbaseObj:: getPassword()
void tqfDAMbaseObj::getPassword(str255 &pPassword)

This method returns the password that was previously passed to the $logon() method, or empty if the logon failed.

- **pPassword** – The output parameter to return the password into.

tqfDAMbaseObj::emptyDateIsNull()
qbool tqfDAMbaseObj::emptyDateIsNull()

This returns the value of the session object's $emptydateisnull property. This is set to qtrue to allow empty dates to be inserted as Null values.

qreal tqfDAMbaseObj::stringToReal()
qreal tqfDAMbaseObj::stringToReal(strxxx &pString)

This method extracts a real value from the supplied string, converting from server decimal and thousand separators ($sqldecimalseparator & $sqlthousandseparator) to Omnis separators.

- **pString** – The string containing the server formatted decimal number.

tqfDAMbaseObj::realToString()
void tqfDAMbaseObj::realToString(qreal pRealVal, qshort numDp, strxxx &pString)

This method converts a real number to the server string representation, converting from Omnis decimal and thousand separators to server separators.

- **pRealVal** – The decimal value to be converted
- **numDp** – The number of decimal places required in the output
- **pString** – The output string

tqfDAMbaseObj::getApiEncoding()
eSessionEncoding tqfDAMbaseObj::getApiEncoding()

Returns the value of mApiEncoding, previously set using setApiEncoding(), or its default value; kSessionEncodingUtf8. mApiEncoding refers to the encoding required for SQL text and other literal arguments being passed to the client API.

tqfDAMbaseObj::setApiEncoding()
qbool tqfDAMbaseObj::setApiEncoding(qlong pEncoding)

This method sets the Unicode encoding required by the database client API. This refers to the encoding required for SQL text and other literal arguments being passed to the client API.

- **pEncoding** – One of the eSessionEncoding constants, e.g. kSessionEncodingUtf16

tqfDAMbaseObj::getDbEncoding()
eSessionEncoding tqfDAMbaseObj::getDbEncoding()

Returns the value of mDbEncoding, set previously using setDbEncoding(), or its default value; kSessionEncodingUtf8. mDbEncoding is used to convert character bind variables to/from the encoding expected by the server database

tqfDAMbaseObj::setDbEncoding()
qbool tqfDAMbaseObj::setDbEncoding(qlong pEncoding)

This method sets the Unicode encoding expected by the database server.

Affects conversion of bound character data being sent to and received from the server.

- **pEncoding** – One of the eSessionEncoding constants, e.g. kSessionEncodingUtf16

tqfDAMbaseObj::getCodePage()
preconst tqfDAMbaseObj::getCodePage()

This method returns the session object's $codepage property. See mCodePage.

tqfDAMbaseObj::convertCharacterDataToEncoding()
qHandle tqfDAMbaseObj::convertCharacterDataToEncoding(EXTfldval &pData, eSessionEncoding pEncoding)

This method extracts a character string from pData and converts it to the specified Unicode encoding. If pEncoding is kSessionEncodingAnsi, then mCodePage is used to determine the character mapping into the 8-bit character set.

- **pData** - Omnis field value containing the character data
- **pEncoding**- One of the eSessionEncoding constants specifying the target encoding

Example:

```
qHandle hSQLText = mMySession->convertCharacterDataToEncoding(sqlTextFldVal, kSessionEncodingUtf16);
UChar *mySQLText = hSQLText->mData;
```

tqfDAMbaseObj::convertCharacterDataToEncoding()

qlong tqfDAMbaseObj::convertCharacterDataToEncoding(str255 &pString, qbyte *pEncodedString, eSessionEncoding pEncoding)

This method converts the supplied string to the specified Unicode encoding, writing the output to pEncodedString. The user is responsible for ensuring the destination buffer is large enough to accommodate the output.

- **pString** - The source character data.
- **pEncodedString** - The output buffer for the conversion.
- **pEncoding** - One of the eSessionEncoding constants specifying the target encoding

tqfDAMbaseObj::convertCharacterDataToEncoding()

qlong tqfDAMbaseObj::convertCharacterDataToEncoding(qchar *pData, qbyte *pEncodedString, eSessionEncoding pEncoding)

This method converts the supplied string to the specified Unicode encoding, writing the output to pEncodedString. The user is responsible for ensuring the destination buffer is large enough to accommodate the output. The source data must be null-terminated.

- **pData** - The null-terminated source character data.
- **pEncodedString** - The output buffer for the conversion.
- **pEncoding**- One of the eSessionEncoding constants specifying the target encoding

tqfDAMbaseObj::convertCharacterDataFromEncoding()

void tqfDAMbaseObj::convertCharacterDataFromEncoding(qbyte *pParamData, qlong pParamLen, EXTfldval &pOutputData, eSessionEncodi

This method converts character data from the specified Unicode encoding to the Omnis character set, i.e. to UTF-32 for Unicode DAMs

- **pParamData** - The source character data (cast as qbyte *)
- **pParamLen** - The length of the source data in bytes.
- **pOutputData** - An Omnis field value which receives the converted data.
- **pEncoding** - One of the eSessionEncoding constants specifying the source encoding

tqfDAMbaseObj::apiByteLen()

qlong tqfDAMbaseObj::apiByteLen(qlong pLen, eSessionEncoding pEncoding)

This utility method derives the maximum storage size of a character string from the supplied character length. The value of pEncoding is used to determine the size based on the storage requirements for the encoding, e.g. Utf8 has a potential storage requirement of 4 bytes per character. apiByteLen() is not a pure virtual, meaning that the implementation can override this method if required. Otherwise the default implementation should be sufficient.

- **pLen** – The length in characters of a string
- **pEncoding** – The encoding used by the string

Example:

```
fieldLength = mMySession->apiByteLen(fieldLength);
```

tqfDAMbaseObj::apiCharLen()
qlong tqfDAMbaseObj::apiCharLen(qlong pLen)

This utility method derives the equivalent client API character length from the supplied byte length. The value of mApiEncoding is used to determine the divisor used.

dApiCharLen() is not a pure virtual, meaning that the implementation can override this method if required. Otherwise the default implementation should be sufficient.

tqfDAMbaseObj::debugMsg()
void tqfDAMbaseObj::debugMsg(char pLevel, char *pMsg, ...)

This method submits a debug message to the debug file. The message is only written if mDebugLevel >= pLevel and only if a debug file has been previously specified via $debugfile.

pLevel specifies the detail level of the message to allow the user to filter out low level diagnostic messages if necessary. Detail level should conform to the following convention:

1. High level debugging. Used by the base class.
2. More detailed base class debugging.
3. Basic implementation messages, e.g. at the top of implementation methods.
4. Detailed implementation messages, e.g. verbose messages & variable values

The message may contain one or more format markers to be substituted with variable values which are supplied via additional parameters. The following format markers are accepted:

%i or %d – a decimal integer value

%x – a hexadecimal integer value

%f – a floating point decimal value

%b – a non-Unicode string argument (first 80 characters will be written)

%u – a UTF-16 encoded string (first 80 characters will be written, converted to 8-bit)

%s (or any other marker) – UTF-32 encoded string (first 80 characters, converted to 8-bit)

String arguments should be null terminated. The message should also contain a newline escape character ('\n') to format the output correctly. Messages with detail level 3 or higher will be indented when written to the debug file.

- **pLevel** – The session debug level at which the message should appear.
- **pMsg** – The formatted debug message.
- **...** - Zero or more additional parameters corresponding to any variables to be displayed.

Example:

```
mMySession->debugMsg(3,"dConvertParam called, response type=%d, mColumnNum=%d\n", mResponse->type, mColumnNum)
```

tqfDAMbaseObj::getValidateUtf8()
qbool tqfDAMbaseObj::getValidateUtf8()

This method returns the value previously assigned to the $validateutf8 session property.

tqfDAMbaseObj::setValidateUtf8()
void tqfDAMbaseObj::setValidateUtf8(qbool pVal)

This method assigns a new value to the session object's $validateutf8 property; mValidateUtf8.

The statement base class uses this property to determine whether character data should be converted from an ANSI codepage (mCodePage) or treated as UTF-8.

- **pVal** – Determines whether UTF-8 data should be tested to see if it contains only non-Unicode data.

tqfDAMbaseObj::isUnicode() Studio 5+ only
qbool tqfDAMbaseObj::isUnicode()

This method returns the value of the session $unicode property, indicating whether the DAM is operating in Unicode mode (default) or in non-Unicode mode. In non-Unicode mode the DAM should attempt to read/write non-Unicode data, generate and map to non-unicode server data types only.

tqfDAMbaseObj::getDebugFile()
FILE *tqfDAMbaseObj::getDebugFile()

This method returns the file pointer used for writing debug information, allowing custom output to the debug file via fprintf() (stdio.h). Implementation normally calls debugMsg() instead.

tqfDAMbaseObj::getQuotedIdentifier()
qbool tqfDAMbaseObj::getQuotedidentifier()

Returns the value of mQuotedIdentifier. When qtrue, the base class methods; selectnames(), updatenames(), wherenames() and createnames() place quotes around column names. Exposed via $quotedidentifier.

tqfDAMbaseObj::fetch64BitInts()
qbool tqfDAMbaseObj::fetch64BitInts()

Returns the value of mFetch64BitInts. When qtrue, fetched 64-bit integers are returned to Omnis as 64-bit integers. When qfalse, they are converted to character data. Exposed via $fetch64bitints

tqfDAMbaseObj::addbytesReceived()
void tqfDAMbaseObj::addBytesReceived(qlongarch pBytes)

Adds pBytes to mBytesReceived. Used to count the total amount of data received by the session object and its statement objects since logon. Exposed via $bytesreceived.

tqfDAMbaseObj::addbytesSent()
void tqfDAMbaseObj::addBytesSent(qlongarch pBytes)

Adds pBytes to mBytesSent. Used to count the total amount of data sent by the session object and its statement objects since logon. Exposed via $bytessent.

tqfDAMbaseObj::getNativeErrorCode()
qlong tqfDAMbaseObj::getNativeErrorCode()

Returns the value of mErrorInfo.mNativeErrorCode. (Added for use by worker delegate class)

tqfDAMbaseObj::getLocalTimezoneOffset()
int tqfDAMbaseObj::getLocalTimezoneOffset(str80 &pTzOffset, qbool &pIsDst)

Obtains the client machine's local timezone offset as +/-HH:MM and returns the local timezone offset calculated in minutes.

- **pTzOffset** – Returns the client machine's local timezone offset relative to GMT.
- **pIsDst** – pIsDst is set to qtrue if daylight saving time is in effect, qfalse otherwise.

Protected Methods

tqfDAMbaseObj::clearStatements()
qbool tqfDAMbaseObj::clearStatements()

clearStatements() is invoked by calling $clear() from Omnis. clearAll() is called for each object in the session object's linked-list of statement objects (mStatements). clearStatements() fails if any of the clearAll() calls returns qfalse.

tqfImpDAMObj::setSerialised() Deprecated
void tqfImpDAMObj::setSerialised(qbool pSerialised)

Historically used in conjunction with an ECOisSerialised() call which compares the supplied product code against the Omnis serial number to determine whether the serial number supports this component. setSerialised() assigns a value to mIsSerialised which the implementation can access via the isSerialised method().

Example:

```
static qchar sD3OM[5] = {'D','3','O','M',0};   //This product code is for the ODBC DAM
setSerialised(ECOisSerialised(sD3OM));
```

tqfImpDAMObj::isSerialised() Deprecated
void tqfImpDAMObj::isSerialised()

Returns the value of mIsSerialised. mIsSerialised defaults the qfalse when the object is constructed.

Example:

```
if(isSerialised() == qtrue)
  status = sqlAllocHandle(SQL_HANDLE_DBC, mEnvironHandle, &mConnectionHandle);
else
  setError(kDAMNotSerialised);
```

tqfImpDAMObj::setQuotedIdentifier()
void tqfImpDAMObj::setQuotedIdentifier(qbool pQuotedIdentifier)

Assigns mQuotedIdentifier to the value of pQuotedIdentifier. mQuotedIdentifier is used to place quotes around column names generated by the metadata functions; selectnames(), updatenames(), wherenames() and createnames().

tqfImpDAMObj::setError()
void tqfImpDAMObj::setError(qlong pErrorCode)

This sets the session error and retrieves a native session error if it exists. dGetNativeError() is called to obtain the native error code and error text.

Example:

```
setError(kDAMParameterNumError);
```

tqfImpDAMObj::setNativeError()
qbool tqfDAMbaseObj::setNativeError()

setNativeError() is invoked by calling $nextnativeerror() from Omnis as well as from the setError() protected method. It calls dGetNativeError() and sets the mErrorInfo error text & error code with the results. If dGetNativeError() returns kErrorPending, the error pending flag of mErrorInfo is set to qtrue.

Protected Virtual Methods

Unless otherwise stated, these are all pure-virtual functions meaning that they must be implemented by the subclass. Non-pure virtual functions have a default implementation, so overriding them is optional.

tqfImpDAMObj::dGetStatementId()
qlong tqfImpDAMObj::dGetStatementId()

This method is used by the base class to retrieve the implementation's statement object id. The statement object id is a value that is required by the component library and should be specified by the developer in their implementations header file. For example the id for the ODBC DAM is

```
const cObject_ODBCObj = 15,
      cObject_ODBCStat = 16;
```

Example:

```
qlong tqfODBCDAMObj::dGetStatementId()
{
  return cObject_ODBCStat;
}
```

tqfImpDAMObj::dSetBindMarker()
qbool tqfImpDAMObj::dSetBindMarker()

This method is used by the base class to obtain the implementation's bind marker.

The bind marker is the symbol required by the client to act as an indicator for a bind variable. For ODBC the bind marker is a '?'. The implementation normally loads the bind marker from the resource file.

Example:

```
qbool tqfODBCDAMObj::dSetBindMarker()
{
  RESloadString(gInstLib, cResBindMarker, mStatementBindMarker);
  return qtrue;
}
```

tqfImpDAMObj::dLogon()

qbool tqfImpDAMObj::dLogon(str255& pHostname, str255& pUsername, str255& pPassword)

This method implements the session object's logon code. It should log on to the host server using the supplied username and password and establish a connection that this session can use to send commands to the server. Once a connection is created the kState property is set to kSessionLoggedOn. If the parameters are left empty then depending on the DAM object type the user should, (where possible,) be prompted by the database client to provide a value. Some servers will implicitly begin a transaction. Where this is not the case then in kTranAutomatic transaction mode a transaction should be started. The $hostname, $username and $password properties are set to those values passed to this call.

- **pHostname** - The host name required to connect to the database.
- **pUsername** – The user name required to connect to the database.
- **pPassword** – The password required to connect to the database.

Example:

```
qbool tqfODBCDAMObj::dLogon(str255& pHostname, str255& pUsername, str255& pPassword)
{
  qbool rtnStatus = qfalse;
  status = SQLAllocConnect(mEnvironHandle, &mConnectionHandle);
  if(status == SQL_SUCCESS)
  {
    status = SQLConnect(mConnectionHandle, pHostname.cString(), SQL_NTS, pUsername.cString(), SQL_NTS, pPasswo
  }
  if(status == SQL_SUCCESS)
  {
    // Do post logon processing
    rtnStatus = qtrue;
  }
  return rtnStatus;
}
```

tqfImpDAMObj::dLogoff()

qbool tqfImpDAMObj::dLogoff()

This method disconnects from the database and frees any connection resources. The connection is dropped and the session state is set to kSessionLoggedOff. Depending on the session state, statements are cleared, cursors used by statements are closed and pending results are cleared. This call should fail if the session is not currently logged on. If it fails for any other reason then the connection may be in an undefined state and the current object should be destroyed. This will not destroy any statement objects used by this session. If there was an active transaction on this session then the behaviour of the DBMS should determine if that transaction is committed or rolled back. The $hostname, $username and $password properties are cleared.

Example:

```
qbool tqfODBCDAMObj::dLogoff()
{
  RETCODE status = SQL_ERROR;
  qbool rtnStatus = qtrue;
```

```
  if(mConnectionHandle != NULL)
  {
    if(mTransactionMode != kTranServer)
      rtnStatus = dCommit();
    if(rtnStatus == qtrue)
      status = SQLDisconnect(mConnectionHandle);
    if(status == SQL_SUCCESS)
      status = SQLFreeConnect(mConnectionHandle);
    else
      rtnStatus = qfalse;
  }
  // Perform log off processing here
  return rtnStatus;
}
```

tqfImpDAMObj::dNewStatement()

tqfDAMStatementObjPtr tqfImpDAMObj::dNewStatement()

This method is called by the base class newStatement() method when creating a new database specific statement object. The resulting statement object can be used to send commands and process results. If successful a statement object is returned otherwise this call has no effect possibly due to insufficient resources. The new object should assign defaults to all its custom properties.

Example:

```
tqfDAMStatementObjPtr tqfODBCDAMObj::dNewStatement()
{
  tqfDAMStatementObjPtr newObject = NULL;
  newObject = new tqfODBCStatementObj();
  return newObject;
}
```

tqfImpDAMObj::dColtext()

qbool tqfImpDAMObj::dColtext(ffttype pfftType, qshort pSubType, qlong pLength, str255 &pRtnString)

This method is called by the base class and requires that the implementation return the server-specific SQL text description of supplied Omnis variable.

- **pfftType** – The ffttype of the Omnis variable.
- **pSubType** – The subtype of the Omnis variable.
- **pLength** – The length of the Omnis variable.
- **pRtnString** – The String to return the output into.

Example:

```
qbool tqfODBCDAMObj::dColtext(ffttype pfftType, qshort pSubType, qlong pLength, str255 &pRtnString)
{
  qlong precision = 0;
  qshort scale = 0;
  str15 precText,scaleText;
  eODBCDecisionType odbcType = getIdealType(pfftType,pSubType,pLength, precision,scale);
  ODBCTypeInfoPtr typePtr = matchODBCType(odbcType,precision,scale);
  if(typePtr != NULL)
  {
```

```
    str255 tempString;
    typePtr->getName(tempString);
    pRtnString.concat(tempString);
    if(typePtr->getCreateParams())
    {
      pRtnString.concat('(');//Open Bracket
      qlongToString(precision,precText);
      pRtnString.concat(precText);
      if(scale != -1)
      {
        qlongToString((qlong)scale,scaleText);
        pRtnString.concat(',');
        pRtnString.concat(scaleText);
      }
      pRtnString.concat(')');//Close Bracket
    }
    return qtrue;
  }
  return qfalse;
}
```

tqfImpDAMObj::dGetTypeTable()
DAMTypeTablePtr tqfImpDAMObj::dGetTypeTable()

This function returns the DAM specific type table which maps Omnis types to database types. The DAMTypeTable class contains the mapping between Omnis data types and C Types used for transferring data. The Type table is created during construction of the class.

Example:

```
DAMTypeTablePtr tqfODBCDAMObj::dGetTypeTable()
{
  return (DAMTypeTablePtr)&mDAMTypeTable;
}
```

tqfImpDAMObj::dSetTranMode()
qbool tqfImpDAMObj::dSetTranMode(eTranMode pTranMode)

This method is called by the base class and requires that the implementation sets the transaction mode for the session.

- **pTranMode** – The transaction mode- one of the eTranMode constants.

Example:

```
qbool tqfODBCDAMObj::dSetTranMode(eTranMode pTranMode)
{
  RETCODE status = SQL_SUCCESS;
  qbool rtnStatus = qtrue;
  if(mTransactionMode != pTranMode)
  {
    if(mTransactionMode == kTranServer)
    // turn off driver auto commit
    {
      status = SQLSetConnectOption(mConnectionHandle, SQL_AUTOCOMMIT, qfalse);
    }
    if(status == SQL_SUCCESS)
```

```
    {
      switch(pTranMode)
      {
        case kTranAutomatic:  // do a commit
        {
          rtnStatus = commit();
          if(rtnStatus == qfalse)
          {
            rollback();
       }
       break;
        }
        case kTranServer:  // turn on driver auto commit
        {
          status = SQLSetConnectOption(mConnectionHandle, SQL_AUTOCOMMIT, qtrue);
          if(status != SQL_SUCCESS)
          {
            rtnStatus = qfalse;
          }
          break;
        }
        default:
       {
          // do nothing
        }
      }
    }
    else
    {
      rtnStatus = qfalse;
    }
  }
  return rtnStatus;
}
```

tqfImpDAMObj::dGetCommitMode()
eCommitMode tqfImpDAMObj::dGetCommitMode()

This method returns the implementation's cursor commit behaviour- one of the eCommitMode constants.

Example:

```
eCommitMode tqfODBCDAMObj::dGetCommitMode()
{
  RETCODE status;
  SWORD length;
  SWORD transBehaviour;
  eCommitMode commitMode = kCommitPreserve;
  status = SQLGetInfo(mConnectionHandle, SQL_CURSOR_COMMIT_BEHAVIOR, &transBehaviour, sizeof(SWORD), (SWORD FA
  if(status == SQL_SUCCESS)
  {
    if(transBehaviour == SQL_CB_DELETE)
    {
      commitMode = kCommitDelete;
    }
    else if(transBehaviour == SQL_CB_CLOSE)
    {
      commitMode = kCommitClose;
```

```
    }
  }
  return commitMode;
}
```

tqfImpDAMObj::dGetRollbackMode()
eRollbackMode tqfImpDAMObj::dGetRollbackMode()

This method returns the implementation's cursor rollback behaviour. This can be kRollbackDelete, kRollbackClose or kRollbackPreserve.

Example:

```
eRollbackMode tqfODBCDAMObj::dGetRollbackMode()
{
  RETCODE status;
  SWORD length;
  SWORD transBehaviour;
  eRollbackMode rollbackMode = kRollbackPreserve;
  status = SQLGetInfo(mConnectionHandle, SQL_CURSOR_ROLLBACK_BEHAVIOR, &transBehaviour, sizeof(SWORD), (SWORD
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
  {
    if(transBehaviour == SQL_CB_DELETE)
      rollbackMode = kRollbackDelete;
    else if(transBehaviour == SQL_CB_CLOSE)
      rollbackMode = kRollbackClose;
  }
  return rollbackMode;
}
```

tqfImpDAMObj::dUsesAutoBegin()
qbool tqfImpDAMObj::dUsesAutoBegin()

This method is called by the base class begin() method to determine whether the implementation automatically starts a new transaction whenever a $commit() is executed. If qfalse is returned, this prompts dBegin() to be called.

Example:

```
qbool tqfODBCDAMObj::dUsesAutoBegin()
{
  return qtrue;
}
```

tqfImpDAMObj::dCommit()
qbool tqfImpDAMObj::dCommit()

This method is called by the base class when in manual transaction mode and requires that the implementation issues a commit. This should fail if the $state property is kSessionLoggedOff, the transaction mode is not kTranManual or there is no current transaction. With certain types of databases this will commit and clear all statements, close any cursors used by a statement and clear pending results. This will not destroy the statement objects used by a DAM object. The server may start a new transaction in response to the commit.

Example:

```
qbool tqfODBCDAMObj::dCommit()
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = SQLTransact(mEnvironHandle, mConnectionHandle, SQL_COMMIT);
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpDAMObj:: dRollback()
qbool tqfImpDAMObj:: dRollback()

This method is called by the base class when in manual transaction mode and requires that the implementation issues a rollback. This should fail if the $state property is kSessionLoggedOff, the transaction mode is not kTranManual or there is no current transaction. This call cancels the current transaction. With certain types of database this will rollback and clear all statements, close any cursors used by a statement and clear pending results. This will not destroy the statement objects used by a DAM object. The server may start a new transaction in response to the rollback.

Example:

```
qbool tqfODBCDAMObj::dRollback()
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = SQLTransact(mEnvironHandle, mConnectionHandle, SQL_ROLLBACK);
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpDAMObj::dBegin()
qbool tqfImpDAMObj::dBegin()

This method is only applicable to certain types of database where the $autobegintran property is kFalse. This call is used to start a transaction when in kTranManual mode and is necessary where the database does not implicitly start a transaction when a connection is established or the current transaction is committed or rolled back. This call should fail if there is a current transaction or the server does not support nested transactions.

Example:

```
qbool tqfODBCDAMObj::dBegin()
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = SQLTransact(mEnvironHandle, mConnectionHandle, SQL_BEGIN);
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpDAMObj::dClearNativeError()
void tqfImpDAMObj::dClearNativeError()

This method should clear any currently cached native error.

Example:

```
void tqfODBCDAMObj::dClearNativeError()
{
  mErrorInfo.clearError();
}
```

tqfImpDAMObj::dGetNativeError()

eErrorState tqfImpDAMObj::dGetNativeError(qlong &errorCode, EXTfldval &pErrorText)

This method should return the currently cached error code and message.

- **errorCode** – The native error code to be returned.
- **pErrorText** – The native error message to be returned.

Example:

```
eErrorState tqfODBCDAMObj::dGetNativeError(qlong &errorCode, EXTfldval &pErrorText)
{
  eErrorState errorState = kErrorOk;
  if(mErrorInfo.getErrorPending() == qtrue) // get cached error
  {
    errorCode = mErrorInfo.getErrorCode();
    mErrorInfo.getErrorText(pErrorText);
    errorState = mErrorInfo.setError(); // next error ?
  }
  else // set up error
  {
    errorState = mErrorInfo.setError();
    errorCode = mErrorInfo.getErrorCode();
    mErrorInfo.getErrorText(pErrorText);
    if(errorState == kErrorPending)
      errorState = mErrorInfo.setError(); // set up next error
  }
  return errorState;
}
```

tqfImpDAMObj::dAllowsDirectExec()

qbool tqfImpDAMObj::dAllowsDirectExec()

This method is called by the base class to determine whether the implementation supports direct statement execution. If qtrue is returned, the base class calls StatObj::dExecDirect() in respect of $execdirect() calls. Otherwise StatObj::dPrepare() and StatObj::dExecute() are called.

Example:

```
qbool tqfODBCDAMObj::dAllowsDirectExec()
{
  return qtrue;
}
```

tqfImpDAMObj::dMethodCall()

qbool tqfImpDAMObj::dMethodCall(EXTCompInfo* pEci, EXTfldval &rtnVal, qbool &hasRtnVal)

This method is called by the base class for any function calls defined by the implementation. These methods are the DAM specific methods that will appear in the Session object in addition to the generic methods. For further information about adding methods to you DAM object please review the Omnis External Components Tutorial.

- **pEci** – The external component information object
- **rtnVal** – The value returned by the dam specific method
- **hasRtnVal** – Set if rtnVal contains a value.

Example:

```
qbool tqfODBCDAMObj::dMethodCall(EXTCompInfo* pEci, EXTfldval &rtnVal, qbool &hasRtnVal)
{
  qbool rtnStatus = qfalse;
  switch(ECOgetId(pEci))
  {
    case cODBCMethodId_DefaultDate:
    {
      hasRtnVal = qtrue;
      rtnStatus = setDefaultDate(pEci);
      rtnVal.setLong(rtnStatus);
      break;
    }
    default:
    {
    }
  }
  return rtnStatus;
}
```

---

tqfImpDAMObj::dPropertyCanAssign()

---

qbool tqfImpDAMObj::dPropertyCanAssign(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci)

This method determines whether a property is read only. The properties included in this function are those that are specific to the DAM implementation. If the end user can modify a property then qtrue should be returned , if it is read only then qfalse should be returned. For further information about adding properties to your DAM object please review the Omnis External Components Tutorial.

- **wParam** – Not Used
- **lParam** – Not Used
- **pEci** – The external component information object

Example:

```
qbool tqfODBCDAMObj::dPropertyCanAssign(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci)
{
  qbool rtnStatus = qfalse;
  switch(ECOgetId(pEci))
  {
    case cODBCSessionLoginTimeOut:
    {
      rtnStatus = qtrue;
      break;
    }
    default:
    {
      break;
    }
  }
  return rtnStatus;
}
```

tqfImpDAMObj::dSetProperty()

qbool tqfImpDAMObj::dSetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci)

This method is called by the base class to set the value of a custom property. The properties included in this function are those that are specific to the DAM implementation. For further information about adding properties to your DAM object please review the Omnis External Components Tutorial.

- **wParam** - wParam is set to ECM_WPARAM_PROPBUTTON if the Property Manager popup button was pressed to set the property. For example, a file name property may wish to use a file open dialog if the popup button was pressed.
- **lParam** – Not Used
- **pEci** – The external component information object

Example:

```
qbool tqfODBCDAMObj::dSetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci)
{
  qbool rtnStatus = qfalse;
  mErrorInfo.clearError();
  // get the externals parameter
  EXTParamInfo* param = ECOfindParamNum( pEci, 1 );
  if ( param )
  {
    // get the fldval from the parameter.
    EXTfldval fval( (qfldval)param->mData );
    switch(ECOgetId(pEci))
    {
      case cODBCSessionLoginTimeOut:
      {
        rtnStatus = setLoginTimeOut(fval.getLong());
        break;
      }
 //...
    }
  }
  return rtnStatus;
}
```

tqfImpDAMObj::dGetProperty()

qbool tqfImpDAMObj::dGetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci, EXTfldval &pPropVal)

This method is called by the base class to retrieve the value of a custom property. The properties included in this function are those that are specific to the DAM implementation. For further information about adding properties to your DAM object please review the Omnis External Components Tutorial and earlier sections of this manual.

- **wParam** – Not used
- **lParam** – Not used
- **pEci** – The external component information object
- **pPropVal** – The value of the property

Example:

```
qbool tqfODBCDAMObj::dGetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* pEci, EXTfldval &pPropVal)
{
  qbool rtnStatus = qtrue;
  switch(ECOgetId(pEci))
  {
    case cODBCSessionDBMSName:
    {
      pPropVal = mDBMSName;
      break;
    }
    case cODBCSessionLoginTimeOut:
    {
      pPropVal.setLong(mLoginTimeOut);
      break;
    }
    default:
    {
      rtnStatus = qfalse;
    }
  }
  return rtnStatus;
}
```

tqfImpDAMObj::dGetVersion()
qbool tqfImpDAMObj::dGetVersion(str255 &pVersion)

This method is called by the base class to obtain the database version of the session connection.

- **pVersion** – (output) The version string

Example:

```
qbool tqfODBCDAMObj::dGetVersion(str255 &pVersion)
{
  str255 nameStr;
  str255 versionStr;
  mDBMSName.getChar(nameStr);
  mDBMSVersion.getChar(versionStr);
  nameStr.concat(' ');
  pVersion.concat(nameStr, versionStr);
  return qtrue;
}
```

tqfImpDAMObj::dGetApiVersion()
qbool tqfImpDAMObj::dGetApiVersion(str255 &pApiVersion)

This method is called by the base class to obtain the API version of the client library that the DAM was built against. If this cannot be determined dynamically, it should be stored as a resource and loaded when requested.

- **pApiVersion** – (output) The API version string.

Example:

```
qbool tqfODBCDAMObj::dGetApiVersion(str255 &pApiVersion)
{
  RESloadString(gInstLib, cResApiVersion, pApiVersion);
  return qtrue;
}
```

tqfImpDAMObj::dAllowsCharConversion()
qbool tqfImpDAMObj::dAllowsCharConversion(eCharMapMode pMode)

Relevant to DAMs operating in non-Unicode mode only, this function allows the implementation to refuse internal character conversion between Windows and MacRoman character sets. Internal character set conversion is selectively performed before custom output character mapping and after custom input character mapping to ensure that custom character maps always operate using the Omnis/MacRoman character set irrespective of the client platform.

This is not a pure virtual function and so it only needs to be overridden in the DAM if desired. The default implementation returns qtrue.

- **pMode** – The character map mode (direction). Either kCharMapIn or kCharMapOut.

Example:

```
qbool tqfDAMbaseObj::dAllowsCharConversion(eCharMapMode pMode)
{
  if (pMode == kCharMapIn)
    return qtrue;
  else
    return qfalse;
}
```

tqfImpDAMObj::dUnicodeCanAssign()
qbool tqfImpDAMObj::dUnicodeCanAssign()

This method allows the implementation to prevent changes to the session $unicode property. This may be useful if the implementation only supports Unicode or visa-versa.

This is not a pure virtual function and so it only needs to be overridden in the DAM if desired. The default implementation returns qtrue, implying that the implementation can switch modes. See mIsUnicode.

Example:

```
qbool tqfDAMbaseObj::dUnicodeCanAssign()
{
  return qtrue;
}
```

tqfImpDAMObj::dUnicodeChanged()
void tqfImpDAMObj::dUnicodeChanged()

This method is called by the base class following a change to the session $unicode property. Not called if dUnicodeCanAssign() returns qfalse.

This is not a pure virtual function and so it only needs to be overridden in the DAM if desired. The default implementation does nothing.

Example:

```
void tqfODBCDAMObj::dUnicodeChanged()
{
  eSessionEncoding encoding = (mIsUnicode==qtrue)? kSessionEncodingUtf16 : kSessionEncodingAnsi;
  setDbEncoding(encoding);
}
```

tqfImpDAMObj::dSeparatorsChanged()	Deprecated
void tqfImpDAMObj::dSeparatorsChanged()	

Called by the base class whenever an assignment is made to the $sqldecimalseparator property.

The mDecimalSep private member is used internally when converting from the server decimal separator to the Omnis decimal separator and the server decimal separator is invariably set to the SQL standard '.' Character (except in the case of the OmnisSQL DAM).

tqfDAMStatementObj

This is a generic object that encapsulates a database cursor or statement handle. Its purpose is to be sub-classed into a more specific database implementation. It contains all attributes and operations relevant to a statement or cursor.

Protected Members

Type	Name	Description
DAMErrorInfo	mErrorInfo	An object containing the statement object's error information.
tqfDAMbaseObj	*mSession	The session object that created this statement.
qlong	mRowsAffected	The number of rows affected by the last SQL statement.
qshort	mColumnCount	The number of columns in the current result set.
qlong	mRowCount	The number of rows in the current result set.
qlong	mNoRows	The number of rows in current fetch batch.
qlong	mBatchRow	The row number of the next batch row to be processed.
EXTfldval	mRpcReturnValue	The value returned by the last RPC called.
DAMData	mInputParams	An object containing information about all the input variables or columns.
DAMData	mOutputParams	An object containing information about all the output variables or columns.
DAMstaAddRowMode	mAddRowMode	The mode in which add row is to execute after a fetch. See Constants
DAMbindVariables	mBindVars	The object containing information about all the Omnis bound variables.
qbool	mUseCursor	When qtrue, a database cursor should be used where possible. Can be scrollable or updateable depending on implementation.
EXTfldval	mSQLText	The current SQL statement.
qbool	mResultsPending	qtrue when a result set is pending, waiting to be fetched.
qlong	mBatchSize	The maximum number of concurrent rows to fetch in a single network transaction. Read only if not implemented. See mCanChangeBatchSize
qlong	mMaxBufferSize	The maximum buffer size used to store an array of fetched column data. Used when batch fetching.
qbool	mCanChangeBatchSize	Default value qfalse. Implementation must set to qtrue to allow batchsize to be changed.

Type	Name	Description
qbool	mCancelled	Set to qtrue when cancel() is called in order to cancel processing of a result set. Added for use by the worker delegate class.
qlong	mColumnsFlags	Stores the value of parameter 2 supplied to the $columns() method. Flag constants are contained in the Catalog under 'Statement Flags' and are currently used by the MySQL DAM to return datetime, numeric and integer precisions via the Length column.

Private Methods

Although you may not call private methods directly, these are referred to in the Omnis Methods section and in the damstat.he header file so are described here in overview.

tqfDAMStatementObj::defineList()
void tqfDAMStatementObj::defineList()

defineList() uses the value of mAddRowMode to define the statement object's internal fetch list; mTableRef.

If mAddRowRow = eDAMaddRowNormal, the list is defined from the buffer types, subtypes and field lengths contained in mOutput-Params. Otherwise, defineMetaDataList() is called with the appropriate pre-defined column structure.

tqfDAMStatementObj::defineMetaDataList()
void tqfDAMStatementObj::defineMetaDataList(EXTqlist *pList, DAMlistDefn *pListDefn, qshort pColCount)

defineMetaDataList() defines the supplied list (mTableRef) according to the specified definition structure, reading column headings from resource strings as needed. The resulting list definition will be suitable for returning a meta-data result set, e.g. $columns(), $indexes(), $rpcparameters().

tqfDAMStatementObj::fetchRows()
eFetchStatus tqfDAMStatementObj::fetchRows(qlong pRowCount)

fetchRows() fetches a batch of rows from the pending result set and adds them to the internal fetch list. It calls dFetch() to populate the output parameter buffers, then calls addRow() for each row in the batch (i.e. once when mBatchSize = 1).

fetchRows() continues until the fetch status becomes kFetchFinished or kFetchError, or until the requisite number of rows have been fetched. mRowsFetched is set to indicate the number of rows added to mTableRef.

tqfDAMStatementObj::rowsAvailable()
qlong tqfDAMStatementObj::rowsAvailable()

rowsAvailable() subtracts the maximum number of rows for an Omnis list variable from the number of rows in the internal fetch list (mTableRef) to derive the maximum number of rows that can safely be inserted into mTableRef.

tqfDAMStatementObj::addRow()
qlong tqfDAMStatementObj::addRow()

addRow() adds one row to the fetch list. If mAddRowMode = eDAMaddRowNormal, it then uses mColumnCount to loop through each output parameter, calling setOmnisColumnData() to set each column value. If mAddRowMode indicates a meta-data result set, addTableRow(), addColumnRow(), addIndexRow(), addRpcProcedureRow() or addRpcParameterRow() is called to populate the column values instead.

tqfDAMStatementObj::callFetch()
eFetchStatus tqfDAMStatementObj::callFetch(EXTCompInfo* pEci)

callFetch() verifies that a result set is pending (mResultsPending). It then validates the parameters that were supplied to the $fetch() method, determines ownership of the fetch list and whether it is a list or a row before calling the fetch() public method.

tqfDAMStatementObj::callFetchInto()
eFetchStatus tqfDAMStatementObj::callFetchInto(EXTCompInfo* pEci)

callFetchInto() verifies that a result set is pending (mResultsPending). It then sets mBatchSize to 1. mTableRef is cleared and redefined according to the Omnis variables that were passed to the $fetchinto() method. One row is then added to mTableRef before calling fetch().

callFetchInto() then loops through the output columns, writing the values back to the supplied Omnis variables.

tqfDAMStatementObj::callFetchToFile()
eFetchStatus tqfDAMStatementObj::callFetchToFile (EXTCompInfo* pEci)

callFetchToFile() verifies that a result set is pending. It then validates the parameters that were supplied to the $fetchtofile() method and clears mTableRef. It then calls fetch() followed by writeListToFile(); passing it the specified filename, rowCount, append, columnNames and encoding parameters.

tqfDAMStatementObj::writeListToFile()
qbool tqfDAMStatementObj::writeListToFile(str255 &pFilename, qlong pRowCount, qbool pAppend, qbool pColumnNames, preconst pEncodi

writeListToFile() writes the contents of mTableRef to the specified external file. All data is converted to character data and to the specified Unicode encoding. The file is closed when the last row has been written.

tqfDAMStatementObj::clear()
qbool tqfDAMStatementObj::clear()

clear() calls clearResults() and resets the statement object's private attributes back to their default states. mInputParams and mOutputParms are dropped and clear_mTableRef() is called.

See also: clearAll()

tqfDAMStatementObj::clearResults()
qbool tqfDAMStatementObj::clearResults()

clearResults() sets mRowsFetched = 0, mBatchRow = 1, mRowsAffected = 0, mColumnCount = 0, mRowCount = 0, mNoRows = 0 and mResultsPending = qfalse.

tqfDAMStatementObj::tables()
qbool tqfDAMStatementObj::tables(EXTCompInfo* pEci)

tables() first validates the parameters that were supplied to the $tables() method. It then calls prepareForMetaDataMethod() and dTables(), which implements the SQL to generate the result set. readyForFetch(eDAMaddRowTables) and describe() are then called to process the pending result set.

tqfDAMStatementObj::columns()
qbool tqfDAMStatementObj::columns(EXTCompInfo* pEci)

columns() first validates the parameters that were supplied to the $columns() method. It then calls prepareForMetaDataMethod() and dColumns(), which implements the SQL to generate the result set. readyForFetch(eDAMaddRowColumns) and describe() are then called to process the pending result set.

tqfDAMStatementObj::indexes()
qbool tqfDAMStatementObj::indexes(EXTCompInfo* pEci)

indexes() first validates the parameters that were supplied to the $indexes() method. It then calls prepareForMetaDataMethod() and dIndexes(), which implements the SQL to generate the result set. readyForFetch(eDAMaddRowIndexes) and describe() are then called to process the pending result set.

tqfDAMStatementObj::results()
qbool tqfDAMStatementObj::results(EXTCompInfo* pEci)

results() first validates the parameters that were supplied to the $results() method. It then clears and defines the supplied list by calling defineMetaDataList() passing it a predefined column structure.

If mAddRowMode is eDAMaddRowNormal, results() processes mOutputParams, extracting the column name, ffttype, subtype and field length of each column before calling dResults(), which allows the implementation to modify these values if required. The DAMgetDataTypeText() callback is then used to obtain the Omnis data type text for each column. dGetColumnInfoForResults() is then called, allowing the implementation to further process each output parameter and/or the SQL data type text and column length if required. addResultsRow() is then called which adds a row containing the information about each result set column.

If $results() was called following a meta data query, results() calls metaDataResults() passing in the appropriate predefined column structure depending on the value of mAddRowMode. This defines and populates the output results list with static values.

tqfDAMStatementObj::addTableRow()
void tqfDAMStatementObj::addTableRow(qlong pRowNumber)

When mAddRowMode is eDAMaddRowTables, addTableRow() is called during addRow() at fetch time. This obtains data required for specific columns of the result set returned by $tables(). The following derived methods are called to achieve this: dTablesGetOwner(), dTablesGetTableName(), dTablesGetTableType(), dTablesGetDescription() and dTablesGetDamInfo().

Each method is passed a reference to the relevant column of mTableRef at the specified row number (pRowNumber), allowing the implementation to set the column values directly.

tqfDAMStatementObj::addColumnRow()
void tqfDAMStatementObj::addColumnRow(qlong pRowNumber)

When mAddRowMode is eDAMaddRowColumns, addColumnRow() is called during addRow() at fetch time. This obtains data required for specific columns of the result set returned by $columns(). The following derived methods are called to achieve this: dColumnsGetDatabaseOrCatalog(), dColumnsGetOwner(), dColumnsGetColumnName(), dColumnsGetOmnisDataTypeText(), dColumnsGetOmnisDataType()*, dColumnsGetOmnisDataSubType()*, dColumnsGetSqlDataType(), dColumnsGetLength(), dColumnsGetScale(), dColumnsGetNull(), dColumnsGetIndex(), dColumnsGetPrimaryKey(), dColumnsGetDescription() & dColumnsGetDamInfo().

Each method is passed a reference to the relevant column of mTableRef at the specified row number (pRowNumber), allowing the implementation to set the column values directly.

* dColumnsGetOmnisDataType() & dColumnsGetOmnisDataSubType() are called by addOmnisTypeAndSubType() which attempts to derive Omnis constants to represent the data type and subtype.

tqfDAMStatementObj::addIndexRow()
void tqfDAMStatementObj::addIndexRow(qlong pRowNumber)

When mAddRowMode is eDAMaddRowIndexes, addIndexRow() is called during addRow() at fetch time. This obtains data required for specific columns of the result set returned by $indexes(). The following derived methods are called to achieve this: dIndexesGetDatabaseOrCatalog(), dIndexesGetOwner(), dIndexesGetColumnName(), dIndexesGetIndexName(), dIndexesGetUnique(), dIndexesGetColumnPosition() & dIndexesGetDamInfo().

Each method is passed a reference to the relevant column of mTableRef at the specified row number (pRowNumber), allowing the implementation to set the column values directly.

tqfDAMStatementObj::addRpcProcedureRow()
void tqfDAMStatementObj::addRpcProcedureRow(qlong pRowNumber)

When mAddRowMode is eDAMaddRowRpcProcedures, addRpcProcedureRow() is called during addRow() at fetch time. This obtains data required for specific columns of the result set returned by $rpcprocedures(). The following derived methods are called to achieve this: dRpcProceduresGetDatabaseOrCatalog(), dRpcProceduresGetOwner(), dRpcProceduresGetProcedureName() & dRpcProceduresGetDamInfo().

Each method is passed a reference to the relevant column of mTableRef at the specified row number (pRowNumber), allowing the implementation to set the column values directly.

tqfDAMStatementObj::addRpcParameterRow()
void tqfDAMStatementObj:: addRpcParameterRow (qlong pRowNumber)

When mAddRowMode is eDAMaddRowRpcParameters, addRpcParameterRow () is called during addRow() at fetch time. This obtains data required for specific columns of the result set returned by $rpcparameters(). The following derived methods are called to achieve this: dRpcParametersGetOmnisDataTypeText(), dRpcParametersGetDatabaseOrCatalog(), dRpcParametersGetOwner(), dRpcParametersGetParameterName(), dRpcParametersGetSqlDataType(), dRpcParametersGetScale(), dRpcParametersGetLength(), dRpcParametersGetPassType() & dRpcParametersGetDamInfo(). addOmnisTypeAndSubType() is also called to derive constant values for the equivalent Omnis data type and subtype.

Each method is passed a reference to the relevant column of mTableRef at the specified row number (pRowNumber), allowing the implementation to set the column values directly.

tqfDAMStatementObj::prepareForMetaDataMethod()
qbool tqfDAMStatementObj::prepareForMetaDataMethod()

This method checks that the session and statement objects are in a valid state for a meta data call ($columns(), etc). If successful, clearAll() is called to clear the statement.

tqfDAMStatementObj::metaDataResults()
qbool tqfDAMStatementObj::metaDataResults(EXTqlist *pResultsList, DAMlistDefn *pListDefn, qshort pColCount)

metaDataResults() uses the supplied list column structure and column count to populate pResultsList. It reads result column names from resource strings and calls the DAMgetDataTypeText() callback to obtain the Omnis data type text corresponding to each column. It then calls dMetaDataGetSqlDataTypeText() in order to obtain the implementation's SQL data type corresponding to the column's ffttttype. addResultsRow() is then called to add a description of each column of the meta data results.

tqfDAMStatementObj::addResultsRow()
qbool tqfDAMStatementObj::addResultsRow(EXTqlist *pResultsList, str255 &pColumnName, str255 &pOmnisDataTypeText, str255 &pSqlDataTypeText, qlong pColumnLength, qlong pScale, qbool pCanBeNull)

addResultsRow() adds a row to the result set generated by a $results() call. A row is inserted into pResultsList. The row is then populated using the supplied parameter values.

tqfDAMStatementObj::columnCount()
qshort tqfDAMStatementObj::columnCount()

If mAddRowMode is eDAMaddRowNormal, columnCount() simply returns mColumnCount.

For eDAMaddRowTables, eDAMaddRowColumns, eDAMaddRowIndexes, eDAMaddRowRpcProcedures and eDAMaddRowRpcParameters it returns the number of columns contained in the relevant static list definition, i.e. 5 for $tables(), 14 for $columns(), etc.

tqfDAMStatementObj::bindOutputBuffers()
qbool tqfDAMStatementObj::bindOutputBuffers()

bindOutputBuffers() allocates data and length buffers for mOutputParams. If mBatchSize is > , the buffers are multiplied in size accordingly. For each output parameter with a bindType of kBindFetch; dBindColumn() is called so that the implementation can present the fetch buffers to the database clientware.

tqfDAMStatementObj::setOmnisColumnData()
qbool tqfDAMStatementObj::setOmnisColumnData(DAMParamPtr pParam, EXTfldval &pOmnisColumn, qshort pParmNumber)

setOmnisColumnData() is called by addRow() at fetch time when mAddRowMode = eDAMaddRowNormal. It handles chunking of result columns as well as data type conversion and character mapping of received data.

If the parameter's chunking flag is set, this method fetches chunks, writes them to the parameter's chunk handle until all chunks have been fetched. Unicode conversion and optionally; character mapping and stripping of spaces is then performed on character data. Character data is converted from the parameter's encoding to the Omnis UTF-32 encoding. Spaces are stripped only if mStripSpaces is qtrue ($sqlstripspaces). Character mapping is performed only if the parameter encoding is kSessionEncodingAnsi or if mSession→isUnicode()=qfalse ($unicode).

If the parameter's chunking attribute is not set, the data in the parameter buffer is optionally converted by the implementation by calling dConvertParam(), non-Unicode character data is character-mapped by calling session→charMapIn(). DAMParam→setOmnisVal() is then called to copy (or convert) the data into pOmnisColumn. dConvertParam() is only called if the parameter's sendType=kConvert.

tqfDAMStatementObj::clearError()
void tqfDAMStatementObj::clearError()

clearError() clears the statement object's mErrorInfo object and also calls dClearNativeError() to allow the implementation to clear any additional error information.

tqfDAMStatementObj::setStatementText()
qbool tqfDAMStatementObj::setStatementText(EXTCompInfo *pEci, EXTfldval& pSQLText, EXTfldval &outStatement, qlong &textLen)

If pSQLText contains bind variable markers, the DAMprepareBindVariables() callback is called, substituting bind marker placeholders for the bind variables. Otherwise any existing bind variables are cleared (DAMfreeBindVariables()). Output character mapping is also performed on the resulting SQL text, which is returned via outStatement. session.mStatementBindMarker is used for the bind marker substitution.

tqfDAMStatementObj::rpcProcedures()
qbool tqfDAMStatementObj::rpcProcedures(EXTCompInfo *pEci)

rpcProcedures() calls prepareForMetaDataMethod(), followed by dRpcProcedures(), readyForFetch(eDAMaddRowRpcProcedures) and describe(). This methods is invoked by calling $rpcprocedures() in Omnis. . Execution stops and an error is returned if any of the primary methods fail.

tqfDAMStatementObj::rpcParameters()
qbool tqfDAMStatementObj::rpcParameters(EXTCompInfo *pEci)

rpcParameters() calls prepareForMetaDataMethod(), followed by dRpcParameters(), readyForFetch(eDAMaddRowRpcParameters) and describe(). This methods is invoked by calling $rpcparameters() in Omnis. Execution stops and an error is returned if any of the primary methods fail.

tqfDAMStatementObj::rpc()
qbool tqfDAMStatementObj::rpc (EXTCompInfo *pEci)

rpc() is invoked by calling $rpc() in Omnis. It fails if the remote procedure was not previously defined ($rpcdefine()) or if mSession→findRpc() cannot match the RPC name.

rpc() calls clearAll() followed by dRpc() which implements the SQL statement, readyForFetch() and describe(), in case the RPC generates a result set.

tqfDAMStatementObj::addOmnisTypeAndSubType()
void tqfDAMStatementObj::addOmnisTypeAndSubType(qlong pRowNumber, qshort pDataTypeCol, qshort pSubTypeCol)

Adds the data type and sub-type from the current row in the result set to the result set

of a $columns() or $rpcparameters() meta-data method. This method attempts to derive Omnis constant values representing the Omnis ffttype and subtype contained in the *pDataTypeCol* and *pSubTypeCol* columns of thespecified row of the result set. The constant values are written back to these columns if successful.

tqfDAMStatementObj::setState()
void tqfDAMStatementObj::setState(eStatementState pState)

setState() assigns the statement's internal mState attribute to the supplied value. mDoExecDirect is set to indicate whether the last statement was executed directly (kStateExecDirect).

tqfDAMStatementObj::clear_mTableRef()
qbool tqfDAMStatementObj::clear_mTableRef()

If the statement object owns the list contents (i.e. if mTableRefDisposable=qtrue), the contents of mTableRef are cleared (assigned qnil). mTableRef is then deleted and assigned to NULL.

Public Methods

tqfDAMStatementObj::tqfDAMStatementObj()
tqfDAMStatementObj::tqfDAMStatementObj()

The statement base class constructor.

tqfDAMStatementObj::~tqfDAMStatementObj()
tqfDAMStatementObj::~tqfDAMStatementObj()

This is the statement base class destructor.

tqfDAMStatementObj::addRef()
void tqfDAMStatementObj::addRef()

This method increments the number of references to this object instance.

tqfDAMStatementObj::releaseRef()
void tqfDAMStatementObj::releaseRef()

This decrements the reference count of this object instance. If the reference count reaches zero then the base class is destroyed.

tqfDAMStatementObj::methodCall()
qbool tqfDAMStatementObj::methodCall(EXTCompInfo* pEci)

The external component library calls this method every time a DAM function is invoked within Omnis Studio. The pEci parameter will contain the information about the invoked function and the base class will call the appropriate method based on this information.

- **pEci** – The external component information class that contains information regarding the method to be called.

tqfDAMStatementObj::propertySupport()
qlong tqfDAMStatementObj::propertySupport(LPARAM pMessage, WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

This is used to support the DAM statement properties. This will determine whether a property is read only as well as read and write the value of the property if permitted.

- **pMessage** – The type of property request. This can be ECM_PROPERTYCANASSIGN, ECM_SETPROPERTY or ECM_GETPROPERTY
- **wParam** - This is set to ECM_WPARAM_PROPBUTTON when pMessage is ECM_SETPROPERTY and if the Property Manager popup button was pressed to set the property. For example, a file name property may wish to use a file open dialog if the popup button was pressed. Otherwise it is not used.
- **lParam** - Not used
- **eci** - The external component information class that contains information regarding the property to be processed.

tqfDAMStatementObj::prepare()
qbool tqfDAMStatementObj::prepare(EXTfldval& pSQLText, EXTCompInfo *pEci)

This method prepares a SQL Statement. If the pSQLText parameter is empty then the statement in the SQL Buffer is processed.

- **pSQLText** – The SQL statement to be prepared.
- **pEci** - The external component information class that contains information regarding the property to be processed.

tqfDAMStatementObj::workerPrepare()

qbool tqfDAMStatementObj::workerPrepare(EXTfldval& pSQLText, EXTqlist *pParams, qshort *pBindIndexes)

Prepares a SQL Statement for use with a worker delegate object. workerPrepare() is called from StatementWorkerDelegate::dRun() and is re-entrant to allow for multiple statement execution. workerPrepare() parses pSQLText for bind variable markers. Each marker position is used to access the pBindIndexes array. In turn, pBindIndexes contains column numbers used to access pParams; thus pSQLText can refer to pParams column names in a partial or non-linear order.
workerPrepare() then character maps the SQL text, calls dProcessPrepareParams() followed by dPrepare() and bindInputBuffers(), returning qtrue only if these succeed.

- **pSQLText** – A SQL statement to be executed one or more times (dependant on the number of rows in pParams)
- **pParams** – a list or row variable that was supplied during $init(), containing the bind variables for this SQL statement
- **pBindIndexes** - The Worker base class normally creates the pBindIndexes array during $init() by matching bind variable names in the SQL text against column names in the supplied list or row variable. This is stored (in mBindIndexes) and used here

tqfDAMStatementObj::execute()

qbool tqfDAMStatementObj::execute(EXTCompInfo *pEci)

This method executes the previously prepared statement.

- **pEci** - The external component information class that contains information regarding the property to be processed.

tqfDAMStatementObj::execdirect()

qbool tqfDAMStatementObj::execdirect(EXTfldval& pSQLText, EXTCompInfo *pEci)

This method directly executes a SQL Statement. If the pSQLText parameter is empty then the statement in the SQL Buffer is processed.

- **pSQLText** – The SQL statement to be prepared.
- **pEci** - The external component information class that contains information regarding the property to be processed.

tqfDAMStatementObj::prepareExec()

qbool tqfDAMStatementObj::prepareExec(EXTfldval& pSQLText, EXTCompInfo *pEci)

If the DAM elects not to implement direct execution then this simulates an execdirect. This method prepares a SQL Statement. If the pSQLText parameter is empty then the statement in the SQL Buffer is processed. This is then immediately followed by the execution of the statement.

- **pSQLText** – The SQL statement to be prepared.
- **pEci** - The external component information class that contains information regarding the property to be processed.

tqfDAMStatementObj::describe()

qbool tqfDAMStatementObj::describe()

Called after a result set is produced from a statement execution, this method queries the datatypes and specifications of the columns in the result sets.

mColumnCount governs the number of result set columns and dDescribeParam() is responsible for obtaining a description of each result set column.

Output parameters are handled in one of three ways:

If the bind type of an output parameter is set to kBindOmnis, describe() assigns a buffer size of zero implying that the implementation will be writing directly into Omnis field values (EXTfldvals).

If the parameter's field length is greater than the session object's $lobthreshold, chunking of the parameter is enabled and the data will be retrieved using the chunking mechanism. $lobchunksize will be used for the parameter's data buffer.

Otherwise the field length (assigned in dSetResults()) is used to set the parameter's buffer length and the data will be retrieved during dFetch().

When describe() has finished, it calls the implementation's dProcessOutputColumns() which allows the implementation to inspect and modify any output parameter descriptions.

tqfDAMStatementObj::fetch()
eFetchStatus tqfDAMStatementObj::fetch(qlong pRowCount, qshort pAppend)

This method fetches a set of result rows into current list/row.

- **pRowCount** – The number of rows to be fetched
- **pAppend** – qtrue if results are to be appended to current list

tqfDAMStatementObj::clearAll()
qbool tqfDAMStatementObj::clearAll()

This method clears the statement base class and derived members.

tqfDAMStatementObj::clearTran()
qbool tqfDAMStatementObj::clearTran()

This method clears the statement base members after a transaction.

tqfDAMStatementObj::drop()
qbool tqfDAMStatementObj::drop()

This method clears and drops any implementation specific statement and cursor constructs.

tqfDAMStatementObj::close()
qbool tqfDAMStatementObj::close()

This method closes the statement or cursor and resets statement state to kStatePrepared.

tqfDAMStatementObj::setName()
void tqfDAMStatementObj::setName(str255& pStatementName)

This sets the name of statement object.

- **pStatementName** – The name of the statement.

tqfDAMStatementObj::getName()

void tqfDAMStatementObj::getName(str255& pStatementName)

This returns the name of the statement object.

- **pStatementName** – A reference to a variable in which to return the statement name.

tqfDAMStatementObj::setSession()

void tqfDAMStatementObj::setSession(tqfDAMbaseObj *pSession)

This function associates the statement object with the specified session object.

- **pSession** – The session object.

tqfDAMStatementObj::getSession()

tqfDAMbaseObj *tqfDAMStatementObj::getSession()

This function returns the session object associated with the statement object.

tqfDAMStatementObj::removeSession()

void tqfDAMStatementObj::removeSession()

This method removes the statement from the session's statement list and associates the statement with a null session. This is only used when destroying the statement object.

tqfDAMStatementObj::splitTableName()

void tqfDAMStatementObj::splitTableName(str255 &pDatabaseStr, str255 &pOwnerStr, str255 &pTableName)

This method takes the value in pTableName and if necessary splits the string in the form table, or owner.table or database.owner.table, into the three constituent strings returning any part that is absent as an empty string.

- **pDatabaseStr** – The String to return the database part of the table name.
- **pOwnerStr** – The string to return the owner part of the table name.
- **pTableName** – The string that initially contains the full table name in the form table, or owner.table or database.owner.table and returns just the table name.

tqfDAMStatementObj::clearParams()

void tqfDAMStatementObj::clearParams(DAMDataPtr pDAMData)

This function allows the subclass to make any dam specific clean up operations on the parameter objects.

- **pDAMData** – The DAMData object containing the parameters

tqfDAMStatementObj::rpcDefnToBindVars()

void tqfDAMStatementObj::rpcDefnToBindVars(DAMrpcDefn *pRpcDefn, EXTCompInfo *pEci)

This function creates a bind variables array from the RPC definition.

- **pRpcDefn** – The DAMrpcDefn object that contains the RPC definition.
- **pEci** - The external component information class that contains information regarding the allocation of bind variables.

tqfDAMStatementObj::storeRpcOutputParameters()

qbool tqfDAMStatementObj::storeRpcOutputParameters(DAMrpcDefn *pRpcDefn, EXTCompInfo *pEci)

This function returns the RPC return value and any output parameters to the Omnis application.

- **pRpcDefn** - The DAMrpcDefn object that contains the RPC definition.
- **pEci** - The external component information class that contains information used for returning output parameters to Omnis.

tqfDAMStatementObj::getState()

eStatementState tqfDAMStatementObj::getState()

This function returns the current state of the statement object. Result can be kStateClear, kStatePrepared, kStateExecuted, kStateFetching or kStateExecDirect.

tqfDAMStatementObj::getStripSpaces()

qbool tqfDAMStatementObj::getStripSpaces()

This method returns the value of the statement object's $stripspaces property, i.e. whether or not spaces will be stripped from character data being returned to Omnis.

tqfDAMStatementObj::setStripSpaces()

void tqfDAMStatementObj::setStripSpaces(qbool pDoStrip)

This function sets the statement object's $sqlstripspaces property to pDoStrip. If set to qtrue spaces will be stripped from character values being returned to Omnis. If set to qfalse no stripping will take place.

- **pDoStrip** – The value of stripspaces.

tqfDAMStatementObj::setFetchList()

void tqfDAMStatementObj::setFetchList(EXTqlist *pList, qbool isRow, qbool clear)

This function sets the list in which SQL result sets are to be returned.

- **pList** – The list in which results are to returned
- **isRow** – qtrue if list is a row variable
- **clear** – qtrue if the current list contents are to be deleted

tqfDAMStatementObj::getFetchList()
EXTqlist *tqfDAMStatementObj::getFetchList()

This method returns the current fetch list.

tqfDAMStatementObj::getFetchList()
qbool tqfDAMStatementObj::getFetchList(EXTqlist *pList)

This function returns the current fetch list as a parameter and returns qtrue if the list is a row.

- **pList** – A reference to the variable in which the list is to be returned.

tqfDAMStatementObj::isFetchListARow()
qbool tqfDAMStatementObj::isFetchListARow()

This function returns qtrue if the current fetch list is a row.

tqfDAMStatementObj::setCompInfo()
void tqfDAMStatementObj::setCompInfo(EXTCompInfo *pCompInfo)

This function associates the supplied external component information object with the statement object.

- **pCompInfo** - The external component information class that contains information regarding the current function.

tqfDAMStatementObj::getCompInfo()
EXTCompInfo *tqfDAMStatementObj::getCompInfo()

This function returns the current external component information object.

tqfDAMStatementObj::cancel()
void tqfDAMStatementObj::cancel()

For use by worker delegate objects. Sets the mCancelled protected member to qtrue, indicating that the delegate worker object should abort when it next tests this attribute.

Protected Methods

tqfDAMStatementObj::setError()
void tqfDAMStatementObj::setError(qlong pErrorCode)

This function sets the error code in the statement object. See DAM Error Constants

- **pErrorCode** – The error code to be set in the statement object

tqfDAMStatementObj::setupInputVars()
qbool tqfDAMStatementObj::setupInputVars(eBindType pBindType)

This function creates a new set of DAMParam objects based on the bind variables for a SQL statement. It calls mSession→getTypeTable() to obtain the implementation's data type table.

setupInputVars() uses the type table to obtain, the C-type, scale, sendType and bufferLength based on each bind variable's type and subtype.

- pBindType – The bind type constant specifying when binding should take place. See Constants for further details.

tqfDAMStatementObj::setBufferValues()
qbool tqfDAMStatementObj::setBufferValues(qbool pRpc)

This function copies the values from the Omnis fldvals into the DAMParam object buffers.

- **pRpc** – qtrue if the current SQL statement is an RPC call

tqfDAMStatementObj::bindInputBuffers()
qbool tqfDAMStatementObj::bindInputBuffers(eBindType pBindType)

This method binds each input buffer to its database bind marker. See dBindParameter() for implementation-specific notes.

- **pBindType** –Constant specifying when binding should take place. See Constants for further details. Only bind variables matching pBindType will be processed by this call.

tqfDAMStatementObj::sendInputValues()
qbool tqfDAMStatementObj::sendInputValues()

This function sends data to the database in chunks if chunking is required.

tqfDAMStatementObj::setOmnisColumnData()
qbool tqfDAMStatementObj::setOmnisColumnData(DAMParamPtr pParam, EXTfldval &pOmnisColumn, qshort pParmNumber)

This function moves data from the bound output buffers into the Fetch list after a fetch.

- **pParam** – The current parameter containing SQL data
- **pOmnisColumn** – A EXTfldval object for the column in the list
- **pParmNumber** – The parameter number. 1 refers to the first parameter.

tqfDAMStatementObj::setNativeError()
qbool tqfDAMStatementObj::setNativeError()

Calls dGetNativeError() and assigns the resulting error code and error text to mErrorInfo.
If dGetNativeError() returns kErrorPending, the mErrorInfo.mNativeErrorPending is set to expect another error.

setNativeError() returns qfalse if dGetNativeError() returns kErrorFailed.

tqfDAMStatementObj::readyForFetch()

qbool tqfDAMStatementObj**::**readyForFetch(DAMstaAddRowMode pAddRowMode= eDAMaddRowNormal)

readyForFetch() is called after a SQL statement has been executed and any chunked input values have been processed. It calls to the implementation's dSetResults() method which is responsible for establishing whether a result set was generated by the SQL statement.

If dSetResults() sets mColumnCount > 0, mResultsPending is set to qtrue.

mAddRowMode is also set to the value of pAddRowMode. This governs how the base class will process each row in the result set.

In automatic transaction mode, the session is also committed at this point.

- **pAddRowMode** – The type of result set being described. This may be eDAMaddRowTables, eDAMaddRowColumns, eDAMaddRowIndexes, eDAMaddRowRpcProcedures or eDAMaddRowRpcParameters to indicate that static column descriptions should be returned. eDAMaddRowNormal indicates that the result set description will be based on the result of a previously-executed SELECT statement, and is the default option.

tqfDAMStatementObj::getErrorText()

qbool tqfDAMStatementObj::getErrorText(str255 &pErrorText)

getErrorText() reads an internal error message from the Omnis core resources. The RESgetOmnisDAT() callback is used to obtain a handle on the core's resource strings.

The statement object's mErrorInfo→getErrorCode() is used to derive the resource number for the error text.

Protected Virtual Methods

Unless otherwise stated, these are all pure-virtual functions meaning that they must be implemented by the subclass. Non-pure virtual functions have a default implementation, so overriding them is optional.

tqfImpStatementObj::dPrepare()

qbool tqfImpStatementObj::dPrepare(qchar *pSQLText, qlong pSQLTextLen)

This function prepares the SQL statement as required by the database specific client. Depending on the database the SQL statement is checked for syntax errors and any character substitution is performed. If the $usecursors property is kTrue and the database client does not automatically support server based cursors then the SQL statement should be a SQL SELECT statement. This method will generate a SQL DECLARE statement with the text provided if server based cursors are not implemented by the database client. The cursor name for the SQL DECLARE will be the name of the statement object. A successful call will result in the statements state being set to kStatePrepared. This call will fail if the DAM objects state is $loggedoff. This call will implicitly issue a $clear() call if the $state property is not kStateClear. Therefore, any active statement, cursor or result set defined on this object is cleared.

- **pSQLText** – Buffer containing the SQL statement to be prepared.
- **pSQLTextLen** – The length of the SQL statement

Example:

```
qbool tqfODBCStatementObj::dPrepare(qchar *pSQLText, qlong pSQLTextLen)
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = SQLPrepare(mStatementHandle, pSQLText, pSQLTextLen);
  if(status == SQL_SUCCESS)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpStatementObj::dExecute()

qbool tqfImpStatementObj::dExecute()

This function executes or re-executes the statement command previously prepared. If the statement $status property is kStateClear then this call should fail. Re-executing with $resultsPending set to kTrue should clear the current result set and close any open cursor.

It should be possible to re-execute remote procedures with this call.

Example:

```
qbool tqfODBCStatementObj::dExecute()
{
  RETCODE status = SQL_ERROR;
  qbool rtnStatus = qtrue;
  if(mStatementHandle != NULL)
  {
    tqfODBCDAMObjPtr session = (tqfODBCDAMObjPtr)getSession();
    SQLSetStmtOption(mStatementHandle, SQL_QUERY_TIMEOUT, session->getQueryTimeOut());
    status = SQLExecute(mStatementHandle);
  }
  if(status == SQL_SUCCESS)
  {
    // Do any post execute processing
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dExecDirect()

qbool tqfImpStatementObj::dExecDirect(qchar *pSQLText, qlong pSQLTextLen)

This function will directly execute a SQL statement if allowed by the client. If direct execution is supported without a separate prepare stage then the SQL statement is executed immediately, otherwise this invokes a separate $prepare and $execute. Note that if direct execution is supported certain operations may fail and re-execution may not be available. If the statement $state property is kStatePrepared then the previously prepared statement should be cleared.

- **pSQLText** – Buffer containing the SQL statement to be prepared.
- **pSQLTextLen** – The length of the SQL statement

Example:

```
qbool tqfODBCStatementObj::dExecDirect(qchar *pSQLText, qlong pSQLTextLen)
{
  RETCODE status = SQL_ERROR;
  qbool rtnStatus = qtrue;
  if(mStatementHandle != NULL)
  {
    tqfODBCDAMObjPtr session = (tqfODBCDAMObjPtr)getSession();
    SQLSetStmtOption(mStatementHandle, SQL_QUERY_TIMEOUT, session->getQueryTimeOut());
    status = SQLExecDirect(mStatementHandle, pSQLText, pSQLTextLen);
  }
  if(status == SQL_SUCCESS)
  {
    // Do any post execute processing
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dDescribeParam()

qbool tqfImpStatementObj::dDescribeParam(qshort pParamNumber, DAMParamPtr pParam)

This function should populate the DAMParam with database specific information about a column in a result set such as SQL Type and length.

- **pParamNumber** – The parameter number
- **pParam** – The structure containing all the DAMParam objects.

Example:

```
qbool tqfODBCStatementObj::dDescribeParam(qshort pParamNumber, DAMParamPtr pParam)
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  qchar columnName[MAX_COLUMN_NAME_LEN];
  qshort nameLength, dataType, decimalDigits, nullable;
  qlong columnSize;
  status = SQLDescribeCol(mStatementHandle, pParamNumber, columnName, MAX_COLUMN_NAME_LEN, &nameLength, &dataT
  if((status == SQL_SUCCESS) || (status == SQL_SUCCESS_WITH_INFO))
  {
    if(nameLength < MAX_COLUMN_NAME_LEN)
    {
      columnName[nameLength] = '\0';
    }
    str255 name(columnName);
    pParam->setName(name);
    if(nullable != SQL_NO_NULLS)
      pParam->setNullable(qtrue);
    // Convert and transfer data to pParam here.
    rtnStatus = qtrue;
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dSetResults()

qbool tqfImpStatementObj::dSetResults()

This function is called by the base class to get meta data information regarding the results set. This information includes column count, row count and rows affected.

Example:

```
qbool tqfODBCStatementObj::dSetResults()
{
  RETCODE status;
  qbool rtnStatus = qfalse;
  status = SQLNumResultCols(mStatementHandle, &mColumnCount);
  if((status == SQL_SUCCESS) || (status == SQL_SUCCESS_WITH_INFO))
  {
    status = SQLRowCount(mStatementHandle, &mRowsAffected);
    if(mColumnCount != 0)
      mRowCount = mRowsAffected;
  }
  if((status == SQL_SUCCESS) || (status == SQL_SUCCESS_WITH_INFO))
```

```
  {
    rtnStatus = qtrue;
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dBindColumn()

qbool tqfImpStatementObj::dBindColumn(qshort pParamNum, DAMParamPtr pParam)

This function binds a result set column by sending various attributes(pointers) from the DAMParam object to the database server. This typically includes the buffer address for the data and for the resulting data length as well as the server data type (for conversion purposes).
The database server populates these during a subsequent dFetch(). pParamNum is the 1-based column number of the pending result set.

Example:

```
qbool tqfODBCStatementObj::dBindColumn(qshort pParamNum, DAMParamPtr pParam)
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = mODBCSession->sqlBindCol(mStatementHandle, pParamNum, (SSHORT)pParam->getServerType(), (PTR)pParam-
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpStatementObj::dFetch()

eFetchStatus tqfImpStatementObj::dFetch()

This function is called by the base class to fetch a batch of rows from a database results set and returns the fetch status. If there are no more rows left in the current set then kFetchFinished is returned otherwise kFetchOk is returned. If there are no more result sets pending this statements $resultsPending flag is set to kFalse. An attempt to fetch when $resultsPending is kFalse should result in a return status of kFetchError.

The fetch status can be kFetchOk, kFetchFinished, kFetchError or kFetchRowAdded. See fetch constants for more details.

Example:

```
eFetchStatus tqfODBCStatementObj::dFetch()
{
  eFetchStatus rtnStatus = kFetchError;
  RETCODE status;
  mNoRows = 0;
  status = SQLFetch(mStatementHandle);
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
  {
    rtnStatus = kFetchOk;
    mNoRows = 1;
  }
  else if(status == SQL_NO_DATA_FOUND)  // end of result set
    rtnStatus = kFetchFinished;
  return rtnStatus;
}
```

tqfImpStatementObj::dPreFetch()
qbool tqfImpStatementObj::dPreFetch(qlong &pRowCount, qshort &pAppend)

dPreFetch() allows the implementation to intercede at the start of the fetch, changing the pRowCount and pAppend parameters as well as any other accessible attributes if necessary.

dPreFetch() is not a pure virtual function, so implementation in the derived statement object is optional. The default implementation simply returns qtrue.

Example:

```
qbool tqfAmazonStatementObj::dPreFetch(qlong &pRowCount, qshort &pAppend)
{
  do {
    EXTqlist *fetchList= getFetchList();
    if (mResponse == NULL) break;
    if ((fetchList) && (mResponse->type != SDB_R_DOMAIN_LIST))
      fetchList->clear(listVlen); //clear list contents and definition (forces a redefine)
    if (mResponse->type == SDB_R_DOMAIN_LIST)
      mAddRowMode = eDAMaddRowTables;
    else if (mResponse->type == SDB_R_DOMAIN_METADATA)
      mAddRowMode = eDAMaddRowColumns;
    else
    { //items and attributes
      pRowCount = 2; //ensures that dFetch gets called twice
    }
    mNoRows = 1; //tell base class how many rows to fetch
  } while (0);
  return qtrue;
}
```

tqfImpStatementObj::dPostFetch()
qbool tqfDAMStatementObj::dPostFetch(qbool pIsRow, eFetchStatus &pFetchStatus)

dPostFetch() allows the implementation to intercede at the end of the fetch, changing the fetch status and modifying output parameter buffers if necessary. dPostFetch() is not a pure virtual function, so implementation in the derived statement object is optional. The default implementation simply returns qtrue.

Example:

```
bool tqfAmazonStatementObj::dPostFetch(qbool pIsRow, eFetchStatus &pFetchStatus)
{
  if (pIsRow == qtrue)
    pFetchStatus = kFetchFinished;
  return qtrue;
}
```

tqfImpStatementObj::dMoreResults()
qbool tqfImpStatementObj::dMoreResults()

This call should return qtrue if there are further result sets to be processed.

Example:

```
qbool tqfODBCStatementObj::dMoreResults()
{
  qbool rtnStatus = qfalse;
  RETCODE status;
  status = SQLMoreResults(mStatementHandle);
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  return rtnStatus;
}
```

tqfImpStatementObj::dCanAddRow()

qbool tqfImpStatementObj::dCanAddRow()

Called by the base class to determine whether the last fetched row can be added to the fetch list. If qfalse is returned the base class discards the row and proceeds to fetch the next row.

Example:

```
qbool tqfODBCStatementObj::dCanAddRow()
{
  if (mAddRowMode != eDAMaddRowIndexes && mAddRowMode != eDAMaddRowRpcParameters) return qtrue; // rmm3693
  // Doing $indexes or $rpcparameters
  if (eDAMaddRowRpcParameters == mAddRowMode)
  {
    DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_COLUMN_PASS_TYPE);
    EXTfldval fval;
    param->setFldVal(fval, mBatchRow);
    qlong passType = fval.getLong();
    return SQL_PARAM_INPUT == passType || SQL_PARAM_OUTPUT == passType || SQL_PARAM_INPUT_OUTPUT == passType |
  }
  return qtrue;
}
```

tqfImpStatementObj::dConvertParam()

qbool tqfImpStatementObj::dConvertParam(DAMParamPtr pParam, qlong pBatchRow)

This function is called by the base class and should convert database specific types to Omnis equivalent C types. This occurs when returning data from a database server that requires a modification before being returned to Omnis.

- **pParam** – The DAMParam object containing parameter information.
- **pBatchRow** – The row number to be converted.

Example:

```
qbool tqfODBCStatementObj::dConvertParam(DAMParamPtr pParam, qlong pBatchRow)
{
  eCType cType = pParam->getCType();
  qbyte *buffer = pParam->getParamBuffer();
  qlong *ParamLen = pParam->getParamLenBuffer();
  buffer = &(buffer[(pBatchRow-1)*bufferLen]);
  ParamLen = &(ParamLen[pBatchRow-1]);
  qbool rtnStatus = qtrue;
  switch(cType)
  {
    case omDate:
```

```
    {
      datestamptype omnisDS;
      DATE_STRUCT tempDate;
      MEMmovel(buffer, &tempDate, sizeof(DATE_STRUCT));
      omnisDS.mYear = (qshort)tempDate.year;
      omnisDS.mMonth = (qbyte)tempDate.month;
      omnisDS.mDay = (qbyte)tempDate.day;
      omnisDS.mDateOk = (qbool)qtrue;
      omnisDS.mTimeOk = (qbool)qfalse;
      omnisDS.mSecOk = (qbool)qfalse;
      omnisDS.mHunOk = (qbool)qfalse;
      MEMmovel(&omnisDS, buffer, sizeof(datestamptype));
      *ParamLen = sizeof(datestamptype);
      break;
    }
    case omTime:
    {
      //Perform Omnis Time conversion here
    }
    //Perform other type conversions here
    default:
    {
      rtnStatus = qfalse;
    }
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dClearStatement()
qbool tqfImpStatementObj::dClearStatement()

This function is called by the base class to clear all SQL activity from the statement . This method should not drop the statement.

Example:

```
qbool tqfODBCStatementObj::dClearStatement()
{
  qbool rtnStatus = qtrue;
  RETCODE status;
  tqfODBCDAMObjPtr session = (tqfODBCDAMObjPtr)mSession;
  if( (session == NULL) || (session->getConnection() == SQL_NULL_HDBC) )
    mStatementHandle = SQL_NULL_HSTMT;
  if(mStatementHandle != SQL_NULL_HSTMT)
  {
    status = SQLFreeStmt(mStatementHandle, SQL_CLOSE);
    if(status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
      rtnStatus = qfalse;
    status = SQLFreeStmt(mStatementHandle, SQL_UNBIND);
    if(status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
      rtnStatus = qfalse;
    status = SQLFreeStmt(mStatementHandle, SQL_RESET_PARAMS);
    if(status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
      rtnStatus = qfalse;
  }
  // Clear any other statement members here.
 return rtnStatus;
}
```

tqfImpStatementObj::dDropStatement()
qbool tqfImpStatementObj::dDropStatement()

This function should drop the statement and free any resources associated with it.

Example:

```
qbool tqfODBCStatementObj::dDropStatement()
{
  qbool rtnStatus = qtrue;
  RETCODE status = SQL_ERROR;
  tqfODBCDAMObjPtr session = (tqfODBCDAMObjPtr)mSession;
  if((session != NULL) && (session->getConnection() != SQL_NULL_HDBC) && (mStatementHandle != SQL_NULL_HSTMT))
  {
    status = SQLFreeStmt(mStatementHandle, SQL_DROP);
    if(status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    {
      mErrorInfo.setError();
      rtnStatus = qfalse;
    }
  }
  mStatementHandle = SQL_NULL_HSTMT;
  mErrorInfo.setStatementHandle(SQL_NULL_HSTMT);
  return rtnStatus;
}
```

tqfImpStatementObj::dCloseStatement()
qbool tqfImpStatementObj::dCloseStatement()

Thus function closes the database cursor associated with statement (if one was defined) and discard all pending results. The application can reopen this cursor later by re-executing the SQL statement again with the same or different parameter values.

Example:

```
qbool tqfODBCStatementObj::dCloseStatement()
{
  qbool rtnStatus = qtrue;
  RETCODE status;
  tqfODBCDAMObjPtr session = (tqfODBCDAMObjPtr)mSession;
  if( (session != NULL) && (session->getConnection() == SQL_NULL_HDBC) )
    mStatementHandle = SQL_NULL_HSTMT;
  if(mStatementHandle != SQL_NULL_HSTMT)
  {
    status = SQLFreeStmt(mStatementHandle, SQL_CLOSE);
    if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
      status = SQLFreeStmt(mStatementHandle, SQL_UNBIND);
    if(status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
      rtnStatus = qfalse;
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dClearParams()
void tqfImpStatementObj::dClearParams(DAMDataPtr pDAMData)

This function allows the DAM implementation to clear any parameter memory it previously allocated.

- **pDAMData** – The parameter group for current input/output parameters

Example:

```
void tqfODBCStatementObj::dClearParams(DAMDataPtr pDAMData)
{
  DAMParamPtr param;
  qshort colNum = 1,maxCol;
  param = pDAMData->getParam(colNum);
  maxCol = pDAMData->numParams();
  while((param)&&(colNum <= maxCol))
  {
    qlong *addr = (qlong *) param ->getDamStorage();
    if(*addr != NULL)
    {
      MEMfree((qbyte *)*addr);
      *addr = NULL;
      addr++;
    }
    colNum++;
    param = pDAMData->getParam(colNum);
  }
}
```

tqfImpStatementObj::dTables()
qbool tqfImpStatementObj::dTables(qlong pTableType, str255 &pOwner)

This function executes database specific code to build a result set containing information about the tables and views on the database server. Depending on the DAM object there may be additional types of table that can be specified, e.g. system tables. The $fetch() call is used to return the result set rows. This call will clear any previous pending results or statements.

- **pTableType** – The type of tables to be included in the results. This can be kStatementServerAll, kStatementServerTable or kStatementServerView. See Table type constants for further details.
- **pOwner** – The owner of the table. Leave blank to request all tables

Example:

```
qbool tqfODBCStatementObj::dTables(qlong pTableType, str255 &pOwner)
{
  if (!allocStatement()) return qfalse;
  qchar *owner = (pOwner.length() ? pOwner.cString() : 0);
  qshort ownerLen = pOwner.length() ? SQL_NTS : 0;
  qchar *tableType;
  qshort tableTypeLen = SQL_NTS;
  if (pTableType == kStatementServerView)
    tableType = "'VIEW'";
  else if (pTableType == kStatementServerTable)
    tableType = "'TABLE'";
  else
    tableType = "'TABLE','VIEW'";
  RETCODE status = SQLTables(mStatementHandle, 0, 0, owner, ownerLen, 0, 0, tableType, tableTypeLen);
  if (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    return qfalse;
  return qtrue;
}
```

tqfImpStatementObj::dColumns()

qbool tqfImpStatementObj::dColumns(str255 &pTableName)

This function executes database specific code to build a result set containing information about the columns in a table.

- **pTableName** – The table name

Example:

```
qbool tqfODBCStatementObj::dColumns(str255 &pTableName)
{
  tqfODBCDAMObjPtr currSession = (tqfODBCDAMObjPtr)getSession();
  // pTableName can be in the form tableName, owner.TableName, or database.owner.TableName
  str255 ownerStr, databaseStr;
  splitTableName(databaseStr, ownerStr, pTableName);
  qchar *database = databaseStr.length() ? databaseStr.cString() : 0);
  qshort databaseLen = databaseStr.length() ? SQL_NTS : 0;
  qchar *owner = ownerStr.length() ? ownerStr.dataPtr() : 0);
  qshort ownerLen = ownerStr.length() ? SQL_NTS : 0;
  qchar *tableName = (pTableName.length() ? pTableName.dataPtr() : 0);
  qshort tableNameLen = pTableName.length() ? SQL_NTS : 0;
  RETCODE status = SQLColumns(mStatementHandle, database, databaseLen, owner, ownerLen, tableName, tableNameLe
  if (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    return qfalse;
  return qtrue;
}
```

tqfImpStatementObj::dIndexes()

qbool tqfImpStatementObj::dIndexes(str255 &pTableName, qlong pIndexType)

This function executes database specific code to build a result set containing information about the unique and non-unique indexes in a table.

- **pTableName** – The table name
- **pIndexType** – The index type. This can be kStatementIndexUnique, kStatementIndexNonUnique and kStatementIndexAll. See Index type constants for further information.

Example:

```
qbool tqfODBCStatementObj::dIndexes(str255 &pTableName, qlong pIndexType)
{
  tqfODBCDAMObjPtr currSession = (tqfODBCDAMObjPtr)getSession();
  // pTableName can be in the form tableName, owner.TableName, or database.owner.TableName
  str255 ownerStr, databaseStr;
  splitTableName(databaseStr, ownerStr, pTableName);
  qchar *database = databaseStr.length() ? databaseStr.cString() : 0);
  qshort databaseLen = databaseStr.length() ? SQL_NTS : 0;
  qchar *owner = ownerStr.length() ? ownerStr.dataPtr() : 0);
  qshort ownerLen = ownerStr.length() ? SQL_NTS : 0;
  qchar *tableName = (pTableName.length() ? pTableName.dataPtr() : 0);
  qshort tableNameLen = pTableName.length() ? SQL_NTS : 0;
  if (pIndexType == kStatementIndexUnique)
    unique = SQL_INDEX_UNIQUE;
  else if (pIndexType == kStatementIndexNonUnique)
```

```
    unique = SQL_INDEX_ALL;
  RETCODE status = SQLStatistics(mStatementHandle, database, databaseLen, owner, ownerLen, tableName, tableNam
  if (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    return qfalse;
  return qtrue;
}
```

tqfImpStatementObj::dTablesGetOwner()
void tqfImpStatementObj::dTablesGetOwner(EXTfldval &pColVal)

This function should extract the table owner from the results set after a $tables() call.

- **pColVal** – The Omnis field value corresponding to the Owner column in the $tables() result set.

Example:

```
void tqfODBCStatementObj::dTablesGetOwner(EXTfldval &pColVal)
{
  //DODBC_TABLES_TABLE_SCHEM defines the result set column number that contains the schema name
  DAMParamPtr param = mOutputParams.getParam(DODBC_TABLES_TABLE_SCHEM);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dTablesGetTableName()
void tqfImpStatementObj::dTablesGetTableName(EXTfldval &pColVal)

This function should extract the table name from the results set after a $tables() call.

- **pColVal** - The Omnis field value corresponding to the Table Name column in the $tables() result set.

Example:

```
void tqfODBCStatementObj::dTablesGetTableName(EXTfldval &pColVal)
{
  // DODBC_TABLES_TABLE_NAME defines the result set column number that contains the table name
  DAMParamPtr param = mOutputParams.getParam(DODBC_TABLES_TABLE_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dTablesGetTableType()
void tqfImpStatementObj::dTablesGetTableType(EXTfldval &pColVal)

This function extracts the table type from the results set after a $tables() call.

- **pColVal** – The Omnis field value corresponding to the Table Type column in the $tables() result set.

Example:

```
void tqfODBCStatementObj::dTablesGetTableType(EXTfldval &pColVal)
{
  // DODBC_TABLES_TABLE_TYPE defines the result set column number that contains the table type information
  DAMParamPtr param = mOutputParams.getParam(DODBC_TABLES_TABLE_TYPE);
  param->setFldVal(pColVal, mBatchRow);
  // Convert text returned for table type to an integer constant value
  str255 typeString;
  pColVal.getChar(typeString);
  qchar *tableType = typeString.cString();
  preconst tableTypeVal;
  if (!strcmp(tableType, "TABLE"))
    tableTypeVal = preStatementServerTable;
  else if (!strcmp(tableType, "VIEW"))
    tableTypeVal = preStatementServerView;
  else
    tableTypeVal = preBoolUnknown; // Should never occur
  pColVal.setConstant(tableTypeVal);
}
```

tqfImpStatementObj::dTablesGetDescription()
void tqfImpStatementObj::dTablesGetDescription(EXTfldval &pColVal)

This function extracts the table description from the results set after a $tables() call.

- **pColVal** – The Omnis field value corresponding to the Description column in the $tables() result set.

Example:

```
void tqfODBCStatementObj::dTablesGetDescription(EXTfldval &pColVal)
{
  // DODBC_TABLES_REMARKS defines the result set column number that contains the description information
  DAMParamPtr param = mOutputParams.getParam(DODBC_TABLES_REMARKS);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dTablesGetDamInfo()
void tqfImpStatementObj::dTablesGetDamInfo(EXTfldval &pColVal)

This function extracts any DAM specific information about the table from the results set after a $tables() call. This is a non-pure virtual function, meaning that it only needs to be overridden if desired. The default implementation returns an empty row.

- **pColVal** – The destination Omnis field value for any DAM specific information about the table.

Example:

```
void tqfODBCStatementObj::dTablesGetDamInfo(EXTfldval &pColVal)
{
  // Add a row containing a single character column
  EXTqlist *infoRow = new EXTqlist(listVlen); //pColVal will deallocate this
  infoRow->clear(listVlen);
  str15 colSqlText;
  RESloadString(gInstLib, 2135, colSqlText); //e.g. hasindexes
  infoRow->addCol(1,fftCharacter, dpFcharacter, 16, NULL, &colSqlText);
  infoRow->insertRow();
```

```
  EXTfldval columnData; //set column data
  DAMParamPtr param = mOutputParams.getParam(DB_TABLES_HAS_INDEXES);
  char *buffer = (char *)param->getParamBuffer();
  if (buffer[0] == 1)
    columnData.setBool(preBoolTrue);
  else
    columnData.setBool(preBoolFalse);
  infoRow->putColVal(1,1,columnData);
  pColVal.setList(infoRow, qtrue); //transfer ownership to pColVal
  delete infoRow; //delete local copy
}
```

tqfImpStatementObj::dColumnsGetDatabaseOrCatalog()

void tqfImpStatementObj::dColumnsGetDatabaseOrCatalog(EXTfldval &pColVal)

This function should extract the database, schema or catalog from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the database, schema or catalog to which the table belongs.

Example:

```
void tqfODBCStatementObj::dColumnsGetDatabaseOrCatalog(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_TABLE_CAT defines the result set column number that contains the catalog information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_TABLE_CAT);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dColumnsGetOwner()

void tqfImpStatementObj::dColumnsGetOwner(EXTfldval &pColVal)

This function extracts the owner of the table from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the owner of the table.

Example:

```
void tqfODBCStatementObj::dColumnsGetOwner(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_TABLE_SCHEM defines the result set column number that contains the owner information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_TABLE_SCHEM);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dColumnsGetColumnName()

void tqfImpStatementObj::dColumnsGetColumnName(EXTfldval &pColVal)

This function should extract the column name from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the column name.

Example:

```
void tqfODBCStatementObj::dColumnsGetColumnName(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_COLUMN_NAME defines the result set column number that contains the column name information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_COLUMN_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

---

tqfImpStatementObj::dColumnsGetOmnisDataTypeText()

---

void tqfImpStatementObj::dColumnsGetOmnisDataTypeText(EXTfldval &pColVal)

This function should extract the Omnis data type text from the results set after a $columns() call.

- **pColVal** – The destination Omnis field value for the data type text.

Example:

```
void tqfODBCStatementObj::dColumnsGetOmnisDataTypeText(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_DATA_TYPE defines the result set column number that contains the column data type informati
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_DATA_TYPE);
  param->setFldVal(pColVal, mBatchRow);
  mCurColOdbcDataType = (qshort) pColVal.getLong();
  // DODBC_COLUMNS_COLUMN_SIZE defines the result set column number that contains the column size information
  param = mOutputParams.getParam(DODBC_COLUMNS_COLUMN_SIZE);
  param->setFldVal(pColVal, mBatchRow);
  qlong columnSize = pColVal.getLong();
  // DODBC_COLUMNS_DECIMAL_DIGITS defines the result set column number that contains the decimal digits inform
  param = mOutputParams.getParam(DODBC_COLUMNS_DECIMAL_DIGITS);
  param->setFldVal(pColVal, mBatchRow);
  qshort decimalDigits = (qshort) pColVal.getLong();
  // Use a temporary parameter, to allow us to map from the above information to the OMNIS data type
  DAMParam tempParam;
  odbc2Param(&tempParam, mCurColOdbcDataType, columnSize, decimalDigits);
  mCurColType = tempParam.getBufferType();
  mCurColSubType = tempParam.getBufferSubType();
  // Get the OMNIS data type string
  str255 typeText;
  DAMgetDataTypeText(mCurColType, mCurColSubType, tempParam.getFieldLen(), &typeText);
  pColVal.setChar(typeText);
}
```

---

tqfImpStatementObj::dColumnsGetOmnisDataType()

---

void tqfImpStatementObj::dColumnsGetOmnisDataType(EXTfldval &pColVal)

This function should extract the Omnis data type from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the Omnis data type.

Example:

```
void tqfODBCStatementObj::dColumnsGetOmnisDataType(EXTfldval &pColVal)
{
  pColVal.setLong(mCurColType); // mCurColType set in dColumnsGetOmnisDataTypeText()
}
```

tqfImpStatementObj::dColumnsGetOmnisDataSubType()

void tqfImpStatementObj::dColumnsGetOmnisDataSubType(EXTfldval &pColVal)

This function should extract the Omnis data sub type from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the Omnis data sub type.

Example:

```
void tqfODBCStatementObj::dColumnsGetOmnisDataSubType(EXTfldval &pColVal)
{
  pColVal.setLong(mCurColSubType); // mCurColSubType set in dColumnsGetOmnisDataTypeText()
}
```

tqfImpStatementObj::dColumnsGetSqlDataType()

void tqfImpStatementObj::dColumnsGetSqlDataType(EXTfldval &pColVal)

This function should extract the SQL data type from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the SQL data type.

Example:

```
void tqfODBCStatementObj::dColumnsGetSqlDataType(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_TYPE_NAME defines the result set column number that contains the sql type name information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_TYPE_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dColumnsGetLength()

void tqfImpStatementObj::dColumnsGetLength(EXTfldval &pColVal)

This function should extract the column length from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the column length.

Example:

```
void tqfODBCStatementObj::dColumnsGetLength(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_COLUMN_SIZE defines the result set column number that contains the column size information
  qbool retVal;
  switch (mCurColOdbcDataType)
  {
    case SQL_CHAR: case SQL_VARCHAR: case SQL_LONGVARCHAR:
    case SQL_WCHAR: case SQL_WVARCHAR: case SQL_WLONGVARCHAR:
    case SQL_BINARY:case SQL_VARBINARY: case SQL_LONGVARBINARY:
      DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_COLUMN_SIZE);
      param->setFldVal(pColVal, mBatchRow);
      break;
    default:
      pColVal.setLong(0);
  }
  return retVal;
}
```

tqfImpStatementObj::dColumnsGetScale()
void tqfImpStatementObj::dColumnsGetScale(EXTfldval &pColVal)

This function extracts the column scale from the results set after a $columns() call.

- **pColVal** – The destination EXTfldval for the column scale.

Example:

```
void tqfODBCStatementObj::dColumnsGetScale(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_DECIMAL_DIGITS defines the result set column number that contains the scale information
  if (SQL_NUMERIC == mCurColOdbcDataType || SQL_DECIMAL == mCurColOdbcDataType)
  {
    DAMParamPtr param = mOutputParams.getParam(pScaleCol);
    param->setFldVal(pColVal, mBatchRow);
  }
  else pColVal.setLong(0);
}
```

tqfImpStatementObj::dColumnsGetNull()
void tqfImpStatementObj::dColumnsGetNull(EXTfldval &pColVal)

This function should extract the column's null value from the results set after a $columns() call.

- **pColVal** – The destination Omnis field value for the column's null state.

Example:

```
void tqfODBCStatementObj::dColumnsGetNull(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_NULLABLE defines the result set column number that contains the null information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_NULLABLE);
  param->setFldVal(pColVal, mBatchRow);
  qlong nullable = pColVal.getLong();
  preconst nullableConst;
  switch (nullable)
  {
    case SQL_NO_NULLS: nullableConst = preBoolFalse; break;
    case SQL_NULLABLE: nullableConst = preBoolTrue; break;
    default: nullableConst = preBoolUnknown;
  }
  pColVal.setConstant(nullableConst);
}
```

tqfImpStatementObj::dColumnsGetIndex()
void tqfImpStatementObj::dColumnsGetIndex(EXTfldval &pColVal)

This function should extract the column's index state from the results set after a $columns() call.

- **pColVal** – The destination Omnis field value for the column's index state.

Example:

```
void tqfODBCStatementObj::dColumnsGetIndex(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_INDEX_KEY defines the result set column number that contains the index information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_INDEX_KEY);
  param->setFldVal(pColVal, mBatchRow);
  str255 index = pColVal.getChar();
  preconst indexConst;
  if(index.length())
    indexConst = preBoolTrue;
  else
    indexConst = preBoolFalse;
  pColVal.setConstant(indexConst);
}
```

tqfImpStatementObj::dColumnsGetPrimaryKey()
void tqfImpStatementObj::dColumnsGetPrimaryKey(EXTfldval &pColVal)

This function should extract the column's primary key state from the results set after a $columns() call.

- **pColVal** – The destination Omnis field value for the column's primary key state.

Example:

```
void tqfODBCStatementObj::dColumnsGetPrimaryKey(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_PRIMARY_KEY defines the result set column number that contains the primary key information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_PRIMARY_KEY);
  param->setFldVal(pColVal, mBatchRow);
  str255 pk = pColVal.getChar();
  preconst pkConst;
  if(pk.length())
    pkConst = preBoolTrue;
  else
    pkConst = preBoolFalse;
  pColVal.setConstant(pkConst);
}
```

tqfImpStatementObj::dColumnsGetDescription()
void tqfImpStatementObj::dColumnsGetDescription(EXTfldval &pColVal)

This function should extract the column description from the results set after a $columns() call.

- **pColVal** – The destination Omnis field value for the column description.

Example:

```
void tqfODBCStatementObj::dColumnsGetDescription(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_REMARKS defines the result set column number that contains the primary key information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_REMARKS);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dColumnsGetDamInfo()

void tqfImpStatementObj::dColumnsGetDamInfo(EXTfldval &pColVal)

This function should extract any DAM specific information about the column from the results set after a $columns() call. This is a non-pure virtual function, meaning that it only needs to be overridden if desired. The default implementation returns an empty row.

- **pColVal** – The destination Omnis field value for any DAM specific information about the column.

Example:

```
void tqfODBCStatementObj::dColumnsGetDamInfo(EXTfldval &pColVal)
{
  // DODBC_COLUMNS_OTHER defines the result set column number that contains the other relevant information
  DAMParamPtr param = mOutputParams.getParam(DODBC_COLUMNS_OTHER);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetDatabaseOrCatalog()

void tqfImpStatementObj::dIndexesGetDatabaseOrCatalog(EXTfldval &pColVal)

This function should extract the database, schema or catalog from the results set after a $indexes() call.

- **pColVal** – The destination Omnis field value for the database, schema or catalog to which the table belongs.

Example:

```
void tqfODBCStatementObj::dIndexesGetDatabaseOrCatalog(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_TABLE_CAT defines the result set column number that contains the catalog information
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_TABLE_CAT);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetOwner()

void tqfImpStatementObj::dIndexesGetOwner(EXTfldval &pColVal)

This function extracts the owner of the table from the results set after a $indexes call.

- **pColVal** – The destination EXTfldval for the owner of the table.

Example:

```
void tqfODBCStatementObj::dIndexesGetOwner(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_TABLE_SCHEM defines the result set column number that contains the schema information
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_TABLE_SCHEM);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetColumnName()
void tqfImpStatementObj::dIndexesGetColumnName(EXTfldval &pColVal)

This function should extract the column name from the results set after a $indexes() call.

- **pColVal** – The destination field value for the column name.

Example:

```
void tqfODBCStatementObj::dIndexesGetColumnName(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_COLUMN_NAME defines the result set column number that contains the column name informati
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_COLUMN_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetIndexName()
void tqfImpStatementObj::dIndexesGetIndexName(EXTfldval &pColVal)

This function should extract the index name from the results set after a $indexes() call.

- **pColVal** – The destination field value for the index name.

Example:

```
void tqfODBCStatementObj::dIndexesGetIndexName(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_INDEX_NAME defines the result set column number that contains the index name information
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_INDEX_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetUnique()
void tqfImpStatementObj::dIndexesGetUnique(EXTfldval &pColVal)

This function extracts the index's unique attribute from the results set after a $indexes() call

- **pColVal** – The destination field value for the index's unique value.

Example:

```
void tqfODBCStatementObj::dIndexesGetUnique(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_NON_UNIQUE defines the result set column number that contains the unique status.
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_NON_UNIQUE);
  param->setFldVal(pColVal, mBatchRow);
  preconst unique = qbool(pColVal.getLong() == 0) ? preBoolTrue : preBoolFalse;
  pColVal.setConstant(unique);
}
```

tqfImpStatementObj::dIndexesGetColumnPosition()

void tqfImpStatementObj::dIndexesGetColumnPosition(EXTfldval &pColVal)

This function should extract the index column position from the results set after a $indexes() call.

- **pColVal** – The destination field value for the index column position.

Example:

```
void tqfODBCStatementObj::dIndexesGetColumnPosition(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_ORDINAL_POSITION defines the result set column number that contains the index position.
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_ORDINAL_POSITION);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dIndexesGetDamInfo()

void tqfImpStatementObj::dIndexesGetDamInfo(EXTfldval &pColVal)

This function extracts any DAM specific information about the index from the results set after a $indexes() call. This is a non-pure virtual function, meaning that it only needs to be overridden if desired. The default implementation returns an empty row.

- **pColVal** – The destination field value for any DAM specific information about the index.

Example:

```
void tqfODBCStatementObj::dIndexesGetDamInfo(EXTfldval &pColVal)
{
  // DODBC_STATISTICS_OTHER defines the result set column number that contains the other relevant information
  DAMParamPtr param = mOutputParams.getParam(DODBC_STATISTICS_OTHER);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dMetaDataGetSqlDataTypeText()

void tqfImpStatementObj::dMetaDataGetSqlDataTypeText(ffttype pType, str255 &pSqlDataTypeText)

This function is called by the base class to obtain the text suitable for describing the SQL type of a result column of a meta-data method, e.g. $columns(). This only needs to support fftCharacter, fftInteger and fftBoolean.

- **pType** – The Internal ffttype of the variable.
- **pSqlDataTypeText** – The SQL data type text.

Example:

```
void tqfODBCStatementObj::dMetaDataGetSqlDataTypeText(ffttype pType, str255 &pSqlDataTypeText)
{
  qlong resourceNumber = 0;
  switch (pType)
  {
    case fftCharacter: resourceNumber = 2100; break;
    case fftInteger: resourceNumber = 2101; break;
    case fftBoolean: resourceNumber = 2102; break;
  }
  if (!resourceNumber) pSqlDataTypeText[0] = 0;
  else RESloadString(gInstLib, resourceNumber, pSqlDataTypeText);
}
```

tqfImpStatementObj::dGetColumnInfoForResults()

void tqfImpStatementObj::dGetColumnInfoForResults(DAMParamPtr pParam, str255 &pSqlDataTypeText, qlong &pColumnLength)

This function is called by the base class to obtain the SQL data type text, column length and scale after a $results call.

- **pParam** – The DAMParam object containing the column data
- **pSqlDataTypeText** – The SQL data type text
- **pColumnLength** – The column length

Example:

```
void tqfODBCStatementObj::dGetColumnInfoForResults(DAMParamPtr pParam, str255 &pSqlDataTypeText, qlong &pColum
{
  // The SQL data type is stored in the DAM storage member of the param structure
  SSHORT *storage = (SSHORT *)pParam->getDamStorage();
  qshort index = *storage - SQL_TYPE_MIN;
  ODBCTypeInfoPtr *typeTable = getSessionTypes();
  ODBCTypeInfoPtr typeDesc = typeTable[index];
  if (index < 0 || index >= (SQL_TYPE_MAX - SQL_TYPE_MIN + 1))
  {
    qbool foundMatch = qfalse;
    ODBCTypeInfoPtr unknown = typeTable[SQL_TYPE_NULL-SQL_TYPE_MIN];
    while(unknown != NULL && foundMatch == qfalse)
    {
      if (unknown->getType() == *storage)
      {
        foundMatch = qtrue;
        typeDesc = unknown;
      }
      else
        unknown=(ODBCTypeInfoPtr)unknown->getNext();
    }
    if (foundMatch == qfalse) return;
  }
  if (!typeDesc) return;
  typeDesc->getName(pSqlDataTypeText);
  pColumnLength = needsColumnLength(typeDesc->getType()) ? pParam->getFieldLen() : 0;
}
```

tqfImpStatementObj::dResults()

qbool tqfImpStatementObj::dResults(str255 *columnName, ffttype *omnisType, qshort *subType, qlong *fieldLen)

This function allows the DAM implementation to modify the result set returned following a $results() call. This is a non-pure virtual function, meaning that it only needs to be overridden if desired. The default implementation simply returns qtrue.

- **columnName** – The name of the column
- **omnisType** – The fft Type of the column
- **subtype** – The fft sub type of the column
- **fieldLen** - The length of the column

Example:

```
qbool tqfORAbaseStatementObj::dResults(str255 *columnName, ffttype *omnisType, qshort *subType, qlong *fieldLe
{
  if (*omnisType == fftCharacter)
  // do some character specific processing here
return qtrue;
}
```

---

tqfImpStatementObj::dProcessPrepareParams()

---

qbool tqfImpStatementObj::dProcessPrepareParams(DAMDataPtr pInputParams)

This function allows the implementation to provide DAM specific processing to set up input parameters at prepare time such as matching SQL type and whether parameters can be bound.

- **pInputParams** – The DAMData object that contains all of the parameters

Example:

```
qbool tqfODBCStatementObj::dProcessPrepareParams(DAMDataPtr pInputParams)
{
  qbool rtnStatus = qtrue;
  qshort i, noParams = pInputParams->numParams();
  for(i = 1;i <= noParams && rtnStatus == qtrue;i++)
  {
    DAMParamPtr currParam = pInputParams->getParam(i);
    //Perform pre-prepare param specific functionality
    currParam->setBind(kBindExecute);
  }
  return rtnStatus;
}
```

---

tqfImpStatementObj::dProcessExecuteParams()

---

qbool tqfImpStatementObj::dProcessExecuteParams(DAMDataPtr pInputParams)

This function allows the implementation to provide DAM specific processing to set up input parameters at execute time such as matching SQL type and whether parameters can be bound.

- **pInputParams** – The DAMData object that contains all of the parameters

Example:

```
qbool tqfORAbaseStatementObj::dProcessExecuteParams(DAMDataPtr pInputParams)
{
  qbool rtnStatus = qtrue;
  qshort noParams = pInputParams->numParams();
  tqfORACLEbaseObjPtr session = (tqfORACLEbaseObjPtr)getSession();
  for(qshort i = 1;i<=noParams;i++)
  {
    DAMParamPtr currParam = pInputParams->getParam(i);
    //Perform pre-execute param specific functionality
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dBindParameter()

qbool tqfImpStatementObj::dBindParameter(DAMParamPtr currParam, qshort parmNumber)

This function should bind a single input parameter or otherwise prepare the parameter to be sent to the database server.

- **currParam** – The parameter to be bound
- **parmNumber** – The parameter number

Example:

```
qbool tqfODBCStatementObj::dBindParameter(DAMParamPtr currParam, qshort parmNumber)
{
  qbool rtnStatus = qfalse;
  RETCODE status = SQL_ERROR;
  SSHORT valueType;
  SLONG *strLen = (SLONG *)currParam->getParamLenBuffer();
  SSHORT parameterType = (SSHORT)currParam->getServerType();
  SSHORT decimalDigits = (SSHORT)currParam->getServerScale();
  SLONG bufferLength = (SLONG)currParam->getBufferLen();
  ULONG columnSize = (ULONG)currParam->getServerPrecision();
  PTR parameterValuePtr = (PTR)currParam->getParamBuffer();
  SSHORT paramType = SQL_PARAM_INPUT;
  switch (currParam->getParamType())
  {
    case kParameterOutput: paramType = SQL_PARAM_OUTPUT; break;
    case kParameterInputOutput: paramType = SQL_PARAM_INPUT_OUTPUT; break;
  }
  if (rtnStatus == qtrue && allocStatement() == qtrue)
    status = SQLBindParameter(mStatementHandle, parmNumber, paramType, valueType, parameterType, columnSize, d
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
    rtnStatus = qtrue;
  else
    rtnStatus = qfalse;
  return rtnStatus;
}
```

tqfImpStatementObj::dSetInputBufferValue()

qbool tqfImpStatementObj::dSetInputBufferValue(DAMParamPtr pInputParam)

This function allows the DAM implementation to set up database specific information for each input parameter. It requires the implementation to set or convert the data in the parameter's data buffer and/or length buffer. The example below uses dSetInputBufferValue() to convert Omnis datetimes into the format required by the ODBC API.

- **pInputParam** – The current parameter

Example:

```
qbool tqfODBCStatementObj::dSetInputBufferValue(DAMParamPtr pInputParam)
{
  qbool rtnStatus = qtrue;
  eCType cType = pInputParam->getCType();
  EXTfldval *dataVal = pInputParam->getOmnisRef();
  qbyte *paramBuffer = pInputParam->getParamBuffer();
  qlong *paramLenBuffer = pInputParam->getParamLenBuffer();
```

```
  switch(cType)
  {
    case omDate:
    {
      DATE_STRUCT destDate;
      om2ODBCDate((datestamptype *)paramBuffer, &destDate, paramLenBuffer);
      MEMmovel((void *)&destDate, (void *)paramBuffer, *paramLenBuffer);
      break;
    }
    case omTime:
    {
      TIME_STRUCT destDate;
      om2ODBCTime((datestamptype *)paramBuffer, &destDate, paramLenBuffer);
      MEMmovel((void *)&destDate, (void *)paramBuffer, *paramLenBuffer);
      break;
    }
    case omTimeStamp:
    {
      TIMESTAMP_STRUCT destDate;
      om2ODBCTimeStamp((datestamptype *)paramBuffer, &destDate, paramLenBuffer);
      MEMmovel((void *)&destDate, (void *)paramBuffer, *paramLenBuffer);
      break;
    }
    default:
    {
      rtnStatus = qfalse;
    }
  }
  return rtnStatus;
}
```

---

tqfImpStatementObj::dSendParameter()

---

qbool tqfImpStatementObj::dSendParameter(DAMParamPtr pParam, qshort pParamNum)

This function sends a chunk of data to the database server if the data for a column has been chunked.

- **pParam** – The parameter to be bound
- **pParamNum** – The parameter number

Example:

```
qbool tqfODBCStatementObj::dSendParameter(DAMParamPtr pParam, qshort pParamNum)
{
  qbool rtnStatus = qtrue;
  eChunkState chunkState = kChunkOk;
  RETCODE status;
  qlong parameter, chunkLength = pParam->getBufferLen();
  qlong *parmPtr = &parameter;
  PTR putData = (PTR)pParam->getParamBuffer();
  status = SQLPutData(mStatementHandle, putData, chunkLength);
  while( (status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO) && chunkState == kChunkOk)
  {
    chunkState = pParam->readNextChunk(chunkLength);
    if(chunkState != kChunkFailed)
      status = SQLPutData(mStatementHandle, (PTR)putData, chunkLength);
  }
```

```
  if(chunkState == kChunkFailed || (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO))
    rtnStatus = qfalse;
  // get next data at execution parameter
  status = SQLParamData(mStatementHandle, (PTR *)&parmPtr);
  if(status == SQL_ERROR)
    rtnStatus = qfalse;
    return rtnStatus;
}
```

---

tqfImpStatementObj::dProcessOutputColumns()

---

qbool tqfImpStatementObj::dProcessOutputColumns(DAMDataPtr pOutputParams)

This function allows the implementation to provide DAM specific processing to set up output parameters after a result set has been described.

- **pOutputParams** – The DAMData object that contains all of the parameters

Example:

```
qbool tqfODBCStatementObj::dProcessOutputColumns(DAMDataPtr pOutputParams)
{
  qshort i, numParams = pOutputParams->numParams();
  if(pOutputParams->doSend() == qtrue)
  {
    for(i = 1; i <= numParams; i++)
    {
      DAMParamPtr currParam = pOutputParams->getParam(i);
      currParam->setBind(kNoBind);
      currParam->setChunk(qtrue);
    }
  }
  return qtrue;
}
```

---

tqfImpStatementObj::dGetNextChunk()

---

eChunkState tqfImpStatementObj::dGetNextChunk(DAMParamPtr pParam, qshort pParamNum)

This function should obtain a chunk of data from the database if chunking is required. The return value can be kChunkOk, kChunk-Failed or kChunkFinished.

- **pParam** – The current parameter
- **pParamNum** – The parameter number

Example:

```
eChunkState tqfODBCStatementObj::dGetNextChunk(DAMParamPtr pParam, qshort pParamNum)
{
  eChunkState chunkState = kChunkOk;
  RETCODE status;
  SLONG dataLen = 0;
  qlong *paramLen = pParam->getParamLenBuffer();
  qlong bufferLen = pParam->getBufferLen();
  void *paramBuffer = pParam->getParamBuffer();
  status = SQLGetData(mStatementHandle, pParamNum, pParam->getServerType(), paramBuffer, bufferLen, &dataLen);
```

```
  if(status == SQL_SUCCESS || status == SQL_SUCCESS_WITH_INFO)
  {
    if(dataLen >= bufferLen || dataLen == SQL_NO_TOTAL)
    {
      if(pParam->getBufferType() == fftCharacter)
        *paramLen = bufferLen-1; // remove null
      else
        *paramLen = bufferLen;
    }
    else
      *paramLen = dataLen;
  }
  if(status == SQL_SUCCESS)
    chunkState = kChunkFinished;
  else if(status != SQL_SUCCESS_WITH_INFO)
    chunkState = kChunkFailed;
  return chunkState;
}
```

tqfImpStatementObj::dClearNativeError()
void tqfImpStatementObj::dClearNativeError()

This function should clear the errors from the statement object.

Example:

```
void tqfODBCStatementObj::dClearNativeError()
{
  mErrorInfo.clearError();
}
```

tqfImpStatementObj::dGetNativeError()
eErrorState tqfImpStatementObj::dGetNativeError(qlong &pErrorCode, EXTfldval &pErrorText)

This function gets the error code and error text from the statement object and returns an error state that can be kErrorFailed, kErrorOk or kErrorPending.

- **pErrorCode** – A long to return the error code into.
- **pErrorText** – A EXTfldval that the error text can be returned into.

Example:

```
eErrorState tqfODBCStatementObj::dGetNativeError(qlong &errorCode, EXTfldval &pErrorText)
{
  eErrorState errorState = kErrorOk;
  if(mErrorInfo.getErrorPending() == qtrue)
  {
    errorCode = mErrorInfo.getErrorCode();
    mErrorInfo.getErrorText(pErrorText);
    errorState = mErrorInfo.setError();
  }
  else
  {
    errorState = mErrorInfo.setError();
    errorCode = mErrorInfo.getErrorCode();
```

```
    mErrorInfo.getErrorText(pErrorText);
    if(errorState == kErrorPending)
      errorState = mErrorInfo.setError();
  }
  return errorState;
}
```

---

tqfImpStatementObj::dMethodCall()

---

qbool tqfImpStatementObj::dMethodCall(EXTCompInfo* pEci, EXTfldval &rtnVal, qbool &hasRtnVal)

This function is an extension of the methodCall() function in the statement base class and allows the DAM implementation to handle custom methods.

- **pEci** – The External Component Information object containing information about the custom method.
- **rtnVal** – The value returned from the custom method
- **hasRtnVal** – qtrue if the custom method has returned a value.

Example:

```
qbool tqfODBCStatementObj::dMethodCall(EXTCompInfo* pEci, EXTfldval &rtnVal, qbool &hasRtnVal)
{
  qbool rtnStatus = qfalse;
  switch(ECOgetId(pEci))
  {
    case cODBCMethodId_PrepareForUpdate:
    {
      rtnStatus = prepareForUpdate(pEci);
      rtnVal.setLong(rtnStatus);
      rtnStatus = hasRtnVal = qtrue;
      break;
    }
    default:
    {
    }
  }
  return rtnStatus;
}
```

---

tqfImpStatementObj::dPropertyCanAssign()

---

qbool tqfImpStatementObj::dPropertyCanAssign(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

This function is an extension of the ECM_PROPERTYCANASSIGN message in the statement base class propertySupport function and allows the DAM implementation to handle custom properties. dPropertyCanAssign() should return qtrue if a custom property is assignable, qfalse if the property is read-only.

- **wParam** - Not Used
- **lParam** - Not Used
- **eci** - The External Component Information object containing information about the property.

Example:

```
qbool tqfODBCStatementObj::dPropertyCanAssign(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)
{
  qbool rtnStatus = qfalse;
  switch(ECOgetId(eci))
  {
    case cODBCPropertyNativeStaWarnCode:
    {
      rtnStatus = qfalse;
      break;
    }
    case cODBCPropertyNativeStaWarnText:
    {
      rtnStatus = qfalse;
      break;
    }
    case cODBCPropertyArraySize:
    {
      rtnStatus = qtrue;
      break;
    }
    default:
    {
    }
  }
  return rtnStatus;
}
```

---

tqfImpStatementObj::dSetProperty()

---

qbool tqfImpStatementObj::dSetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

This function is an extension of the ECM_SETPROPERTY message in the statement base class propertySupport function and allows the DAM implementation to handle custom properties. dSetProperty() is responsible for setting internal property values and returns qtrue is the property was successfully handled.

- **wParam** - wParam is set to ECM_WPARAM_PROPBUTTON if the Property Manager popup button was pressed to set the property. For example, a file name property may wish to use a file open dialog if the popup button was pressed.
- **lParam** - Not Used
- **eci** - The External Component Information object containing information about the property.

Example:

```
qbool tqfODBCStatementObj::dSetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)
{
  qbool rtnStatus = qtrue;
  EXTParamInfo* param = ECOfindParamNum( eci, 1 );
  if ( param )
  {
    EXTfldval fval( (qfldval)param->mData );
    switch(ECOgetId(eci))
    {
      case cODBCPropertyArraySize:
      {
        mArraySize = fval.getLong();
        break;
      }
      default:
```

```
      {
        rtnStatus = qfalse;
      }
    }
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dGetProperty()

qbool tqfImpStatementObj::dGetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

This function is an extension of the ECM_GETPROPERTY message in the statement base class propertySupport function and allows the DAM implementation to handle custom properties. dGetProperty() is responsible for returning custom property values to Omnis and returns qtrue if the requested property id was successfully handled.

- **wParam** - Not Used
- **lParam** - Not Used
- **eci** - The External Component Information object containing information about the property.

Example:

```
qbool tqfODBCStatementObj::dGetProperty(WPARAM wParam, LPARAM lParam, EXTCompInfo* eci, EXTfldval &pPropVal)
{
  qbool rtnStatus = qtrue;
  switch(ECOgetId(eci))
  {
    case cODBCPropertyArraySize:
    {
      pPropVal.setLong(mArraySize);
      break;
    }
    case cODBCPropertyNativeStaWarnCode:
    {
      pPropVal.setLong(mWarnCode);
      break;
    }
    case cODBCPropertyNativeStaWarnText:
    {
      pPropVal.setChar(mWarnText.getChar());
      break;
    }
    default:
    {
      rtnStatus = qfalse;
    }
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dProcessInlines()

qbool tqfImpStatementObj::dProcessInlines(DAMDataPtr pDAMData)

This function allows the DAM implementation to process inline data before the statement is executed.

- **pDAMData** – The DAMData object containing the parameters

Example:

```
qbool tqfODBCStatementObj::dProcessInlines(DAMDataPtr pInputParams)
{
  qbool rtnStatus = qtrue;
  qshort i;
  DAMParamPtr currParam = NULL;
  for(i = 1; i <= pInputParams->numParams() && rtnStatus == qtrue; i++)
  {
    currParam = pInputParams->getParam(i);
    if (currParam->getSendType() == kDeferInline)
    {
      //perform in-line processing for param here.
    }
  }
  return rtnStatus;
}
```

tqfImpStatementObj::dRpcProcedures()
qbool tqfImpStatementObj::dRpcProcedures(str255 &pOwner)

This function should issue a call to generate a result set containing a list of remote procedures available in the database, optionally belonging to the specified owner.

- **pOwner** - If non-empty, this is can be either *owner* or *database.owner*

Example:

```
qbool tqfODBCStatementObj::dRpcProcedures(str255 &pOwner)
{
  // Owner is either just the owner (can be empty), or database.owner
  str255 database;
  if (pOwner.length())
  {
    qshort dotPos = pOwner.pos('.');
    if (dotPos)
    {
      database = pOwner;
      database[0] = dotPos - 1;
      pOwner.deleet(1, dotPos);
    }
  }
  qshort ownerLen = pOwner.length() ? SQL_NTS : 0;
  RETCODE status = SQLProcedures(mStatementHandle, database, database.length(), owner, ownerLen, 0, 0);
  if (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    return qfalse;
  return qtrue;
}
```

tqfImpStatementObj::dRpcParameters()
qbool tqfImpStatementObj::dRpcParameters(str255 &pProcedureName)

This function should issue a call to generate a result set containing a list of parameters for the specified remote procedure.

- **pProcedureName** – The name of the procedure

Example:

```
qbool tqfODBCStatementObj::dRpcParameters(str255 &pProcedureName)
{
  // pProcedureName can be in the form procedureName, owner.ProcedureName, or database.owner.ProcedureName
  str255 ownerStr, databaseStr;
  splitTableName(databaseStr, ownerStr, pProcedureName);
  // Generate the result set
  qshort databaseLen = databaseStr.length() ? SQL_NTS : 0;
  qshort ownerLen = ownerStr.length() ? SQL_NTS : 0;
  qshort procedureNameLen = pProcedureName.length() ? SQL_NTS : 0;
  RETCODE status = SQLProcedureColumns(mStatementHandle, databaseStr, databaseLen, ownerStr, ownerLen, pProced
  if (status != SQL_SUCCESS && status != SQL_SUCCESS_WITH_INFO)
    return qfalse;
  return qtrue;
}
```

tqfImpStatementObj::dRpcProceduresGetDatabaseOrCatalog()

void tqfImpStatementObj::dRpcProceduresGetDatabaseOrCatalog(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcprocedures(). This function should extract the database or catalog of the procedure from the results set.

- **pColVal** – The destination list field value for the database or catalog of the procedure.

Example:

```
void tqfODBCStatementObj::dRpcProceduresGetDatabaseOrCatalog(EXTfldval &pColVal)
{
  // DODBC_PROCEDURES_CAT defines the result set column number that contains the catalog information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PROCEDURES_CAT);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcProceduresGetOwner()

void tqfImpStatementObj::dRpcProceduresGetOwner(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcprocedures(). This function should extract the owner of the procedure from the results set.

- **pColVal** – The destination list field value for the owner of the procedure.

Example:

```
void tqfODBCStatementObj::dRpcProceduresGetOwner(EXTfldval &pColVal)
{
  // DODBC_PROCEDURES_SCHEMA defines the result set column number that contains the schema information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PROCEDURES_SCHEMA);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcProceduresGetProcedureName()

void tqfImpStatementObj::dRpcProceduresGetProcedureName(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcprocedures(). This function extracts the name of the procedure from the results set.

- **pColVal** – The destination list field value for the name of the procedure.

Example:

```
void tqfODBCStatementObj::dRpcProceduresGetProcedureName(EXTfldval &pColVal)
{
  // DODBC_PROCEDURES_NAME defines the result set column number that contains the procedure name.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PROCEDURES_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcProceduresGetDamInfo()

void tqfImpStatementObj::dRpcProceduresGetDamInfo(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcprocedures(). This function extracts any DAM specific information about the procedure from the results set after a $rpcprocedures call. This is not a pure virtual function and so it only needs to be overridden in the DAM if desired.

- **pColVal** – The destination field value for any DAM specific information about the procedure. Can be assigned from a row variable to return more than one attribute.

Example:

```
void tqfODBCStatementObj::dRpcProceduresGetDamInfo(EXTfldval &pColVal)
{
  // DODBC_PROCEDURES_OTHER defines the result set column number that contains other information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PROCEDURES_OTHER);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetDatabaseOrCatalog()

void tqfImpStatementObj::dRpcParametersGetDatabaseOrCatalog(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the database or catalog of the RPC parameter from the results set.

- **pColVal** – The destination field value for the database or catalog.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetDatabaseOrCatalog(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_CAT defines the result set column number that contains the catalog information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_CAT);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetOwner()

void tqfImpStatementObj::dRpcParametersGetOwner(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the owner of the parameter from the results set.

- **pColVal** – The destination field value for the owner of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetOwner(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_SCHEMA defines the result set column number that contains the schema information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_SCHEMA);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetParameterName()

void tqfImpStatementObj::dRpcParametersGetParameterName(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the name of the parameter from the results set.

- **pColVal** – The destination EXTfldval for the name of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetParameterName(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_PARAM_NAME defines the result set column number that contains the parameter name.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_PARAM_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetOmnisDataTypeText()

void tqfImpStatementObj::dRpcParametersGetOmnisDataTypeText(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the Omnis data type text of the parameter from the results set.

- **pColVal** – The destination field value for the Omnis data type text of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetOmnisDataTypeText(EXTfldval &pColVal)
{
  // Get the data type
  // DODBC_PARAMETERS_DATA_TYPE defines the result set column number that contains the parameter data type.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_DATA_TYPE);
  param->setFldVal(pColVal, mBatchRow);
  mCurColOdbcDataType = pColVal.getLong();
  // Get the column size
```

```
  // DODBC_PARAMETERS_COLUMN_SIZE defines the result set column number that contains the parameter column size
  param = mOutputParams.getParam(DODBC_PARAMETERS_COLUMN_SIZE);
  param->setFldVal(pColVal, mBatchRow);
  qlong columnSize = pColVal.getLong();
  // Get the decimal digits
  // DODBC_PARAMETERS_DECIMAL_DIGITS defines the result set column number that contains the decimal digits.
  param = mOutputParams.getParam(DODBC_PARAMETERS_DECIMAL_DIGITS);
  param->setFldVal(pColVal, mBatchRow);
  qshort decimalDigits = pColVal.getLong();
  // Use a temporary parameter, to allow us to map from the above information to the OMNIS data type
  DAMParam tempParam;
  odbc2Param(&tempParam, mCurColOdbcDataType, columnSize, decimalDigits);
  // Remember these values, for later columns
  mCurColType = tempParam.getBufferType();
  mCurColSubType = tempParam.getBufferSubType();
  // Get the OMNIS data type string
  str255 typeText;
  DAMgetDataTypeText(mCurColType, mCurColSubType, tempParam.getFieldLen(), &typeText);
  pColVal.setChar(typeText);
}
```

tqfImpStatementObj::dRpcParametersGetSqlDataType()

void tqfImpStatementObj::dRpcParametersGetSqlDataType(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the SQL data type of the parameter from the results set.

- **pColVal** – The destination EXTfldval for the SQL data type of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetSqlDataType(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_TYPE_NAME defines the result set column number that contains the sql data type name.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_TYPE_NAME);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetLength()

void tqfImpStatementObj::dRpcParametersGetLength(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the length of the parameter from the results set.

- **pColVal** – The destination EXTfldval for the length of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetLength(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_COLUMN_SIZE defines the result set column number that contains the column size.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_COLUMN_SIZE);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetScale()

void tqfImpStatementObj::dRpcParametersGetScale(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the scale of the parameter from the results set.

- **pColVal** – The destination field value for the scale of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetScale(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_DECIMAL_DIGITS defines the result set column number that contains the parameters scale.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_DECIMAL_DIGITS);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpcParametersGetPassType()

void tqfImpStatementObj::dRpcParametersGetPassType(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the pass type of the parameter from the results set.

- **pColVal** – The destination field value for the pass type of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetPassType(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_COLUMN_PASS_TYPE defines the result set column number that contains the parameters pass
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_COLUMN_PASS_TYPE);
  param->setFldVal(pColVal, mBatchRow);
  qlong passType = pColVal.getLong();
  switch (passType)
  {
  case SQL_PARAM_INPUT:
    pColVal.setConstant(preParameterInput); break;
  case SQL_PARAM_OUTPUT:
    pColVal.setConstant(preParameterOutput); break;
  case SQL_PARAM_INPUT_OUTPUT:
    pColVal.setConstant(preParameterInputOutput); break;
  case SQL_RETURN_VALUE:
    pColVal.setConstant(preParameterReturnValue); break;
  }
}
```

tqfImpStatementObj::dRpcParametersGetOmnisDataType()

void tqfImpStatementObj::dRpcParametersGetOmnisDataType(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the Omnis data type of the parameter from the results set.

- **pColVal** – The destination field value for the Omnis data type of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetOmnisDataType(EXTfldval &pColVal)
{
  //mCurColType was set in dRpcParametersGetOmnisDataTypeText()
  pColVal.setLong(mCurColType);
}
```

tqfImpStatementObj::dRpcParametersGetOmnisDataSubType()

void tqfImpStatementObj::dRpcParametersGetOmnisDataSubType(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts the Omnis data sub type of the parameter from the results set.

- **pColVal** – The destination field value for the Omnis data sub type of the parameter.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetOmnisDataSubType(EXTfldval &pColVal)
{
  // mCurColSubType was set in dRpcParametersGetOmnisDataTypeText()
  pColVal.setLong(mCurColSubType);
}
```

tqfImpStatementObj::dRpcParametersGetDamInfo()

void tqfImpStatementObj::dRpcParametersGetDamInfo(EXTfldval &pColVal)

This function is called by the base class during a fetch of the results set generated by a previous call to $rpcparameters(). This function extracts any DAM specific information about the parameter from the results set. This is not a pure virtual function and so it only needs to be overridden in the DAM if desired.

- **pColVal** – The destination field value for any DAM specific information about the parameter. Can be assigned a row variable to return several attributes.

Example:

```
void tqfODBCStatementObj::dRpcParametersGetDamInfo(EXTfldval &pColVal)
{
  // DODBC_PARAMETERS_OTHER defines the result set column number that contains other information.
  DAMParamPtr param = mOutputParams.getParam(DODBC_PARAMETERS_OTHER);
  param->setFldVal(pColVal, mBatchRow);
}
```

tqfImpStatementObj::dRpc()

qbool tqfImpStatementObj::dRpc(DAMrpcDefn *pRpcDefn, EXTCompInfo *pEci)

The function is called to execute the RPC defined in pRpcDefn and stores any output from the procedure.

- **pRpcDefn** – The DAMrpcDefn object containing information about the RPC to be executed
- **pEci** – The EXTCompInfo object contain information for the RPC execution.

Example:

```
qbool tqfODBCStatementObj::dRpc(DAMrpcDefn *pRpcDefn, EXTCompInfo *pEci)
{
  eDAMError errorCode = kDAMNoError;
  qbool rtnStatus = qfalse;
  do {
    // Build the text for the RPC call. This requires parameter markers for the return value, and the paramete
    EXTfldval rpcCall;
    buildRpcCall(pRpcDefn, rpcCall);
    // Now, bind the RPC parameters. These are all bound using mInputParams, even the RPC output parameters, s
    rpcDefnToBindVars(pRpcDefn, pEci);
    // Set up the bind variables
    if (!setupInputVars(kBindExecute)) break;
    if (!dProcessPrepareParams(&mInputParams))
    {
      errorCode = kStatementRpcPrepareParamsError;
      break;
    }
    qret bindStatus = DAMgetBindVariableValues(pEci, mBindVars);
    if (bindStatus != e_ok)
    {
      errorCode = kStatementRpcGetParamValuesError;
      break;
    }
    if (!setBufferValues(qtrue)) break;
    if (!bindInputBuffers(kBindExecute)) break;
    // Now call the procedure and store the output parameters in the caller's parameters
    qHandle textHandle = rpcCall.getHandle(qfalse);
    qHandleTextPtr textHp(textHandle, 0);
    if (!dExecDirect(*textHp, textHp.charLen()) || !sendInputValues() || !storeRpcOutputParameters(pRpcDefn, p
    {
      errorCode = kStatementExecDirectFailed;
      if ((mSession != NULL) && (kTranAutomatic == mSession->getTranMode())) mSession->rollback();
        break;
    }
    mSQLText = rpcCall;
    // Success
    rtnStatus = qtrue;
  } while (0);
  if (!rtnStatus && errorCode != kDAMNoError) setError(errorCode);
    return rtnStatus;
}
```

---

tqfImpStatementObj::dRpcBindIndex()

---

qshort tqfImpStatementObj::dRpcBindIndex(DAMrpcDefn *pRpcDefn, qshort pParamIndex)

This function is called to map a zero-based index in the parameter definition, to a zero-based index in the bind variables array for the parameters. Allows the implementation to bind variables in a different order to that specified in the RPC definition.

- **pRpcDefn** – The DAMrpcDefn object containing information about the RPC
- **pParamIndex** – The Parameter number

Example:

```
qshort tqfODBCStatementObj::dRpcBindIndex(DAMrpcDefn *pRpcDefn, qshort pParamIndex)
```

```
{
  // Need to make sure the return value binds as the first parameter
  qshort returnValueIndex = pRpcDefn->returnValueIndex();
  if (returnValueIndex < 0) return pParamIndex; // No return value
  else if (pParamIndex == returnValueIndex) return 0;
  else if (pParamIndex < returnValueIndex) return pParamIndex + 1;
  else return pParamIndex;
}
```

StatementWorker

The StatementWorker class provides an interface object for the SQL Worker Delegate object and is designed to be further subclasses by the derived statement worker object. Once initialised and started the worker delegate normally executes on a background thread and has access to the custom DAM's session and statement objects. Please refer to the main product documentation for a high level description of Worker Objects and their use. The generic DAM sample also contains worker object support.

Protected Members

Type	Name	Description
StatementWorkerDelegate *	mDelegate	The detachable object that performs the actual *work* of the worker

Private Methods

Although you may not call private methods directly, these are referred to in the Omnis Methods section and in the damworker.he header file so are described here in overview.

dup()
StatementWorker* StatementWorker::dup(qobjinst objinst, EXTCompInfo* pEci)

Duplicates a client worker object into a new object. Creates a new StatementWorkerObject, then calls copy(this) to initialize it and returns the new object.

copy()
void StatementWorker::copy(StatementWorker* pObj)

Copies pObj objects contents into this objects contents. Specifically, the object's mObjPtr (qobjinst) is copied, then this object's copy constructor is called.

methodInit()
qbool StatementWorker::methodInit(EXTCompInfo *pEci);

Client worker object initialisation method invoked when $init() is called. Calls mDelegate→init().
If mDelegate is marked to be detached (orphaned) then the delegate is dup()licated at this point and its init() method is called.
Returns qtrue if one (or both) calls to mDelegate→init() succeeds, qfalse otherwise.

methodRun()
qbool StatementWorker::methodRun(EXTCompInfo *pEci);

methodRun() is invoked when $run() is called. Calls mDelegate→run() on the main thread followed by mDelegate→pushWorkerCallbackFromRun
Returns qtrue if run() succeeds, qfalse otherwise.

methodStart()
qbool StatementWorker::methodStart(EXTCompInfo *pEci);

methodStart() is invoked when $start() is called. Calls mDelegate→start() and returns qtrue is start() succeeds, qfalse otherwise.

methodCancel()
qbool StatementWorker::methodCancel(EXTCompInfo *pEci);

methodCancel() is invoked when $cancel() is called on the interface object. Calls mDelegate→cancel() (which sets the delegate's mCancelled flag to qtrue). methodCancel() always returns qtrue.

methodSessionRef()
qbool StatementWorker::methodSessionRef(EXTCompInfo *pEci);

methodSessionRef() is invoked when $sessionref() is called. methodSessionRef() obtains the session object being used by mDelegate (if one exists) and dervies an object reference to it (increasing its reference count by 1). The object reference is written into the supplied pEci structure for return to Omnis.

Public Methods

StatementWorker()
StatementWorker::StatementWorker(qobjinst ptr, EXTCompInfo *pEci);

Constructor. Assigns mEci to pEci, mObjPtr to ptr and sets mRefCount to 0.

~StatementWorker()
StatementWorker::~StatementWorker()

Destructor. If mDelegate is running when the interface object destructs, then its mOrphaned flag is set to qtrue. If mCancelIfRunning it qtrue, then the delegate's methodCancel() is also called. If not running then mDelegate is deleted.

addRef()
void StatementWorker::addRef()

addRef() increments this object's mRefCount.

releaseRef()
void StatementWorker::releaseRef()

releaseRef() decrements this object's mRefCount. If mRefCount reaches zero then this object is deleted.

propCount()
static qshort StatementWorker::propCount()

Returns the number of properties supported by the Statement Worker object. This needs to be implemented in the derived worker object in case additional properties are added.

Example:

```
//Declare standard properties for a SQL worker object
ECOproperty PgSqlStatementWorkerProps[] =
{
  WORKER_PROPS   //(defined in damworker.he)
};
//..

qshort StatementWorker::propCount()
{
  return sizeof(PgSqlStatementWorkerProps)/sizeof(PgSqlStatementWorkerProps[0]);
}
```

methodCount()
static qshort StatementWorker::methodCount()

Returns the number of methods supported by the Statement Worker object. This needs to be implemented in the derived worker object in case additional methods are added.

Example:

```
//Declare standard methods for a SQL worker object
ECOmethodEvent StmtWorkerFuncs[] =
{
  WORKER_FUNCS   //(defined in damworker.he)
};
//..

qshort StatementWorker::methodCount()
{
  return sizeof(StmtWorkerFuncs)/sizeof(StmtWorkerFuncs[0]);
}
```

methodCall()
qbool StatementWorker::methodCall(EXTCompInfo* pEci)

methodCall() handles custom method calls to the StatementWorker object. It calls into one of methodInit(), methodRun(), methodStart(), methodCancel() or methodSessRef(), returning that method's result code where applicable. . If the derived worker object contains additional methods, you will need to override this method.

propertySupport()
qlong StatementWorker::propertySupport(LPARAM pMessage, WPARAM wParam, LPARAM lParam, EXTCompInfo* eci)

propertySupport() handles the StatementWorker object's standard properties; $cancelifrunning, $waitforcomplete, $state, $errortext, $errorcode, $nativeerrorcode, $nativeerrortext and $threadcount. If the derived worker object contains additional properties, you will need to override this method.

StatementWorkerDelegate

StatementWorkerDelegate is derived from the component library's Worker class and is designed to be further subclassed by the derived statement worker delegate object.

Protected Members

Datatype	Name	Description
tqfDAMbaseObj *	mSessionObj	The session object that will be used by the worker delegate
tqfDAMStatementObj *	mStatementObj	The statement object that will be used by the worker delegate
qbool	mSessionObjMine	qtrue if the delegate creates its own session object
qobjinst	mPoolRef	Assigned when initialised using a sessionpool
qbool	mCancelled	qtrue if the delegate receives a cancel request
EXTCompInfo *	mEci	External Component Info structure (assigned during construct)
str255	mHostname	Stores the hostname parameter assigned during $init()
str255	mUsername	Stores the username parameter assign during $init()
str255	mPassword	Stores the password parameter assigned during $init()
str255	mDatabase	Stores the database parameter assigned during $init()

Public Methods

StatementWorkerDelegate
StatementWorkerDelegate::StatementWorkerDelegate(qobjinst objptr, EXTCompInfo* pEci)

Constructor. Initialises all member properties to qfalse/NULL. Assigns mObjPtr to objptr and mEci to pEci.

~StatementWorkerDelegate()
StatementWorkerDelegate::~StatementWorkerDelegate()

Destructor. If using the (deprecated) timer completion model, then this object is unsubscribed from the static timer object.
If mPoolRef was assigned from a sessionpool then this is released (returned to the pool).

dup()
virtual StatementWorkerDelegate* StatementWorkerDelegate::dup(qobjinst objinst, EXTCompInfo* pEci) = 0

dup() is responsible for duplicating an existing statement worker delegate object and must be implemented in the derived object.
The newly duplicated object is returned.

Example:

```
pgsqlWorkerDelegate* pgsqlWorkerDelegate::dup(qobjinst objinst, EXTCompInfo* pEci)
{
  pgsqlWorkerDelegate* theCopy = new pgsqlWorkerDelegate(objinst, pEci); //create new object
  theCopy->setWaitForComplete(mWaitForComplete);
  theCopy->setCancelIfRunning(mCancelIfRunning);
  return theCopy; //return to caller
}
```

damGetSessionObjId()
virtual qlong StatementWorkerDelegate::damGetSessionObjId() = 0

damGetSessionObjId() should be implemented in the derived statement worker delegate object and should return the custom DAM's session object ID as returned via the ECM_GETOBJECT message.

Example:

```
const int cObject_PgSqlObj = 29; // ID of PgSql DAM object
//..

virtual qlong damGetSessionObjId() { return cObject_PgSqlObj; }
```

Protected Methods

dGetNativeError()
void StatementWorkerDelegate::dGetNativeError()

dGetNativeError() attempts to assign a database error code and error message for return via $nativeerrocode and $nativeerrortext. If this object's mErrorInfo.getErrorCode() contains *kStatementPrepareFailed* then mStatementObj->dGetNativeError() is called to obtain the native errorcode and errortext. Otherwise, the base class uses pre-defined error strings read from resources, for example:

```
mErrorInfo.setErrorCode(kWorkerAlreadyRunning);
```

(Applicable constants can be found in the dmconst.he header file and must exist between *preDAMErrorsF* and *preWorkerErrorsL*)

Note that dGetNativeError() is only used for code that executes on the main thread, i.e. during $init(), $run or in $start (before the delegate worker thread starts). Errors generated after the worker delegate thread starts are returned as part of the result row supplied to the worker object's $completed() method.

Protected Virtual Methods

dAbortFetch()
void StatementWorkerDelegate::dAbortFetch()

This method sets the mCancelled protected member to qtrue. (Override is optional). The base class monitors this flag inside the loop which processes result data and breaks-out accordingly. Note that is $cancelifrunning is set to kTrue, the worker delegate mCancelled flag is automatically set before detaching if from the interface object, i.e. the detached thread will terminate as soon as possible.

dInit()
eWorkerError StatementWorkerDelegate::dInit(EXTCompInfo* pEci, EXTqlist* pParams)

dInit() extracts and validates parameters from the supplied Omnis row parameter. If "poolname" is supplied, then dInit() attempts to obtain a session reference from the pool. If "session" is supplied, then dInit() validates and attempts to use the supplied session object or object reference.
dInit() also extracts the "hostname", "username" and "password" parameters if supplied.
It then calls damWorkerInit() which creates the database-specific session and statement objects.

- **pEci** – External Component Information structure(used to create a session object)
- **pParams** – The row variable used to initiailise the worker object

dRun()
worker::eWorkerState StatementWorkerDelegate::dRun()

dRun() executes on a background thread when $start() is called, and on the main thread when $run() is called. It sets up the *Results* and *Errors* lists that are supplied when $completed() is called. If not already connected, it then calls damWorkerLogon() instructing the derived worker delegate to connect to the database. Inside the main execution loop, dRun() then extracts the bind variables for each SQL statement, prepares, executes, describes and fetches any result sets, accumulating any error messages along the way. Calls to the derived implementation are summarised as follows:

```
damWorkerLogon()
 →mStatementObj→workerPrepare()
          clearAll()
               dClearStatement()
               dClearParams(mInputParams)
```

```
             dClearParams(mOutputParams)
          mSession→getStatementBindMarker()
          mSession→charMapOut()
              dProcessPrepareParams(mInputParams)
              dPrepare()
          bindInputBuffers(kBindPrepare)
              dBindParameter()
    mStatementObj j→mInputParams.doInline()
    mStatementOb→dProcessInlines(mInputParams)
    mStatementObj→bindInputBuffers(kBindExecute)
         dBindParameter()
    mStatementObj→dExecute()
    mStatementObj→readyForFetch()
         dSetResults()
    mStatementObj→describe()
         dDescribeParam()
       dProcessOutputColumns(mOutputParams)
    mStatementObj→fetch(kFetchAll)
         dFetch()
 mStatementObj→dClearStatement()

mStatementObj→dDropStatement()
←DAMreleasePoolRef()      (if using mPoolRef)
sessionObj->dLogoff()      (if mSessionObjMine)
```

It should not be necessary to override this method but you should ensure that your custom DAM is able to execute the highlighted derived methods in a thread-safe manner.

dCancel()
void StatementWorkerDelegate::dCancel()

dCancel() sets mCancelled to qtrue- see dAbortFetch(). Override of this method is optional.

addError()
void StatementWorkerDelegate::addError(EXTqlist *pErrList, eWorkerError pErrCode, qlong pQueryNum, qlong pBindRow)

The derived worker delegate class must implement this method. addError() is called during dRun() whenever an error occurs when extracting expected parameters, logging-on, preparing or executing SQL statements. addError() should get the native error code & native error text from the delegate's statement/session object as appropriate.

- **pErrList** – (output) Error information is added to the supplied pErrList
- **pErrCode** – An error code summarising the error, e.g. kWorkerPrepareFailed
- **pQueryNum** – For multiple queries, the query number that generated the error. 1 otherwise
- **pBindRow** – For queries with multiple rows of bind variables, the row number in the list. 1 otherwise

Example:

```
void sqliteWorkerDelegate::addError(EXTqlist *pErrList, eWorkerError pErrCode, qlong pQueryNum, qlong pBindRow
{
  EXTfldval errCode, errText, nativeErrCode, nativeErrText;
  mErrorInfo.setErrorCode(pErrCode); //set errorcode so that errText can be extracted
  mErrorInfo.getErrorText(errText);
  errCode.setConstant(preDAMErrorsF,preWorkerErrorsL,pErrCode);
  tqfSQLiteStatementObj *statement = (tqfSQLiteStatementObj*)mStatementObj;
```

```
  tqfSQLiteDAMObj *session = (tqfSQLiteDAMObj*)mSessionObj;
  qlong nec=0;
  if (statement)   //test for a statement-level error
    statement->dGetNativeError(nec, nativeErrText);
  if (nec==0 && session)   //test for a session-level error
    session->dGetNativeError(nec, nativeErrText);
  nativeErrCode.setLong( nec );
  qlong errRow = pErrList->insertRow(); //Add error info row
  pErrList->putColVal(errRow,1,errCode); //generic error code
  pErrList->putColVal(errRow,2,errText); //generic error text
  pErrList->putColVal(errRow,3,nativeErrCode); //native error code
  pErrList->putColVal(errRow,4,nativeErrText); //native error text
  EXTfldval intVal;
  intVal.setLong(pQueryNum); pErrList->putColVal(errRow,5,intVal);
  intVal.setLong(pBindRow); pErrList->putColVal(errRow,6,intVal);
}
```

damWorkerInit()
void StatementWorkerDelegate::damWorkerInit()

The derived worker delegate class must implement this method. damWorkerInit() creates the database-specific sesssion & statement objects.

Additional custom paramters can also be extracted if desried using the built-in *extractParameter()* method, e.g.

```
EXTfldval fval; str255 colName(QTEXT("my_param");
if (!extractParameter(colName, fval)) break;   //error, parameter not found
str255 myValue = fval.getChar();
```

This is possible because the worker delegate takes a copy of the initialization parameters during $init().

Example:

```
void sqliteWorkerDelegate::damWorkerInit()
{
  if (!mSessionObj)  //we need to create a session object if not already supplied
  {
    mSessionObj = (tqfDAMbaseObj*) new tqfSQLiteDAMObj(mEci);
    mSessionObjMine = qtrue;   //worker object will delete mSessionObj
  }
  if (!mStatementObj) mStatementObj = new tqfSQLiteStatementObj();
  ((tqfSQLiteDAMObj*)mSessionObj)->dSetBindMarker();
  ((tqfDAMStatementObj*)mStatementObj)->setSession(mSessionObj);
}
```

damWorkerLogon()
qbool StatementWorkerDelegate::damWorkerLogon()

The derived worker delegate class must implement this method. damWorkerLogon() is responsible for performing any additional configuration and logging on to the derived session object. This can usually be achieved by simply calling the derived session object's dLogon() method.

Example:

```
qbool sqliteWorkerDelegate::damWorkerLogon()
{
  tqfSQLiteDAMObj *session = (tqfSQLiteDAMObj*)mSessionObj;
  return session->dLogon(mHostname, mUsername, mPassword);
}
```

tqfDAMObjCont

This is a standard container class used by Omnis external components to encapsulate a non-visual object. tqfDAMObjCont is tailored for a tqfDambaseObj. Container classes are used inside the component's entry procedure or "ObjProc" when processing the ECM_OBJCONSTRUCT,

ECM_OBJDESTRUCT, ECM_PROPERTYCANASSIGN, ECM_SETPROPERTY, ECM_GETPROPERTY and ECM_METHODCALL messages.

Public Members

Type	Name	Description
tqfDAMbaseObjPtr	mObject	The DAM base object.
qobjinst	mObjPtr	The DAM object instance ptr

Public Methods

tqfDAMObjCont::tqfDAMObjCont()

tqfDAMObjCont::tqfDAMObjCont(qobjinst pObjPtr, tqfDAMbaseObjPtr pNewObject)

This constructor initialises the object with the parameters shown and is normally used in response to the ECM_OBJCONSTRUCT message inside the DAM's message handler.

- **pObjPtr** – The session object instance pointer
- **pNewObject** – The session object.

Example:

```
extern "C" qlong OMNISWNDPROC ODBCObjProc(OMNISHWND hwnd, LPARAM Msg,WPARAM wParam,LPARAM lParam,EXTCompInfo* 
{
  // Initialize callback tables - THIS MUST BE DONE
  ECOsetupCallbacks(hwnd, eci);
  switch (Msg)
  {
    //…
    case ECM_OBJCONSTRUCT:
    {
      if ( eci->mCompId==cObject_ODBCObj )
      {
        tqfDAMObjCont* object = (tqfDAMObjCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
        if ( !object )
        {
          tqfODBCDAMObj* damObj = new tqfODBCDAMObj(eci);
          tqfDAMObjCont* obj = new tqfDAMObjCont((qobjinst)lParam, damObj);
          ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
          return qtrue;
        }
      }
      else if ( eci->mCompId==cObject_ODBCStat )
      {
        tqfDAMStatementCont* object = (tqfDAMStatementCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
        if ( !object )
        {
          tqfDAMStatementCont* obj = new tqfDAMStatementCont((qobjinst)lParam);
          ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
          return qtrue;
        }
```

```
      }
      else return qfalse;
      return qtrue;
    }
    //…
  }
}
```

tqfDAMObjCont::tqfDAMObjCont()

tqfDAMObjCont::tqfDAMObjCont(qobjinst pObjPtr,tqfDAMObjCont* pObjectContainer)

This constructor initialises the object from an existing DAM object container.

- **pObjPtr** - The session object instance pointer
- **pObjectContainer** – A tqfDAMObjCont object

tqfDAMObjCont::~tqfDAMObjCont()

tqfDAMObjCont::~tqfDAMObjCont()

This destructor decrements the reference count of the DAM object. When the number of containers referencing the DAM object reaches zero, the base class deletes the DAM object.

tqfDAMObjCont::setObject()

void tqfDAMObjCont::setObject(qobjinst pObjPtr,tqfDAMObjCont* pSource)

This function removes any previous objects contained within and replaces it with pSource.

- **pObjPtr** – The session object instance pointer
- **pSource** – The session object.

tqfDAMStatementCont

This is a standard container class used by Omnis external components to encapsulate a non-visual object. tqfDAMStatementCont is tailored for a tqfDAMStatementObj.

Public Members

Type	Name	Description
tqfDAMStatementObjPtr	mObject	The DAM statement object.
qobjinst	mObjPtr	The DAM statement object instance ptr

Public Methods

tqfDAMStatementCont::tqfDAMStatementCont()

tqfDAMStatementCont:: tqfDAMStatementCont(qobjinst pObjPtr, tqfDAMbaseObjPtr pNewObject)

This constructor initialises the object with the parameters shown and is normally used in response to the ECM_OBJCONSTRUCT message inside the DAM's message handler.

- **pObjPtr** – The statement object instance pointer
- **pNewObject** – The statement object.

Example: See the tqfDAMObjCont example

tqfDAMStatementCont::tqfDAMStatementCont()
tqfDAMStatementCont::tqfDAMStatementCont(qobjinst pObjPtr,tqfDAMObjCont* pObjectContainer)

This constructor initialises the object from an existing statement object container.

- **pObjPtr** - The statement object instance pointer
- **pObjectContainer** - A tqfDAMStatementCont object

tqfDAMStatementCont::~tqfDAMStatementCont()
tqfDAMStatementCont::~tqfDAMStatementCont()

This destructor decrements the reference count of the statement object. When the reference count reaches zero the base class deletes the statement object.

tqfDAMObjCont::setObject()
void tqfDAMObjCont::setObject(qobjinst pObjPtr,tqfDAMObjCont* pSource)

This function removes any previous objects contained within and replaces it with pSource.

- **pObjPtr** – The statement object instance pointer
- **pSource** – The statement object.

StatementWorkerCont

This is a standard container class used by Omnis external components to encapsulate a non-visual object. StatementWorkerCont is tailored for a StatementWorker interface object.

Public Members

Type	Name	Description
StatementWorkerPtr	mObject	The StatementWorker object
qobjinst	mObjPtr	The Omnis object instance ptr

Public Methods

StatementWorkerCont::StatementWorkerCont()
StatementWorkerCont::StatementWorkerCont(qobjinst pObjPtr, StatementWorkerPtr pNewObject)

This constructor initialises the object with the parameters shown and is normally used in response to the ECM_OBJCONSTRUCT message inside the DAM's message handler.

- **pObjPtr** – The statement object instance pointer

- **pNewObject** – The StatementWorker object.

Example: (except from the SQLite DAM's SQLiteObjProc)

```
case ECM_OBJCONSTRUCT: //This is a message informing you to create a new object
{
  if ( eci->mCompId==cObject_SQLiteSess )
  {
    tqfDAMObjCont* object = (tqfDAMObjCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
    if ( !object )
    {
      tqfSQLiteDAMObj* damObj = new tqfSQLiteDAMObj(eci);
      tqfDAMObjCont* obj = new tqfDAMObjCont((qobjinst)lParam, damObj);
      ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
      return qtrue;
    }
  }
  else if ( eci->mCompId==cObject_SQLiteStat )
  {
    tqfDAMStatementCont* object = (tqfDAMStatementCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
    if ( !object )
    {
      tqfDAMStatementCont* obj = new tqfDAMStatementCont((qobjinst)lParam);
      ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
      return qtrue;
    }
  }
  else if ( eci->mCompId==cObject_StatementWorker )
  {
    StatementWorkerCont* object = (StatementWorkerCont*)ECOfindNVObject( eci->mOmnisInstance, lParam );
    if ( !object )
    {
      sqliteWorker* worker = new sqliteWorker((qobjinst)lParam, eci);
      StatementWorkerCont* obj = new StatementWorkerCont((qobjinst)lParam, worker);
      ECOinsertNVObject( eci->mOmnisInstance, lParam, (void*)obj );
      return qtrue;
    }
  }
  return qfalse;
}
```

StatementWorkerCont::StatementWorkerCont()

StatementWorkerCont::StatementWorkerCont(qobjinst pObjPtr, StatementWorkerCont* pObjectContainer)

This constructor initialises the object from an existing statement object container.

- **pObjPtr** - The statement object instance pointer
- **pObjectContainer** - A tqfDAMStatementCont object

StatementWorkerCont::~StatementWorkerCont()

StatementWorkerCont::~StatementWorkerCont ()

This destructor decrements the reference count of the statement object. When the reference count reaches zero the base class deletes the statement object.

StatementWorkerCont::setObject()

void StatementWorkerCont::setObject(qobjinst pObjPtr, StatementWorkerCont * pSource)

This function removes any previous object contained within and replaces it with pSource.

- **pObjPtr** – The StatementWorker object instance pointer
- **pSource** – The StatementWorker object container.

Support Classes

DAMTypeTable

This class contains the mapping between Omnis data types and C Types used for transferring data. The Type table is created during the construction of the DAMTypeTable class.

Constants and enums referred to in this section are defined as follows:

fft types and subtypes are defined in EXTFVAL.HE.

enum eTypeEntry is defined in DAMOBJ.HE.

enum eSendType is defined in DAMTYPES.HE.

Public Methods

DAMTypeTable::DAMTypeTable()

DAMTypeTable::DAMTypeTable()

This constructor initialises the type table independent of the database. An entry exists in the type table for each Omnis type and associates it with a C type. e.g. kTypeCharacter maps to omChar.

DAMTypeTable::~DAMTypeTable()

DAMTypeTable::~DAMTypeTable()

Default destructor. Automatically frees the type table resources.

DAMTypeTable::setCType()

void DAMTypeTable::setCType(ffttype pSourceType, qshort pSourceSubType, eCType pCType, qshort pScale = dpDefault)

This function locates the entry in the type table specified by pSourceType and pSourceSubtype and updates the entry's C-type and scale.

- **pSourceType** – The ffttype for the entry to be updated, e.g. fftCharacter
- **pSourceSubType** – The subtype for the entry to be updated, e.g. dpFcharacter
- **pCType** – The new C type.
- **pScale** – The new scale (optional).

Example:

```
setCType(fftDate, dpFdate1980, omChar);
```

DAMTypeTable::setCType()

void DAMTypeTable::setCType(eTypeEntry pTypeEntry, eCType pCType, qshort pScale = dpDefault)

This function updates the Type entry's C type and scale.

- **pTypeEntry**– The entry to be updated. See Type Entry Constants.
- **pCType** – The new C type.
- **pScale** – The new scale (optional).

Example:

```
setCType(kTypeDateTime,omChar);
setCType(kTypeShortInteger,omSLong);
```

DAMTypeTable::setSendType()

void DAMTypeTable::setSendType(ffttype pSourceType, qshort pSourceSubType, eSendType pSendType)

This function locates the entry in the type table specified by pSourceType and pSourceSubtype and updates the entry's send type, one of: kConvert, kInline, kDefault or kDeferInline.

- **pSourceType** – The ffttype for the entry to be updated
- **pSourceSubType** – The subtype for the entry to be updated
- **pSendType** – The new send type.

Example:

```
setSendType(fftInteger, dpFsinteger, kConvert);
```

DAMTypeTable::setSendType()

void DAMTypeTable::setSendType(eTypeEntry pTypeEntry, eSendType pSendType)

This function locates the entry in the type table specified by pTypeEntry and updates the entry's send type, one of: kConvert, kInline, kDefault or kDeferInline.

- **pTypeEntry** - The entry to be updated. See Type Entry Constants.
- **pSendType** - The new send type

Example:

```
setSendType(kTypeCharacter, kDeferInline);
setSendType(kTypeNumber, kConvert);
```

DAMTypeTable::getSendType(eTypeEntry)

eSendType DAMTypeTable::getSendType(eTypeEntry pTypeEntry)

Returns the send type of the specified type entry.

- **pTypeEntry** - One of the *eTypeEntry* constants.

DAMTypeTable::getSendType()

eSendType DAMTypeTable::getSendType(ffttype pSourceType, qshort pSourceSubType)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and returns the entry's send type.

- **pSourceType** – The ffttype for the entry.
- **pSourceSubType** – The subtype for the entry

DAMTypeTable::getCType()

eCType DAMTypeTable::getCType(eTypeEntry pTypeEntry)

Returns the C type of the type entry. *enum eCType* is defined in DAMTYPES.HE

- **pTypeEntry** - One of the *eTypeEntry* constants defined in DAMOBJ.HE

DAMTypeTable::getCType()

eCType DAMTypeTable::getCType(ffttype pSourceType, qshort pSourceSubType)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and returns the entry's C type.

- **pSourceType** – The ffttype for the entry.
- **pSourceSubType** – The subtype for the entry

DAMTypeTable::setTypeBufferLength()

void DAMTypeTable::setTypeBufferLength(ffttype pSourceType, qshort pSourceSubType, qlong pLength)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and updates the entry's buffer length attribute.

- **pSourceType** – The ffttype for the entry to be updated
- **pSourceSubType** – The subtype for the entry to be updated
- **pLength** – The new length.

DAMTypeTable::setTypeBufferLength()

void DAMTypeTable::setTypeBufferLength(eTypeEntry pTypeEntry, qlong pLength)

This function updates the type entry's buffer length attribute.

- **pTypeEntry** - One of the *eTypeEntry* constants.
- **pLength** - The new length.

DAMTypeTable::getTypeBufferLength()

qlong DAMTypeTable::getTypeBufferLength(ffttype pSourceType, qshort pSourceSubType)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and returns the entry's buffer length.

- **pSourceType** – The ffttype for the entry
- **pSourceSubType** – The subtype for the entry

DAMTypeTable::getTypeBufferLength()

qlong DAMTypeTable::getTypeBufferLength(eTypeEntry pTypeEntry)

Returns the buffer length of the type entry.

- **pTypeEntry** - One of the *eTypeEntry* constants.

DAMTypeTable::getScale()

qlong DAMTypeTable::getScale(ffttype pSourceType, qshort pSourceSubType)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and returns the entry's scale.

- **pSourceType** – The ffttype for the entry
- **pSourceSubType** – The subtype for the entry

DAMTypeTable::getScale()

qlong DAMTypeTable::getScale(eTypeEntry pTypeEntry)

Returns the scale attribute for the specified type entry.

- **pTypeEntry** - One of the *eTypeEntry* constants.

DAMTypeTable::setFormat()

void DAMTypeTable::setFormat(ffttype pSourceType, qshort pSourceSubType, str255 &pFormat)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and updates the entry's date format string. The format string specifies how a dates and times sent to the database should be formatted. See Omnis *Date codes* constants for a list of format characters.

- **pSourceType** – The ffttype for the entry to be updated
- **pSourceSubType** – The subtype for the entry to be updated
- **pFormat** – The new format string.

DAMTypeTable::setFormat()
void DAMTypeTable::setFormat(eTypeEntry pTypeEntry, str255 &pFormat)

This function updates the Type entry's date format string.

- **pTypeEntry** - The entry to be updated. One of the *eTypeEntry* constants.
- **pFormat** - The new format string

Example:

```
RESloadString(gInstLib, cDefaultDateFormat, formatString); //e.g. "y-M-D H:N:S"
mTypeTable.setFormat(kTypeDateTime, formatString);
```

DAMTypeTable::getFormat()
void DAMTypeTable::getFormat(ffttype pSourceType, qshort pSourceSubType, str255 &pFormat)

This function locates the entry in the type table based on the pSourceType and pSourceSubtype and returns the entry's date format string.

- **pSourceType** – The ffttype for the entry to be retrieved
- **pSourceSubType** – The subtype for the entry to be retrieved
- **pFormat** – The entry's format string (output).

DAMTypeTable::getFormat()
void DAMTypeTable::getFormat(eTypeEntry pTypeEntry, str255 &pFormat)

Returns the date format for the specified type entry.

- **pTypeEntry** - The entry to be retrieved. See Type Entry Constants.
- **pFormat** - The entry's date format string (output).

Example:

```
str255 formatString;
mTypeTable.getFormat(kTypeDateTime, formatString);
```

DAMErrorInfo

This class maintains the native error code, error message and any pending errors for its parent object. Both the session and statement base classes contain a DAMErrorInfo object (mErrorInfo).

DAMErrorInfo was originally private to both the derived session and statement objects. It was later made a protected member of the statement base class to enable advanced features to be implemented, for example allowing errors to be placed directly into the object without waiting for dGetNativeError() to be called.

Thus, derived objects should not normally require access directly to mErrorInfo. Instead, they use their dGetNativeError() and setError() methods as described below.

Public Methods

DAMErrorInfo::DAMErrorInfo()
DAMErrorInfo::DAMErrorInfo()

This constructor initialises object, clearing the private members.

DAMErrorInfo::~DAMErrorInfo()
DAMErrorInfo::~DAMErrorInfo()

Default destructor

DAMErrorInfo::clearError()
void DAMErrorInfo::clearError()

This function clears any stored errors. (Handled automatically by the base class)

DAMErrorInfo::setErrorCode()
void DAMErrorInfo::setErrorCode(qlong pError)

This function sets the internal error code. eDAMError constants are defined in DAMOBJ.HE.

The derived object, should use the setError() method instead.

- **pError** - The internal error code

Example:

```
setError(kDAMParameterError);
```

DAMErrorInfo::getErrorCode()
qlong DAMErrorInfo::getErrorCode()

This function returns the internal error code. Exposed to Omnis via the $errorcode property, this method is not available to the derived session object.

DAMErrorInfo::setNativeErrorCode()
void DAMErrorInfo::setNativeErrorCode(qlong pError)

This function sets the database server error code.

Derived objects should use dGetNativeError() to set the native error code.

- **pError** - The server-specific error code

DAMErrorInfo::getNativeErrorCode()
qlong DAMErrorInfo::getNativeErrorCode()

This function returns the server error code. Exposed to Omnis via the $nativeerrorcode property, this method is not available to the derived session object.

DAMErrorInfo::setNativeErrorPending()
void DAMErrorInfo::setNativeErrorPending(qbool pErrorPending)

This function sets the error pending flag. (Handled automatically by the base class)

- **pErrorPending** – qtrue if error is pending

DAMErrorInfo::getNativeErrorPending()
qbool DAMErrorInfo::getNativeErrorPending()

This function returns qtrue if an error is pending. Exposed to Omnis via the $nativeerrorpending property, this method is not available to the derived session object.

DAMErrorInfo::getNativeErrorText()
void DAMErrorInfo::getNativeErrorText(EXTfldval &pErrorText)

This function returns the server error text. Exposed to Omnis via the $nativeerrortext property, this method is not available to the derived session object.

- **pErrorText** - Field value containing the error text (output).

DAMErrorInfo::setNativeErrorText()
void DAMErrorInfo::setNativeErrorText(EXTfldval &pErrorText)

This function sets the server error text.

Derived objects should use dGetNativeError() to set the native error text.

- **pErrorText** - Field value containing the error text

DAMParam

This class encapsulates information about Omnis bind variables, return values and result set columns.

It stores the name, Omnis and SQL data types, precision & scale, when the parameter should be bound/sent as well as providing a buffer for the actual data and facilitates chunking of binary data. There are also various methods to access and convert the data.

Public Methods

DAMParam::DAMParam()
DAMParam::DAMParam(tqfDAMStatementObj *pStatement)

This constructor initialises the parameter structure

- **pStatement** – The statement object that owns the parameter

DAMParam::~DAMParam()
DAMParam::~DAMParam()

This destructor clears all values and frees any resources owned by the parameter.

DAMParam::reset()
void DAMParam::reset()

This function resets all the parameter settings and frees resources.

DAMParam::clearBuffers()
void DAMParam::clearBuffers()

This function clears and frees the buffers used to contain the parameter data and the data length.

The implementation uses setBuffers() to reallocate these buffers.

DAMParam::setSendType()
void DAMParam::setSendType(eSendType pSendType)

The implementation calls this function at describe time or prepare time to set the send type of the parameter. This governs when and how the parameter will be processed.

- **pSendType** – The send type; one of kConvert, kInline, kDefault or kDeferInline.

DAMParam::getSendType()
eSendType DAMParam::getSendType()

This function returns the send type previously assigned to the parameter.

DAMParam::setName()
void DAMParam::setName(str255 pName)

The implementation calls this function at describe time to set the name of the parameter.

This name will appear in the column heading of the fetch list.

- **pName** – The name of the parameter

DAMParam::getName()
str255* DAMParam::getName()

This function returns (a pointer to) the name previously assigned to the parameter.

The implementation should not free the returned pointer.

DAMParam::setNullable()
void DAMParam::setNullable(qbool pNullStatus)

The implementation calls this function (e.g. at describe time) to indicate whether the parameter accepts NULL values.

- **pNullStatus** – qtrue if parameter allows NULL values

DAMParam::getNullable()
qbool DAMParam::getNullable()

This function returns qtrue if parameter was previously set to allow NULL values.

DAMParam::setBufferLen()
void DAMParam::setBufferLen(qlong pBufferLen)

This function sets the size that will be allocated for the parameter's data buffer.

- **pBufferLen** – The length of the buffer (in bytes)

DAMParam::getBufferLen()
qlong DAMParam::getBufferLen()

This function returns the size of the data buffer. Note that the data buffer is not allocated until setBuffers() is called, either by the base class or by the implementation.

DAMParam::setBufferType()
void DAMParam::setBufferType(ffttype pBufferType)

The implementation calls this function at describe time to set the fft type of the data in the parameter buffer.

- **pBufferType** – One of the ffttype constants, defined in EXTFVAL.HE

DAMParam::getBufferType()
ffttype DAMParam::getBufferType()

This function returns the buffer fft type previously assigned to the parameter.

DAMParam::setBufferSubType()
void DAMParam::setBufferSubType(qshort pBufferSubType)

The implementation calls this function at describe time to set the fft sub-type of the data in the parameter buffer.

- **pBufferSubType** – The buffer sub type, also defined in EXTFVAL.HE

DAMParam::getBufferSubType()
qshort DAMParam::getBufferSubType()

This function returns the buffer sub-type previously assigned to the parameter.

DAMParam::setCType()
void DAMParam::setCType(eCType pCType)

The implementation calls this function to set the C type for the parameter.

The C type describes the data independently of the database server type and the fft type & sub-type.

- **pCType** – One of the *enum eCType* constants defined in DAMTYPES.HE

DAMParam::getCType()
eCType DAMParam::getCType()

This function returns the C type previously assigned to the parameter.

DAMParam::setChunk()
void DAMParam::setChunk(qbool pChunk)

This function is called to indicate whether the data for the parameter is to be fetched in chunks. Chunking of parameter data is disabled by default but may be enabled by the base class if it is determined that the pending data will not fit into the allocated data buffer.

- **pChunk** – Set to qtrue to enable chunking

DAMParam::getChunk()
qbool DAMParam::getChunk()

This function returns qtrue if chunking has been enabled for the parameter, qfalse otherwise.

DAMParam::setBind()
void DAMParam::setBind(eBindType pBind)

The implementation calls this function to set the type of input binding required for the parameter.

This determines when, if at all the parameter will be bound to the server. *enum eBindType* is defined in DAMSTAT.HE

- **pBind** – The type input binding required, i.e. kNoBind, kBindPrepare, kBindExecute, or kBindFetch (kBindOmnis implies that no buffers are required. See eBindType)

DAMParam::getBind()
eBindType DAMParam::getBind()

This function returns the bind type previously assigned to the parameter.

DAMParam::setServerType()
void DAMParam::setServerType(qlong pServerType)

The implementation calls this method to set the server data type to be used to bind the parameter.

Also called at describe time to store the data type and assist with data type conversion later on.

- **pServerType** – The database server type

DAMParam::getServerType()
qlong DAMParam::getServerType()

This function returns the server data type previously assigned to the parameter.

DAMParam::setServerPrecision()
void DAMParam::setServerPrecision(qlong pServerPrecision)

The implementation calls this function to set the precision for the parameter, if required, either at describe time or during input binding. Can be used for numeric or character data.

- **pServerPrecision** – The database data type precision

DAMParam::getServerPrecision()
qlong DAMParam::getServerPrecision()

This function returns the database precision previously assigned to the parameter.

DAMParam::setServerScale()
void DAMParam::setServerScale(qshort pServerScale)

The implementation calls this function to set the (numeric) scale for the parameter, if required, either at describe time or during input binding.

- **pServerScale**– The database data type scale

DAMParam::getServerScale()
qlong DAMParam::getServerScale()

This function returns the database data type scale previously assigned to the parameter.

DAMParam::setBuffers()
qbool DAMParam::setBuffers(qlong pBatchSize)

This function allocates the parameter data and data length buffers. The implementation uses setBufferLen() to establish the size of the data buffer.

- **pBatchSize** – The number of rows to be returned in a batch. The base class multiplies the size of both buffers by this value.

DAMParam::getParamBuffer()
qbyte* DAMParam::getParamBuffer()

This function returns a pointer to the parameter's data buffer. The implementation should not free or resize this buffer. (Use clearBuffers() and setBuffers() if re-sizing is required).

DAMParam::getParamLenBuffer()
qlong* DAMParam::getParamLenBuffer()

This function returns a pointer to the parameter's data-length buffer (a 4-byte buffer) which contains the amount of data written to or placed in the data buffer. The implementation should not free this buffer. Allocation of the parameter data and length buffers is normally handled automatically by the base class.

DAMParam::setOmnisRef()
void DAMParam::setOmnisRef(EXTfldval &pFldVal)

This function sets the parameter's internal field value to a copy of the supplied field value, freeing any previous contents. Resources for the internal field value are handled by the object.

- **pFldVal** – The Omnis field value, passed by reference

DAMParam::setOmnisRef()
void DAMParam::setOmnisRef(qfldval pFldVal)

This function sets the parameter's internal field value based on the supplied qfldval, freeing any previous contents.

- **pFldVal** - The Omnis field value

DAMParam::getOmnisRef()
EXTfldval *DAMParam::getOmnisRef()

This function returns a pointer to the parameter's internal Omnis field value.

DAMParam::clearOmnisRef()
void DAMParam::clearOmnisRef()

This function explicitly clears and frees the parameter's internal field value.

DAMParam::setOmnisVal()

qbool DAMParam::setOmnisVal(qlong pBatchRow, ffttype pOmnisType, qshort pOmnisSubType)

This function sets the parameter's internal field value from data in the parameter's data buffer.

The data will be converted to the specified Omnis type and sub-type if these differ from the internal data type and sub-type. Returns qtrue on success, qfalse otherwise; usually indicating a conversion error.

- **pBatchRow** – When batch fetching, this represents the index of the required row. Pass 1 to access the first (or only) item.
- **pOmnisType** – The Omnis data type required.
- **pOmnisSubType** – The Omnis sub-type required.

DAMParam::setBufferValue()

qbool DAMParam::setBufferValue(qbool pRpc)

This function sets the parameter's data and data length buffers from the value stored in the internal field value. Returns qtrue on success, qfalse otherwise.

- **pRpc** – qtrue if statement is an RPC call. This forces conversion of the internal field value to character data where the parameter's buffer type is fftCharacter.

DAMParam::setBufferScale()

void DAMParam::setBufferScale(qshort pScale)

This function sets the parameter's scale attribute to the supplied value to reflect the numeric scale of the data contained in the data buffer.

DAMParam::getBufferScale()

qshort DAMParam::getBufferScale()

This function returns the previously assigned scale attribute.

DAMParam::setFldVal()

qbool DAMParam::setFldVal(EXTfldval &pParam, qlong pBatchRow)

This function sets the supplied field value from the data in the parameter's data buffer.

The parameter's C type is used to copy and convert the data.

- **pParam** – The Omnis field value that receives the data
- **pBatchRow** – When batch fetching, this represents the index of the required row. A value of 1 represents the first (or only) item.

DAMParam::getDamStorage()

qbyte *DAMParam::getDamStorage()

This function returns a pointer the parameter's DAM storage area; 16 bytes of memory which may be used by the implementation to store additional information about the parameter. It may be useful to interpret this as 4x32bit pointers to user-defined structures. You

will be responsible to allocating and freeing any such structures when parameters get created/destroyed. You should not attempt to free or re-allocate the DAM storage area since this is allocated statically.

DAMParam::setFieldLen()
void DAMParam::setFieldLen(qlong pFieldLen)

This function sets the maximum field length for the parameter, i.e. the Omnis field size.

DAMParam::getFieldLen()
qlong DAMParam::getFieldLen()

This function returns the maximum field length previously assigned to the parameter.

DAMParam::readNextChunk()
eChunkState DAMParam::readNextChunk(qlong &chunkLength)

This function copies a chunk of data of size *chunkLength* from the chunk handle offset into the parameter's data buffer. The chunk handle offset is incremented by chunkLength each time the function is called. (Use setFldChunkHandle() to assign the chunk handle). Returns either kChunkFailed, kChunkOK or kChunkFinished when the last chunk of the source data has been moved. readNextChunk() performs character mapping on the outgoing chunk where applicable.

- **chunkLength** – The size of the chunk to be moved

DAMParam::writeNextChunk()
qbool DAMParam::writeNextChunk()

This function copies the contents of the parameter's data buffer to an internal chunk handle (a *qHandle*), allocating space for the chunk handle if required. Subsequent calls to writeNextChunk() result in data being appended to this handle. Resources for the chunk handle are managed automatically by the object.

DAMParam::setFldChunkHandle()
void DAMParam::setFldChunkHandle()

This function sets the parameter's field value from the contents of the internal chunk handle.

DAMParam::getChunkHandle()
qHandle DAMParam::getChunkHandle()

This function returns the parameter's chunk handle; a pointer to a *qHandleInfo* structure (see basics.h).

DAMParam::clearChunkHandle()
void DAMParam::clearChunkHandle()

This function clears and frees the parameter's chunk handle.

DAMParam::setOffset()
void DAMParam::setOffset(qlong pOffset)

This function is used to store the offset of the parameter's bind marker position in the SQL statement.

The setupInputVars() base class function automatically assigns bind marker offsets at prepare time.

- **pOffset** – Bind marker offset

DAMParam::getOffset()
qlong DAMParam::getOffset()

This function returns the bind marker offset previously assigned to the parameter.

DAMParam::setParamType()
void DAMParam::setParamType(eParameterType pParamType)

This function sets the input/output type for parameters being passed to Remote Procedure Calls. *enum eParameterType* is defined in DAMOBJ.HE

- **pParamType** – The parameter type; one of kParameterInput, kParameterOutput, kParameterInputOutput or kParameterReturnValue

DAMParam::getParamType()
eParameterType DAMParam::getParamType()

This function returns the input/output type previously assigned to the RPC parameter.

DAMParam::setEncoding()
void DAMParam::setEncoding(eSessionEncoding pEncoding)

Sets the Unicode encoding of character data stored in the parameter's data buffer. The base class converts from this encoding to the Omnis encoding (UTF32) when fetching data. When writing non-Unicode data, the base class converts from this encoding to 8-bit data when performing output character mapping. *enum eSessionEncoding* is defined in DAMOBJ.HE. See also mCodePage.

- **pEncoding** – One of: kSessionEncodingUtf32, kSessionEncodingUtf16, kSessionEncodingUtf8 or kSessionEncodingAnsi

DAMParam::getEncoding()
eSessionEncoding DAMParam::getEncoding()

This function returns the Unicode encoding previously assigned to the parameter.

DAMData

DAMData is a container class for each group of parameters used in SQL statements. The statement base class maintains two such groups; mInputParams: for parameters being inserted into SQL statements, and mOutputParams: for result set columns and parameters returned from SQL statements.

Public Methods

DAMData::DAMData()

DAMData::DAMData()

Default constructor. Initialises the object.

DAMData::~DAMData()

DAMData::~DAMData()

The destructor drops all parameters associated with the current object.

DAMData::setParams()

void DAMData::setParams(qshort pNoParams, tqfDAMStatementObj *pStatement)

This function creates a number of empty parameter objects for the specified statement object.

- **pNoParams** – The number of parameters required
- **pStatement** – The statement object

DAMData::numParams()

qshort DAMData::numParams()

This function returns the number of parameters in the group, created previously using the setParams() method.

DAMData::clearParams()

void DAMData::clearParams()

This function resets all the parameters in the group to their default states, calling DAMParam::reset() for each parameter.

DAMData::dropParams()

void DAMData::dropParams()

This function drops all the parameters in the group and frees any resources associated with them.

DAMData::doInline()

qbool DAMData::doInline()

This function determines whether deferred inlining of data is required. doInline() returns qtrue if one or more parameters in the group has a send type set to kDeferInline.

DAMData::doBind()

qbool DAMData::doBind(eBindType pBindType)

This function determines whether binding of a specific type is required. doBind() returns qtrue if one or more parameters in the group has a bind type equal to pBindType. *enum eBindType* is defined in DATSTAT.HE

- **pDoBind** – The bind type required

DAMData::doSend()
qbool DAMData::doSend()

This function determines whether chunking of parameter data is required. doSend() returns qtrue if one or more parameters have their chunking attribute set, i.e. via DAMParam::setChunk()

DAMData::getParam()
DAMParamPtr DAMData::getParam(qshort pParamNum)

This function returns the specified parameter from the group.

- **pParamNum** – The parameter number, 1 specifies the first parameter.

DAMData::setBuffers()
qbool DAMData::setBuffers(qlong pBatchSize)

This function sets up the data buffers for each parameter, equivalent to calling DAMParam::setBuffers() for each parameter in the group. The size required for each parameter buffer is set via DAMParam::setBufferLen()

- **pBatchSize** – The batch size for fetching. The buffer size allocated for each parameter is multiplied by this value.

DAMData::setNoRows()
void DAMData::setNoRows(qlong pRows)

This function sets the number of rows in the current fetch batch. Provided for use by the implementation only. Use is optional.

- **pRows** – The number or rows.

DAMData::getNoRowsPtr()
qlong *DAMData::getNoRowsPtr()

This function returns the address for the variable assigned using setNoRows().

The implementation can bind this to a client variable if required.

DAMData::getNoRows()
qlong DAMData::getNoRows()

This function returns the number of rows previously assigned using setNoRows().

DAMData::numParams()
qshort DAMData::numParams()

Returns the value of mNoParams set previously using setParams().
mNoParams is the number of variables/columns in this group of input/output parameters.

DAMCharMapTable

This class encapsulates the custom character mapping files specified by the user.

Historically, character mapping provides translation between various 8-bit character sets where their displayable characters correspond with differing character codes. Hence, character mapping is only applicable to non-Unicode data, i.e. non-Unicode DAMs and Unicode DAMs operating in non-Unicode mode.

The session base class maintains two DAMCharMapTable objects (mMapIn and mMapOut). The $maptable session property is used to populate the map tables by reading the contents of *filename.IN* and *filename.OUT* respectively.

Public Methods

DAMCharMapTable::DAMCharMapTable()
DAMCharMapTable::DAMCharMapTable()

This constructor initialises the character mapping table.

DAMCharMapTable::~DAMCharMapTable()
DAMCharMapTable::~DAMCharMapTable()

This is the default destructor.

DAMCharMapTable::clear()
void DAMCharMapTable::clear()

This function clears the mapping table, assigning a straight one-to-one mapping to all 256 elements.

DAMCharMapTable::convert()
void DAMCharMapTable::convert(qbyte *pData, qlong pDataLen, eSessionEncoding pEncoding = kSessionEncodingAnsi)

This function converts the string according to the 8-bit character map.

- **pData** – The non-Unicode string to be converted
- **pDataLen** – The length of the string
- **pEncoding** – If specified, pData is assumed to be in this encoding, i.e. Unicode codepoints that contain only ANSI characters. Any non-ANSI codes encountered will generate undefined characters.

DAMCharMapTable::set()
eDAMError DAMCharMapTable::set(str255 &pFileName)

This function loads the character mapping file (a .IN or a .OUT file) and stores the data internally.

- **pFileName** – The name of the character mapping file. The file name is specified as a Unicode string for Unicode DAMs

DAMrpcDefn

This class encapsulates the definition of a Remote Procedure Call. The session base object stores multiple definitions as a linked list (mRpcs).

Public Members

Type	Name	Description
DAMrpcDefn*	mNext	The Linked list of RPC definitions for a session.

Public Methods

DAMrpcDefn::DAMrpcDefn()

DAMrpcDefn::DAMrpcDefn(qchar *pName, qshort pParamCount)

This constructor initialises the class members and allocates memory for the RPC parameters.

The procedure name is case-sensitive.

- **pName** – A null-terminated (Unicode) string containing the name of the RPC
- **pParamCount** – The number of parameters expected by the RPC

DAMrpcDefn::~DAMrpcDefn()

DAMrpcDefn::~DAMrpcDefn()

This destructor frees resources associated with the RPC.

DAMrpcDefn::paramCount()

qshort DAMrpcDefn::paramCount()

This function returns the parameter count previously defined for the RPC.

DAMrpcDefn::paramCount()

void DAMrpcDefn::paramCount(qshort pNewCount)

This function allocates resources for the parameters of the RPC. Any existing parameter definitions are destroyed.

- **pNewCount** – The number of parameters required by the RPC

DAMrpcDefn::params()

DAMrpcParamDefn *DAMrpcDefn::params()

This function returns a pointer to the previously defined array of RPC parameters.

The implementation should not attempt to free this pointer.

DAMrpcDefn::getName()

qchar* DAMrpcDefn::getName(qshort &pNameLen)

This function returns a pointer to the previously assigned name of the RPC and the length of the RPC name (in characters). The implementation should not attempt to free this pointer.

- **pNameLen** – The length of the name (output)

DAMrpcDefn::returnValueIndex(qshort)
void DAMrpcDefn::returnValueIndex(qshort pReturnValueIndex)

This function assigns the index corresponding to the RPC's return-value parameter.

- **pReturnValueIndex** – The parameter number for the return value

DAMrpcDefn::returnValueIndex()
qshort DAMrpcDefn::returnValueIndex()

This function returns the index corresponding to RPC's return-value parameter.

DAMrpcDefn::operator ==()
qbool DAMrpcDefn::operator ==(const strxxx& pRpcName);

This equivalence operator returns qtrue if the supplied name matches the name stored in the object, qfalse otherwise. The match is case-sensitive.

- **pRpcName** – The RPC name to match against the object

DAMunicodeText

This class encapsulates several Unicode conversion functions provided by the component library and provides an interface which is inter-operable between all supported Unicode encodings.

DAMunicodeText can be overridden by the implementation allowing additional functionality to be added if required.

It is safe to use DAMunicodeText objects with both Unicode and non-Unicode targets. For non-Unicode targets, the various member functions simply operate on strings of bytes.

Public Methods

DAMunicodeText::DAMunicodeText()
DAMunicodeText::DAMunicodeText()

Default constructor. Creates and initialises an empty object.

DAMunicodeText::DAMunicodeText()
DAMunicodeText::DAMunicodeText(str255 &pString)

This creates a new object from the contents of a str255 variable to the object.
The parameter can subsequently be discarded.

- **pString** – String variable (encoded as UTF-32 for Unicode targets).

DAMunicodeText::DAMunicodeText()
DAMunicodeText::DAMunicodeText(void *pText, eSessionEncoding pEncoding=kSessionEncodingAnsi, preconst pCodePage=preUniTypeAnsiLatin1)

Creates a new object from the supplied null-terminated text which is in the specified encoding. The contents of pText are copied by the object can can subsequently be discarded.

- **pText** – Null-terminated text string, encoded as either ANSI, UTF8, UTF16 or UTF32
- **pEncoding** – the encoding used by pText, assumed to be kSessionEncodingAnsi if omitted
- **pCodePage** – the ANSI code page used when pEncoding is kSessionEncodingAnsi, Latin1 by default

DAMunicodeText::DAMunicodeText()
DAMunicodeText::DAMunicodeText(void *pText, qlong pCharLen, eSessionEncoding pEncoding=kSessionEncodingAnsi, preconst pCodePage=preUniTypeAnsiLatin1)

Creates a new object from the supplied text which is in the specified encoding. The contents of pText are copied by the object can can subsequently be discarded.

- **pText** – Null-terminated text string, encoded as either ANSI, UTF8, UTF16 or UTF32
- **pCharLen** – the length of pText in character units
- **pEncoding** – the encoding used by pText, assumed to be kSessionEncodingAnsi if omitted
- **pCodePage** – the ANSI code page used when pEncoding is kSessionEncodingAnsi, Latin1 by default

DAMunicodeText::~DAMunicodeText()
DAMunicodeText::~DAMunicodeText()

The default constructor frees any resources used by the object.

DAMunicodeText::concat()
void DAMunicodeText::concat (void *pText, eSessionEncoding pEncoding=kSessionEncodingAnsi, preconst pCodePage=preUniTypeAnsiLatin1

This function concatenates a null-terminated string in the specified encoding to the object.

The source buffer can subsequently be destroyed.

- **pText** – User allocated buffer containing the text to be appended
- **pEncoding** – The Unicode encoding used by pText, assumed to be ANSI if omitted
- **pCodePage** – The code page used when pEncoding is kSessionEncodingAnsi, Latin1 by default

DAMunicodeText::concat()
void DAMunicodeText::concat (void *pText, qlong pCharLen, eSessionEncoding pEncoding=kSessionEncodingAnsi, preconst pCodePage=preU

This function concatenates a string in the specified encoding to the object.

The source buffer can subsequently be destroyed.

- **pText** – User allocated buffer containing the text to be appended
- **pLen** – The length of the source text, in character units
- **pEncoding** – The Unicode encoding used by pText, assumed to be ANSI if omitted
- **pCodePage** – The code page used when pEncoding is kSessionEncodingAnsi, Latin1 by default

DAMunicodeText::concat()
void DAMunicodeText::concat (str255 &pText)

This function concatenates the text inside a str255 class to the object.

The contents of pText are copied and can subsequently be destroyed.

- **pText** – str255 class to be appended

DAMunicodeText::concat()
void DAMunicodeText::concat(DAMunicodeText *pText)

This function concatenates the contents of an existing DAMunicode class to the object.

The contents of pText are copied and can subsequently be destroyed.

- **pText** – DAMunicodeText class to be appended

DAMunicodeText::byteLength()
qlong DAMunicodeText::byteLength()

This function returns the byte length of the data stored in the object converted to the UTF-32 encoding, i.e. 4 bytes for every character stored.

DAMunicodeText::charLength()
qlong DAMunicodeText::charLength()

This function returns the number of characters stored in the object.

DAMunicodeText::dataPtr()
qchar *DAMunicodeText::dataPtr()

This function returns a pointer to the data stored in the object, encoded as UTF-32 by default.

The implementation should not attempt to free or reallocate this pointer.

DAMunicodeText::utf16Ptr()
UChar * DAMunicodeText::utf16Ptr()

This function returns a pointer to a UTF-16 representation of the data stored in the object.

This pointer is only assigned when the data has previously been converted to UTF-16 using convToEncoding(). Otherwise, it returns NULL.

The implementation should not attempt to free or reallocate this pointer.

DAMunicodeText::bytesPtr()
qbyte * DAMunicodeText::bytesPtr()

This function returns a pointer to a non-Unicode/UTF-8 representation of the data stored in the object. This pointer is only assigned when the data has previously been converted to non-Unicode or UTF-8 using convToEncoding(). Otherwise, it returns NULL.

The implementation should not attempt to free or reallocate this pointer.

DAMunicodeText::DAMunicodeText::convToEncoding()
qbyte * DAMunicodeText::DAMunicodeText::convToEncoding(eSessionEncoding pEncoding, preconst pCodePage = preUniTypeAnsiLatin1)

This function returns a pointer to the data stored in the object converted to the specified encoding.

The data returned is null-terminated and the return pointer should be re-cast as appropriate to the encoding. If *kSessionEncodingAnsi* is specified, pCodePage is used to interpret any extended characters.

- **pEncoding** – One of the *eSessionEncoding* constants, defined in DAMOBJ.HE
- **pCodePage** - The ANSI codepage used to interpret non-Unicode data. Defaults to preUniTypeAnsiLatin1 if omitted

General Functions

The following functions are defined in DAMOBJ.HE.

stripTrailing()
qbool stripTrailing(qchar *pStripString, qlong &pLength)

This method strips trailing spaces from the supplied (UTF-32 encoded) string. The string is terminated at the last non-space character and the new character length is returned via pLength.

stripTrailing() returns qtrue if one or more spaces were stripped, qfalse otherwise.

stripCharacter()
qbool stripCharacter(qbyte *pStripString, qlong &pLength, qbyte stripChar)

This method strips all occurrences of the specified character from the source string. The new length is returned via pLength. Returns qtrue if one or more characters were stripped, qfalse otherwise.

Currently works with non-Unicode data only.

usingBundleAsContainer() Mac only
qbool usingBundleAsContainer()

This function checks the info.plist file and returns the status of the usingBundleAsContainer attribute.

The usingBundleAsContainer indicates whether or not Omnis support files are located inside the bundle structure.

usingBundleAsContainer = true : Files are located in Omnis.app/Contents/MacOS

usingBundleAsContainer = false : Files are located in the same folder as the Omnis.app bundle

isLeopard() Mac only
qbool isLeopard()

This method returns qtrue if running under Mac OSX >= 10.5, qfalse otherwise.

Constants and Enumerations

Transaction Mode

The transaction mode constants refer to the different values that can be used for session transaction mode. This can be set using dSetTranMode() in the session implementation and retrieved using getTranMode(). In kTranAutomatic mode a new transaction is started after a successful commit. In kTranManual or kTranServer mode depending on the DBMS a transaction may automatically be started after a $commit()/SQL COMMIT or may require an explicit $begin/SQL BEGIN statement. SQL based transaction commands should only be used in kTranServer mode.

In kTranManual or kTranServer mode the behaviour of the DBMS dictates whether closing a connection commits the current transaction. See commit mode and rollback mode constants for further details.

The effect of a commit or rollback on existing statement cursors is dependant on the behaviour of the DBMS. In most cases a commit or rollback will close all cursors in the session and clear all results sets. This does not destroy the statement object. It may be possible to re-execute the statement but generally the statement will need to be prepared again.

Care should be taken to note the circumstances in which commits occur as this can have a side effect on the processing of other statement objects associated with the session.

eTranMode	Value	Description
kTranAutomatic	0	Automatic Transaction Mode. Each statement is automatically committed after execution. This is the default and specifies that all transaction management is provided automatically. This means that after a command has successfully executed, i.e. $execute() or $execdirect() returns kTrue then the current transaction is committed. If the command fails then the transaction is rolled back.
kTranManual	1	Manual Transaction Mode. Statements or groups of statements require explicit commit or rollback. Allows the application to manage transactions via the use of the session methods $begin, $commit and $rollback.
kTranServer	2	Server Transaction Mode. Server specific transaction handling. Transaction management is provided by the DBMS. The application may also execute SQL BEGIN, COMMIT and ROLLBACK statements to manage transactions depending on the DBMS.

Commit Mode

The commit mode constants refer to the different behaviour of statement objects after a commit has been issued. This behaviour is defined by the database server and can be retrieved using dGetCommitMode() on the derived session object.

eCommitMode	Value	Description
kCommitDelete	0	Prepared statements and cursors on all statement objects are deleted. Any pending results are lost. The statement object itself is not deleted but will be set to a kStateClear state. To re-execute the same statement it must first be re-prepared.
kCommitClose	1	Prepared statements and cursors on all statement objects are closed. Any pending results are lost. A statement can be re-executed without first being re-prepared. Any statement objects which have successfully prepared a statement will be in the kStatePrepared state.
kCommitPreserve	2	The state of all statements and cursors remains unchanged.

Rollback Mode

The rollback mode constants refer to the different behaviour of statement objects after a rollback has been issued. This behaviour is defined by the database server and can be retrieved using dGetRollbackMode() on the derived session object.

eRollbackMode	Value	Description
kRollbackDelete	0	Prepared statements and cursors on all statement objects are deleted. Any pendir results are lost. The statement object itself is not deleted but will be set to a kState state. To re-execute the same statement it must first be re-prepared.
kRollbackClose	1	Prepared statements and cursors on all statement objects are closed. Any pending results are lost. A statement can be re-executed without first being re-prepared. A statement objects which have successfully prepared a statement will be in the kStatePrepared state.
kRollbackPreserve	2	The state of all statements and cursors remains unchanged.

Session state

The session state indicates whether the session object has logged on to the database server. The value is usually set using setState() on the session object after a logon or logoff. The value can be retrieved using getState() on the session object.

eSessionState	Value	Description
kSessionLoggedOff	0	The session is not logged on to the database.
kSessionLoggedOn	1	The session is logged on the database.

Type Entry

The type entry constants refer to the Omnis data types and are used in the Session type tables for mapping to C types. The type table is created during the construction of the DAMTypeTable and the type entries can be viewed or modified using the getCType() function on the DAMTypeTable object.

eTypeEntry	Value	Description
kTypeCharacter	0	Omnis Character Type
kTypeBoolean	1	Omnis Boolean Type
kTypeDateTime	2	Omnis Date Time Type
kTypeDate	3	Omnis Date Type
kTypeTime	4	Omnis Time Type
kTypeSequence	5	Omnis Sequence Type
kTypeNumber	6	Omnis Number Type
kTypeShortNumber	7	Omnis Short Number Type
kTypeFloat	8	Omnis Float Type
kTypeInteger	9	Omnis 32-bit Integer Type
kTypeShortInteger	10	Omnis Short Integer Type
kType64bitIntger	11	Omnis 64-bit Integer Type
kTypePicture	12	Omnis Picture Type
kTypeBinary	13	Omnis Binary Type
kTypeList	14	Omnis List Type
kTypeRow	15	Omnis Row Type
kTypeObject	16	Omnis Object Type
kTypeItemRef	17	Omnis Item Reference Type
kTypeUnknown	18	Default mapping to binary
kTypeLast	kTypeUnknown	Always maps to the last type entry

DAM Errors

The internal DAM Error constants are used to identify an area where a failure occurred. Usually the native error code and error text are more specific. The error code is set using setErrorCode() in the DAMErrorInfo class.

eDAMError	Value	Description
kDAMNoError	0	No error has been reported
kDAMUnknownError	-1	An unknown error has been reported
kDAMNativeErrorFailed	-2	The native error could not be reported
kDAMNoPendingError	-3	There is no pending error to report
kDAMNotUnicode	-4	Not supported by a Unicode DAM
kDAMWrongOmnisVersion	-5	DAM cannot be used with this version
kDAMInternalError	-9	An internal error occurred. Contact Technical Suppo
kDAMParameterError	-10	A method parameter could not be processed
kDAMParameterNumError	-11	The number of method parameters is incorrect
kDAMParameterTypeError	-12	The type of a method parameter is incorrect
kDAMInvalidMapType	-14	Not a valid map type
kDAMInvalidTableType	-15	Not a valid table type
kDAMInvalidIndexType	-16	Not a valid index type
kDAMInvalidTranMode	-17	Not a valid transaction mode
kDAMInvalidFileName	-18	The specified file cannot be opened
kDAMServerPropertyFailed	-19	The server specific property could not be set
kDAMServerMethodFailed	-20	The server specific method failed
kDAMFileOpenFailed	-21	The file could not be opened
kDAMFileReadFailed	-22	The file could not be read
kDAMNotSerialised	-23	The DAM has not been serialised
kDAMSqlError	-24	Error parsing/processing SQL text
kSessionLogonFailed	-30	A connection could not be established
kSessionLogoffFailed	-31	The connection could not be terminated
kSessionNotLoggedOn	-32	The session must be logged on
kSessionRegisterFailed	-33	The session could not be registered - perhaps the na
kSessionRpcDefineFailed	-34	The RPC could not be defined
kSessionRpcNotDefined	-35	The specified RPC is not defined
kSessionClearFailed	-36	An error occurred when clearing the session
kSessionNoDriverManager	-37	ODBC Driver Manager not found
kSessionNotTranManual	-40	The transaction mode is not manual
kSessionCommitFailed	-41	The commit failed
kSessionRollbackFailed	-42	The rollback failed
kSessionSetTranModeFailed	-43	The transaction mode could not be set
kSessionNoTranSupport	-44	The DBMS does not support transactions
kSessionUsesAutoBegin	-45	Transactions are automatically started
kSessionBeginFailed	-46	A transaction could not be started
kSessionLobThresholdOutOfRange	-50	The threshold specified is out of range
kSessionLobChunkSizeOutOfRange	-51	The chunk size specified is out of range
kSessionNewStatementFailed	-60	The statement object could not be created
kSessionColTextFailed	-61	The coltext method failed
kSessionBindMarkerFailed	-62	The bind variable marker could not be read
kSessionSetMapTableFailed	-63	The character map table could not be set
kSessionSetCharMapFailed	-64	The session's character map could not be set
kSessionNoMapTable	-65	There is no map table specified
kSessionInvalidSQLSeparator	-66	An invalid separator character was specified
kStatementPrepareFailed	-80	The statement could not be prepared
kStatementExecuteFailed	-81	The statement could not be executed
kStatementDescribeFailed	-82	The result set could not be described
kStatementFetchFailed	-83	The rows could not be fetched
kStatementClearFailed	-84	The statement could not be cleared
kStatementCloseFailed	-85	The statement could not be closed
kStatementDropFailed	-86	The statement could not be dropped
kStatementExecDirectFailed	-87	The statement could not be executed directly
kStatementRpcPrepareParamsError	-88	Error preparing RPC parameters
kStatementRpcGetParamValuesError	-89	Error getting RPC parameter values
kStatementHasNoSession	-90	The statement is not associated with a session
kStatementReturnedInfo	-91	The statement returned information in $nativeerror
kStatementInsertRowFailed	-100	A row could not be added to the list

eDAMError	Value	Description
kStatementInputVarFailed	-101	An error occurred with an input variable
kStatementResultColFailed	-102	An error occurred with a result column
kStatementBindInputVarFailed	-103	An error occurred binding an input variable
kStatementSendInputVarFailed	-104	An error occurred sending an input variable
kStatementBindResultColFailed	-105	An error occurred binding a result column
kStatementSendResultVarFailed	-106	An error occurred sending a result column
kStatementBufferAllocationFailed	-107	The data buffers could not be allocated
kStatementServerTablesFailed	-120	The tables could not be queried
kStatementServerColumnsFailed	-121	The columns could not be queried
kStatementServerIndexesFailed	-122	The indexes could not be queried
kStatementServerResultsFailed	-123	The result set could not be queried
kStatementServerRpcProceduresFailed	-124	The RPC procedures could not be queried
kStatementServerRpcParametersFailed	-125	The RPC procedure parameters could not be queried
kStatementNotPrepared	-130	The statement must be prepared
kStatementNoFetchList	-131	There is no list to fetch into
kStatementNoResults	-132	There are no results to process
kStatementNoFilename	-133	No filename was specified

Session Character Map

The session character map refers to the character set used for mapping non-Unicode character data by the session object.

eSessionCharMap	Value	Description
kSessionCharMapOmnis	0	Session uses Omnis Macintosh character set
kSessionCharMapNative	1	Session uses platform native character set
kSessionCharMapTable	2	Session uses character mapping table specified by $maptable

Character Map Mode

The character map mode refers to the direction of mapping. This is used by dAllowsCharConversion() in the derived session object to permit or disable character mapping.

eCharMapMode	Value	Description
kCharMapIn	0	Mapping from the server to the DAM
kCharMapOut	1	Mapping from the DAM to the server

Unicode Encoding

The Unicode encoding constants describe the encoding of their accompanying character data.

Unicode encoding constants are used by DAM parameters and the DAMunicodeText class.

eSessionEncoding	Value	Description
kSessionEncodingUtf32	0	The data uses the UTF-32 encoding; 4 bytes per character
kSessionEncodingUtf16	1	The data uses the UTF-16 encoding; two bytes per character
kSessionEncodingUtf8	2	The data uses the UTF-8 encoding; 1 to 4 bytes per character
kSessionEncodingAnsi	3	The data uses 1 byte per character and belongs to the specified 8-bit codepa

Error State

The error state indicates the outcome of retrieving the native error.

eErrorState	Value	Description
kErrorFailed	0	Unable to retrieve native error
kErrorOk	1	Native error successfully returned
kErrorPending	2	Native error pending

Statement Level State

The statement state constants refers to the various states that a statement object can be in.

eStatementState	Value	Description
kStateClear	0	No active operation or result set in this object
kStatePrepared	1	A statement has been successfully prepared in this object
kStateExecuted	2	A statement has been successfully executed in this object
kStateFetching	3	A result set is being processed in this object
kStateExecDirect	4	A statement has been successfully executed directly in this object

Fetch Call State

The fetch call state constants refers to the various states that a statement object can be in after $fetch() has been called.

eFetchStatus	Value	Description
kFetchOk	0	The method completed successfully; there may be more rows to fet
kFetchFinished	1	The method completed successfully; there are no more rows to fetch
kFetchError	2	An error occurred while executing the method
kFetchMemoryUsageExceeded	3	Some rows could not be fetched because $maxresultsetsize was exc
kFetchRowAdded	4	The row has been added take no further action (internal use only).
kFetchAll	1000000000	When passed as the fetch cap to list.$fetch, fetch all the rows

Table Types

The table type constants refer to the various types of parameter that can be passed to the $tables() method in order to filter the result set.

eTableType	Value	Description
kStatementServerAll	0	All Tables
kStatementServerTable	1	Tables only
kStatementServerView	2	Views Only

Index Types

The index type constants refer to the various types of parameter that can be passed to the $indexes() method in order to filter the result set.

eIndexType	Value	Description
kStatementIndexUnique	0	All Tables
kStatementIndexNonUnique	1	Tables only
kStatementIndexAll	2	Views Only

Parameter Types

The parameter types refer to the type of parameter being bound. This is stored in the DAMParam object and can be set using the setParamType() function and retrieved using the getParamType() function.

eParameterType	Value	Description
kParameterInput	1	Parameter is input type only
KParameterOutput	2	Parameter is output type only
kParameterInputOutput	3	Parameter is both input and output
kParameterReturnValue	4	Parameter is a return value

Parameter Send Types

The send type constants indicate how the data for a parameter should be handled during execution of a SQL statement. Use DAMParam::setSendType() to change a parameter's send type or getSendType() to retrieve it.

eSendType	Value	Description
kConvert	0	For input parameters, this value indicates that the contents of the parameter's buffer require conversion before the data is written to the database. For output parameters, this value indicates that the data requires conversion before being returned to Omnis. dConvetParam() is called at the appropriate point for any parameters with this send type.
kInline	1	This value indicates that the parameter buffer contents should be concatenated onto the SQL statement at the bind marker position. Inline expansion usually requires data to be escaped by the implementation to avoid syntax errors.
kDefault	2	This value indicates that the parameter value should be bound and sent as normal via the implementation's dBindParameter() method. The parameter's bindType governs at what point dBindParameter() will be called.
kDeferInline	3	This value indicates that the parameter's data should be sent before the statement has been executed using the implementation's dProcessInlines() method.

Parameter Bind Types

The bind type indicate when binding can should take place during execution of a SQL statement.

eBindType	Value	Description
kNoBind	0	No binding required
kBindPrepare	1	Binding takes place during prepare
kBindExecute	2	Binding takes place during execute
kBindFetch	3	Binding take place during fetch
kBindOmnis	4	Use Omnis fldvals only, no buffers required

Parameter C-Types

The C-Type of a parameter is an implementation-independent type used for the purposes of converting and transferring data to and from the database. The table indicates the default assignments made by the statement object base class when creating its DAMTypeTable object. The implementation is free to change these assignments using its derived type table object.

Use DAMParam::setCType() to set a parameter's C-Type, or getCType() to retrieve it.

eCType	Value	Meaning	Default mapping
omChar	0	Character data	Omnis character variables (and unk
omSShort	1	Signed short integers	Not used
omUShort	2	Unsigned short integers	Not used
omULong	3	Unsigned long integers	Not used
omSLong	4	Signed long integers	for Omnis Sequence and Integer va
omFloat	5	Floating point decimal numbers	Not used
omDouble	6	Long decimal numbers	Omnis Number, Short number and
omSTinyint	7	Signed tiny integers	Not used
omUTinyint	8	Unsigned tiny integers	Omnis Short integer variables
omBinary	9	Binary data	Omnis Picture, Binary, List, Row, Ob
omDate	10	Dates	Omnis Date variables
omTime	11	Times	Omnis Time variables
omTimeStamp	12	Dates with Times	Omnis DateTime variables
omReal	13	Short decimal numbers	Not used
omBit	14	One bit data	Omnis Boolean variables
omList	15	List data	Not used
omSLong64	16	Signed 64-bit integers	Omnis Integer k64bitint variables

Chunk States

The chunk state constants indicate the various responses after a chunk of data is processed.

eChunkState	Value	Description
kChunkOk	0	Success. More data may be available
kChunkFailed	1	Error when processing chunk
kChunkFinished	2	Success. No data remaining

Add Row Modes

The add row mode constants indicate the mapping or any specific processing that addRow should perform when returning data to the list or row.

DAMstaAddRowMode	Value	Description
eDAMaddRowNormal	0	Data has come form a normal results set.
eDAMaddRowTables	1	Meta data about the servers tables
eDAMaddRowColumns	2	Meta data about a tables columns
eDAMaddRowIndexes	3	Meta data about a tables indexes
eDAMaddRowRpcProcedures	4	Meta data about the servers RPCs
eDAMaddRowRpcParameters	5	Meta data about an RPC's parameters

DAM-Specific External Component Library Callbacks

The component library provides additional callbacks that are not documented in the Omnis Component Library Tutorial. These are provided specifically for use by Omnis DAMs. They enable the DAM to call into and access information from the Omnis core.

DAMprepareBindVariables()
qret DAMprepareBindVariables(EXTCompInfo *pEci, EXTfldval &pInStatement, qchar *pPlaceHolder, EXTfldval &pOutStatement, DAMbindVariables &pBindVariables)

This callback parses the SQL statement and replaces any bind variables with the required database specific bind markers. The bind variable attributes are then initialised in the DAMBindVariable structure. Called from tqfDAMStatementObj::setStatementText()

- **pEci** – The EXTCompInfo object with information regarding the current DAM function.

- **pInStatement** – The original statement with bind variables
- **pPlaceHolder** – The database specific bind marker
- **pOutStatement** – The modified statement with bind markers (output)
- **pBindVariables** – The bind variables (output)

DAMallocBindVariables()

void DAMallocBindVariables(EXTCompInfo *pEci, DAMbindVariables &pBindVariables, qshort pBindCount)

This callback allocates the resources for the specified number of bind variables.

Called from tqfDAMStatementObj::rpcDefnToBindVars()

- **pEci** - The EXTCompInfo object with information regarding the current DAM function.
- **pBindVariables** – The statement object's bind variable structure
- **pBindCount** – The number of bind variables

DAMgetBindVariableValues()

qret DAMgetBindVariableValues(EXTCompInfo *pEci, DAMbindVariables &pBindVariables)

This callback populates the bind variable structure with the current values of the Omnis bind variables. Called from tqfDAMStatementObj::execute() and tqfDAMStatementObj::execdirect()

- **pEci** - The EXTCompInfo object with information regarding the current DAM function.
- **pBindVariables** – The statement object's bind variable structure

DAMfreeBindVariables()

void DAMfreeBindVariables(DAMbindVariables &pBindVariables, qbool pFreeVariablesArray);

This function frees the resources associated with the bind variables. The implementation does not need to call this, as the DAMbindVariables destructor will do so. (Called from tqfDAMStatementObj:: setStatementText())

- **pBindVariables** - The bind variable structure
- **pFreeVariablesArray** – qtrue if the Bind variable array should be freed as well.

DAMgetDataTypeText()

void DAMgetDataTypeText(qlong pType, qshort pSubType, qlong pLen, str255 *pTypeText)

This callback returns the Omnis data type text for the specified type and sub-type.

Called from tqfDAMStatementObj::metaDataResults() and tqfDAMStatementObj::results()

- **pType** - The ffttype for the variable
- **pSubType** – The Omnis sub type for the variable
- **pLen** – The length of the variable
- **pTypeText** – The text string representation of the type (output)

DAMconvcset() Deprecated for Unicode DAMs

qbool DAMconvcset(csettype pCSetIn, csettype pCSetOut, qchar *pData, qlong pDataLen)

This callback converts between two 8-bit character sets. The base class no longer uses this callback because the Studio 5 core no longer supports kSessionCharMapOmnis or kSessionCharMapNative, which are still required by Unicode DAMs operating in non-Unicode mode. The base class uses the tqfDAMbaseObj::convertToCharacterSet() private method instead, called from tqfDAMbaseObj::charMapOut() and tqfDAMbaseObj::charMapIn().

- **pCSetIn** – The current character set of the data, defined in BASICS.H
- **pCSetOut** – The character set to map too.
- **pData** – The non-Unicode data to be converted
- **pDataLen** – The length of the data

DAMgetDateString()

void DAMgetDateString(datestamptype *pDate, str255 *pDateFormat, str255 *pOutString)

This callback returns a datetime formatted according to the specified format string. Date time format specifiers are shown in the Omnis Catalog under *Date codes*. Called from DAMParam::setBufferValue().

- **pDate** – The datestamptype structure containing the datetime value
- **pDateFormat** – The date format for the string
- **pOutString** – The string containing the converted date (output)

DAMregisterSession()

qbool DAMregisterSession(EXTCompInfo *pEci, str255 *pHostName, str255 *pUserName, str255 *pPassword, str255 *pSessionName)

This callback registers the session with the Omnis $sessions group.

This callback is used by tqfDAMbaseObj::logon().

- **pEci** - The EXTCompInfo object with information regarding the current DAM function.
- **pHostName** – The database host name
- **pUserName** - The database user name
- **pPassword** – The users password
- **pSessionName** – The session name to appear in $root.$sessions

DAMunregisterSession()

void DAMunregisterSession(str255 *pSessionName)

This callback removes the specified session from the $sessions group.

Used by tqfDAMbaseObj::logoff().

- **pSessionName** - The session name as it appears in $root.$sessions

DAMputBindVariableValues()
qret DAMputBindVariableValues(DAMbindVariables &pBindVariables)

This callback places modified values returned from the server back into the original Omnis bind variables. This callback is provided for use by the implementation in processing output bind variables.

- **pBindVariables** – The bind variables to be written

DAMgetSeparators()
qret DAMgetSeparators(str15 &pSeparators)

This callback places the result of $root.$prefs.$separators() into the supplied string.

- **pSeparators** – string object that receives the separator characters

DAMreleasePoolRef()
qbool DAMreleasePoolRef(qobjinst pInst)

For use by worker delegate objects, this callback releases a session pool reference claimed when the worker object was initialised (using the “poolname” parameter).

- **pInst** – The session pool object reference. See StatementWorkerDelegate::dInit()

DAMgetThreadLocalStorage()
void DAMgetThreadLocalStorage(DAMthreadLocalStorage *tls)

For internal use by the worker object base class, this callback populates a custom structure with information about the main Omnis thread, i.e. global variables and call stack information. This is used by the worker delegate object when calling back to the main thread.

- **tls** – pointer to a structure that receives the thread local storage information

Appendix A—Debugging macOS Components

Introduction

On macOS 11+, Apple have disabled the ability to debug a plug-in/component within the context of a host executable which has been notarized.

This is a security measure to prevent malicious attempts to inject code at runtime.

Our deployed versions of Studio are all notarized to prevent malware.

However, it is possible to duplicate a locally installed version and add the entitlement to allow debugging of the copy.

To setup up a copy of Omnis Studio to allow debugging, do the following:

- Duplicate the already installed version, e.g. Studio 30204. Rename it so it is identifiable as a version that can be debugged, e.g. Omnis Studio 10.2 30204 Debug.

- Create the file debug_entitlements.xml with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>com.apple.security.automation.apple-events</key>
    <true/>
    <key>com.apple.security.cs.disable-library-validation</key>
    <true/>
    <key>com.apple.security.cs.allow-dyld-environment-variables</key>
    <true/>
    <key>com.apple.security.device.audio-input</key>
    <true/>
    <key>com.apple.security.device.camera</key>
    <true/>
    <key>com.apple.security.personal-information.location</key>
    <true/>
    <key>com.apple.security.personal-information.addressbook</key>
    <true/>
    <key>com.apple.security.personal-information.calendars</key>
    <true/>
    <key>com.apple.security.personal-information.photos-library</key>
    <true/>
    <key>com.apple.security.get-task-allow</key>
    <true/>
  </dict>
</plist>
```

- Re-sign the debug version with the debug_entitlements.xml file. This adds the com.apple.security.get-task-allow entitlement to allow debugging.

From a terminal do the following:

```
$ codesign -f --deep -s - --entitlements debug_entitlements.xml /Applications/Omnis\ Studio\ 10.2\ 30204\ Debu
```

This should result in the output:

```
/Applications/Omnis Studio 10.2 30204 Debug.app: replacing existing signature
```

This command will re-sign the whole Omnis tree using an ad hoc signature (for local use) and apply the entitlements to allow debugging.

Xcode can now point to the debug version of Omnis Studio in the Run action of the targets Scheme.

Appendix B—Building macOS Universal Components

To build macOS components that run natively on both Intel and M1 computers (Universal components) in Omnis Studio, you need to use of Xcode 12+. The Omnis resource compiler is now a macOS Universal binary and should replace the version in the Xcode tree:

```
/Applications/Xcode.app/Contents/Developer/Tools/omnisrc64.app
```

The macOS resource compiler now expects a UTF-8 encoding for the strings it reads from a resource file. It will write a message to the build log if a string is encoded incorrectly. Although a message is written to the build log, the build continues, however, you get a warning about an incomplete Localizable.strings file, and this is the indicator that you need to look at the build log.

This is only really an issue for a small number of text resources, e.g. extended ASCII codes such as the copyright symbol or the £ character.

A component which is currently using the existing Intel only 10.2 SDK with Xcode 11 should be compatible with Xcode 12. Once opened with Xcode 12, this should build out a Universal binary version (a fat x86_64 / arm64 binary).

The important Xcode build settings are:

- **Architectures**. Should be
  - $(ARCHS_STANDARD_64_BIT)
  - or $(ARCHS_STANDARD) / Standard Architectures (Apple Silicon, Intel).
- **Build Active Architecture Only** - Should be No to build out both supported architectures. Typically, this will be No for Deployment and Yes for Development.

The generic and jsgeneric components provide a good starting template for a project.

The component library and any third-party libraries that a component uses **MUST** also be Universal to build out a Universal component.

To check the architecture included in a binary use the lipo command from the Terminal, e.g.

```
% lipo -info /Users/macminisilicon/Downloads/OSX-SDK-10.2-31054-Beta/_OSXUnicode/generic.u_xcomp/Contents/MacO

Architectures in the fat file: /Users/macminisilicon/Downloads/OSX-SDK-10.2-31054-Beta/_OSXUnicode/generic.u_x
```

For a binary to load natively on both Intel and M1 computers it must include both the x86_64 and arm64 architectures.

Note: An Intel only component cannot be loaded by the Universal version of Studio running natively on Apple M1 computers.