# 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; };

1. **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).
2. **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.
3. **mReturnDataType** - Returned data type of type fftxxx. Specify 0 for no returned data (e.g. void) and fftNone for an unspecified data type.
4. **mParameterCount** - Number of parameters for the method. Specify zero for no parameters.
5. **mParameters** - Pointer to an array of parameters. Specify NULL if there are no parameters.
6. **mFlags** - Method flags of type EXTD\_FLAG\_xxxx.
7. **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; };

1. **mNameResID** - Resource id which contains the parameters’ name.
2. **mDataType** - fftxxx data type of the parameter. Use fftNone for an unspecified data type.
3. **mFlags** - Parameter flags of type EXTD\_FLAG\_xxxx. Examples are EXTD\_FLAG\_PARAMOPT and EXTD\_FLAG\_PARAMALTER.
4. **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, 0, 0, 0, 0, 0 };

###### // 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\* eci ) {   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\* eci ) {   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\* eci ) {   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

This is the structure which is passed from Omnis to the components’ message function. EXTCompInfo contains all the information needed to process ECM\_xxxx messages.

###### 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. For example, the following property entry (extract from QuickTime) indicates that the property is a external (i.e. Component) constant between constant ids, 23000 & 23004.

###### eQTIME\_Movie\_scaling, 25017, fftNumber, EXTD\_FLAG\_EXTCONSTANT, 0, 23000, 23004

### 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\_PRIMARYDATACHANGE

### 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

Indicates that the popup multi line edit window should be provided.

### 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.

**Parameters:**

* **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\* eci ) {   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.

**Parameters:**

* **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\* eci ) {   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.

**Returns:**

The component should return DATA\_CMPDATA\_SAME if the data is the same, or DATA\_CMPDATA\_DIFFER if the data is different, false otherwise.

###### extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo\* eci ) {   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\* eci ) {   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\* eci ) {   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.

**Parameters:**

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

**Returns:**

Return qtrue if the picture data has had any headers added or removed, false otherwise.

###### extern "C" qlong OMNISWNDPROC PCXWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) { 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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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.

###### example strings (extracts from QuickTime component):

###### // Scaling constant group 23000 "Scaling~kQTScaleNone:0:kQTScaleNone:No Scaling is applied to the movie." 23001 "kQTScaleNoAspectRatio:1:kQTScaleNoAspectRatio:The movie is expanded to fit the current field." 23002 "kQTScaleKeepAspectRatio:2:kQTScaleKeepAspectRatio:The movie is expanded to fit the current field." 23003 "kQTScaleProportional:3:kQTScaleProportional:The movie is equally expanded vertically and horizontally to fit the current field." 23004 "kQTScaleField:4:kQTScaleField:The movie's field is expanded around the movie."

###### // Resource slots 23005-23009 left for future scaling options // Controller constant group

###### 23010 "Controller~kQTnoButtons:0:kQTnoButtons:The Controllers all buttons list." 23011 "kQTstepButton:1:kQTstepButton:The Controllers step and reverse button are removed." 23012 "kQTsoundButton:2:kQTsoundButton:The Controllers sound button are removed." 23013 "kQTgrowButton:4:kQTgrowButton:The Controllers grow button area are removed."

###### extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo\* eci ) {   ECOsetupCallbacks(hwnd,eci);   switch (Msg)   {     case ECM\_GETCONSTNAME:     {       return ECOreturnConstants( gInstLib, eci, 23000, 23013 );     }   }   return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci); }

**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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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..

**Returns:**

Return qtrue if the component supports a picture format conversion, qfalse otherwise.

###### 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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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 data name. **mDrawFocusRect** - True if the focus rectangle should be drawn.

###### extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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 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\* eci ) {   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\* eci) {   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\* eci ) {   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\* eci ) {   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).

**Returns:**

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

**See also** ECM\_LISTDRAWLINE, cObjType\_DropList

### 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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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\* eci ) {   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.

**Returns:**

Return true if successful, false otherwise.

###### extern "C" qlong OMNISWNDPROC GenericWndProc( HWND hwnd, LPARAM Msg, WPARAM wParam, LPARAM lParam, EXTCompInfo\* eci ) {   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.

**Returns:**

Return 1L if you wish to prevent Omnis from duplicating the object.

### 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).

**Returns:**

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

###### 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\* eci ) {   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\* eci ) {   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 ) |

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.

###### str255 myTraceLine(“Some trace information”); ECOaddTraceLine( &myTraceLine );

### 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() (v2.4)

|  |
| --- |
| 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() (v5.1)

|  |
| --- |
| 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() (v2.4)

|  |
| --- |
| 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() (v3.1)

|  |
| --- |
| 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() (v3.1)

|  |
| --- |
| 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() (v2.4)

|  |
| --- |
| 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() (v3.3)

|  |
| --- |
| 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() (v3.3)

|  |
| --- |
| 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() (v4.2)

|  |
| --- |
| 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);

### 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

### 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\* eci) {   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\* eci ) {   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() (v5.0)

|  |
| --- |
| 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.

### 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() (v3.1) Mac OSX only

|  |
| --- |
| void \*ECOgetBundleRef(qlong pBundleID) |

Returns a CFBundleRef dependant on the pBundleID.

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

### ECOgetCrbFieldInfo() (V2.2)

|  |
| --- |
| 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 … }

### 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 ); }

### 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\* eci ) {   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); }

### ECOgetLocalIpAddress() (v4.3)

|  |
| --- |
| qulong ECOgetLocalIpAddress(void) |

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

### ECOgetNVObject() (v3.3)

|  |
| --- |
| 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\* eci ) {   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() (v3.1)

|  |
| --- |
| 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 ); }

### 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 );

### 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.

### 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.

### 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\* eci) {   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\* eci) {   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() (v3.1)

|  |
| --- |
| 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. }

### ECOisMultiSelected()

|  |
| --- |
| qbool ECOisMultiSelected( HWND pHWnd ) |

Allows the component to inquire on whether the object is currently one of many objects selected.

* **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

### 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.

### 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.

5001 "PCX Files (\*.pcx)|\*.pcx|All Mac Text Files|’’,’TEXT’|"

**Note:** Under MacOS you can specify both or either the finder creator & type code, for example, |Omnis Libraries|’OO$$’,’OO$A’|All Omnis Files|’OO$$’,’’|. Under Windows the MacOS specific file filter is ignored.

* **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); }

### ECOmapString() (v5.0)

|  |
| --- |
| 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.

### 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() (v3.3)

|  |
| --- |
| 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 }

### ECOpaintGrayFrame() (v5.0)

|  |
| --- |
| 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); }

### 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.

### ECOreloadLibData() (v4.1)

|  |
| --- |
| 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\* eci) {   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\* eci) {   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

### 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 ECOreturnMethods. 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

### 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() (Web Client 1.2)

|  |
| --- |
| 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

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 ); }

### 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 ECOsendCompEvent.

* **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

### 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).

An example of use may be :-

###### // 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 );

### 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

### 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\* eci) {   ECOsetupCallbacks(hwnd, eci);   switch (Msg)   {   }   return WNDdefWindowProc(hwnd,Msg,wParam,lParam,eci); }

### 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() (v5.1)

|  |
| --- |
| 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.

### 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.

1. **pHandle**-The memory to be handed back into the Omnis memory cache.
2. **pOffset** -The offset into the memory.

### qHandlePtr::qHandlePtr()

|  |
| --- |
| qHandlePtr (const qHandlePtr& pHptr) |

Constructs a qHandlePtr class from an existing qHandlePtr.

1. **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.

1. **pHptr-** an Existing qHandlePtr class.

### qHandlePtr::operator +=()

|  |
| --- |
| void operator +=(qlong pInc) |

Increments the offset in to memory block.

1. **pInc-** The amount to increment the offset.

### qHandlePtr::operator -=()

|  |
| --- |
| void operator -=(qlong pDec) |

Decrements the offset in to memory block.

1. **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.

1. **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.

1. **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 :-

1. **returns** - Memory block base + Offset.

### qHandlePtr::operator \*()

|  |
| --- |
| qchar\* operator \*(qlong pDel) |

Return a qchar pointer which is calculated as :-

1. **returns** - Memory block base + Offset + pDel.

### qHandlePtr::operator []()

|  |
| --- |
| qchar& operator [](qlong pDel) |

Return a qchar reference which is calculated as

1. **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.

1. **returns** - Data Length of the Handle.

### qHandlePtr::dataLen()

|  |
| --- |
| void dataLen(qulong pSize) |

Sets the actual length of the data contain in the handle.

1. **pSize** - Sets the Data Length of the handle.

### qHandlePtr::getOffset()

|  |
| --- |
| qulong getOffset() |

Returns the current offset into the memory block

1. **returns** - offset into the memory block.

### qHandlePtr::getHandle()

|  |
| --- |
| void getHandle(qHandle& pHandle) |

Returns the handle of the qhandleptr

1. **pHan** - a qHandle memory block.

### qHandlePtr::set()

|  |
| --- |
| void set(qHandle pHandle, qlong pOffset) |

Sets the qhandleptr from the provided parameters

1. **pHandle** - a qHandle memory block.
2. **pOffset** - Offset into the memory block.

### qHandlePtr::setOffset()

|  |
| --- |
| void setOffset(qlong pOffset) |

Set the Offset of the qhandleptr.

1. **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

### RESgetOmnisDAT()

|  |
| --- |
| 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.

**Example:**

###### 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