opentext™

GroupWise® Software Developer Kit

Custom Third-Party Object (C3PO™)

October 2023

Legal Notices

Copyright 1993 - 2023 Open Text.

The only warranties for products and services of Open Text and its affiliates and licensors (“Open Text”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Open Text shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Contents

About This Guide

GroupWise C3PO (Custom 3rd-Party Object) is an OLE COM Server object that is used to extend the GroupWise environment. The extensions take various forms, ranging from implementing custom record types (objects) in the GroupWise data store to modifying the GroupWise browser or toolbar functionality. A C3PO can be developed by using C++, Delphi, or Visual Basic.

IMPORTANT: Unless otherwise marked, the features in GroupWise C3PO work with GroupWise 8 and later versions.

This guide contains the following sections:

Audience

This guide is intended for GroupWise developers.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comment feature at the bottom of each page of the online documentation and enter your comments there.

Additional Documentation

For additional GroupWise SDK documentation, see the GroupWise Web site.

1 Overview

GroupWise Custom 3rd-Party Object (C3PO) software combines and replaces the older and less functional notion of custom messages and custom commands.

A C3PO can make the following modifications to the user interface:

- Add new menus, menu items, and separators to existing menus.
- Add new buttons, with user-defined bitmaps, to the toolbar.
- Define new custom message types with their own custom icons.

The ability to modify the user interface allows a C3PO to appear as an integrated feature of GroupWise.

A C3PO can also intercept a number of GroupWise commands and events, allowing the user to customize the handling of these commands and events.

This section covers the following topics:

C3PO Functionality

One way to run a C3PO is to select the menu item or toolbar button that was added by the C3PO. A C3PO can also be registered to run when an existing menu item or toolbar button is selected, thus allowing the C3PO to enhance or replace the normal functionality. In addition, you can trigger a C3PO to run when GroupWise starts, when GroupWise ends, or when a new item arrives in the mailbox.

Each object in the GroupWise data store (mail message, appointment, task) is theoretically a C3PO. By installing a C3PO against the default message class names, customized C3PO behavior can be invoked. For example, arrival of all items in the GroupWise data store can be monitored by installing a C3PO under the root item message class. Multiple handlers for a given message class are allowed. However, the order of execution is dependent on the order of handler installation in the Windows registry.

This section covers the following topics:

C3PO Functionality and the Object API

A C3PO accesses and manipulates the GroupWise data store through the GroupWise Object API. The combination provides a powerful framework to open the GroupWise system to third-party development.

C3PO classes are expressed in the GroupWise Object API as the MessageClass property on the message object and its derivatives. Every object in the GroupWise data store has a message class field, which is a string. The object class ties an instance of a C3PO back to the associated object handler hierarchy.

Empty Objects

A C3PO does not need to generate any data records in the GroupWise data store. This allows the C3PO to fully participate in menu modifications, toolbar negotiations, chaining of command extensions, and so forth, without requiring the overhead of managing instance data. Hence, a single set of extensions (the C3PO specification) can be used for client extensions. Although it is legal to register C3PO implementations that do not generate instance records under other headings, they are typically registered under GW.CLIENT.

C3PO Interfaces

A C3PO is a COM Server that implements the appropriate interfaces to achieve a desired purpose. The C3PO specification defines the prototypes of functions and methods. The C3PO writer must write the code to implement these methods. This is distinct from other APIs, which supply a library of functions that may be called by the user.

The C3PO specification defines the interfaces that a C3PO can choose to support. A C3PO is required to support only those interfaces that are necessary to implement the desired behavior. The GroupWise client determines which interfaces a C3PO supports by calling the Query interface. Minimizing the set of supported interfaces can improve performance. Because excessive processing while adding items to the GroupWise menus can hinder the perceived performance, the C3PO writer should take care not to disturb system processing beyond reasonable limits.

The C3PO interfaces are applicable to remote applications. However, the menu and toolbar modifications cannot be made from a remote application unless the C3PO writer performs custom marshalling using the raw menu and toolbar handles which will be exposed.

Inheritance

C3PO software supports the notion of inheritance. That is, if class Msg.Foobar exists (along with the requisite object handler), the message class Msg.Foobar.Bif represents specialized behavior of Msg.Foobar. For example, when an action is attempted against a database record of type Msg.Foobar.Bif, the associated handler is passed contextual information (COM interfaces) that allow

it to delegate behavior to ancestor classes. Individual C3POServers are responsible for delegation semantics. The C3POManager code (C3POManager) allows the C3POServer to build various enumerators for proper aggregation and delegation semantics.

The inheritance scheme is substantially (though not fully) realized. Each C3PO component can navigate the tree of data types and use the C3PO facilities. This is achieved through use of the C3POManager interfaces coupled with appropriate aggregation and delegation.

C3PO Instance Management

Each object in the GroupWise data store has a message class field, which is a string. The Object Class ties a C3PO instance to the associated object handler hierarchy. The C3PO system supports the multi-server concept, which allows multiple C3POServers to register under a single C3PO name. However, certain C3PO operations can only operate on a single server, in which case the first server registered under the C3PO name is invoked.

Languages

This optional key represents a series of language designations. The languages are precisely those supported by the component, which implies that multilingual support is accomplished by mapping from a single originating component into multiple languages. Presence of this section implies that the component can operate only in the target languages. Because Language Context ID (LCID) of 0 means any language, the restriction can be released by adding this key.

Each language is specified by a subkey, which is the LCID that is supported.

Thread Storage Architecture

When you are loading multiple C3POs as in-process servers (DLLs, rather than EXEs), you should be aware of an architectural limitation that is imposed by the operating system on the number of Thread Local Storage (TLS) indexes that are available. Since both the GroupWise Client and C3POs draw upon the same TLC pool and the GroupWise Client already uses a large number of these indexes (depending on what is being activated in the client), the TLS index limit might be exceeded when too many C3POs are added to the client.

The exact number of TLS indexes that are required by a C3PO can vary widely, depending on the language that is used to create the C3PO, what operations the C3PO performs, and other factors.

The following are TLS index limits for various operating systems:

Operating System	TLS Index Limit
Windows 95 and Windows NT 4.0	64
Windows 98 and Windows ME	80
Windows 2000 and Windows XP	1088

Because of the larger limit, TLS indexes are never a problem for Windows 2000 or Windows XP. Note also that the GroupWise 6.0 client uses slightly less indexes than GroupWise 5.5.EP.

If the TLS limit is exceeded, you might see a number of different runtime error messages.

C3PO Registration

Each C3PO must appear in the Windows system registry under a series of names that identify the C3PO to the GroupWise environment. Information stored in the registry includes ProgID identifications for C3POServers, hints about how the C3PO is to be used, and identification information. This information registers the C3PO with the GroupWise environment.

The registration naming system identifies a type relationship to GroupWise and is used to support inheritance of behavior. Names in the system are of the form GW.Classtype.Subtype. Classtype identifies the category of C3PO being registered (see GW.CLIENT). Subtype is used to relate one C3PO to another for purposes of supporting inheritance of behavioral semantics.

As each C3PO is registered with GroupWise, an association between the C3POServer and a particular type name is made. It is possible to register more than one C3PO at a specific naming point, but be careful when doing so because many operations have meaning only when bound to a single C3PO instance. To guarantee correct behavior, GroupWise will bind to the first registered C3PO in those instances. It is generally preferable to use subtypes or COM TreatAs mechanisms rather than registering more than one C3PO at a specific naming point.

The C3PO naming system has some restrictions. Naming is accomplished by appending a period (.) to an existing type and including an extension, which indicates a subtype. There are fixed root names under which names can be registered. Only one of these root names can be extended by subtyping. The root names available for registration are GW.MESSAGE and GW.CLIENT.

This section covers the following topics:

GW.MESSAGE

C3PO software registered under this root name are intended to implement behavior of specific record types. That is, the C3PO intends to create, display, send, or manipulate individual records.

Each item must be a subtype of one of the following:

GW.MESSAGE.MAIL
GW.MESSAGE.APPOINTMENT
GW.MESSAGE.TASK
GW.MESSAGE.NOTE
GW.MESSAGE.PHONE
GW.MESSAGE.DOCREF

In addition, the following are two subtypes of GW.MESSAGE.MAIL that already exist in GroupWise:

GW.MESSAGE.MAIL.Internet
GW.MESSAGE.MAIL.NGW.DISCUSS

GW.CLIENT

This root name indicates that the C3PO wants to implement some type of global behavior. Typically this is global menu modification, startup processing, or processing of delivery for all object types. The following subtypes are allowed:

Subtype	Description
GW.CLIENT.WINDOW.ATTACHVIEWER	Implements behavior local to the attachment viewer.
GW.CLIENT.WINDOW.BROWSER	Implements behavior local to the browser window.
GW.CLIENT.WINDOW.CALENDAR	Implements behavior local to calendar windows.
GW.CLIENT.WINDOW.DOCUMENTLIST	Implements behavior local to a document version list window.
GW.CLIENT.WINDOW.FINDRESULTS	Implements behavior local to a query results window.
GW.CLIENT.WINDOW.PROPERTIES	Implements behavior local to a properties window.
GW.CLIENT.WINDOW.QUICKVIEWER	Implements behavior local to the quick viewer window.
GW.CLIENT.WINDOW.ATTACHEMENT CONTROL	Implements behavior when you right click in an attachment window (in GroupWise 5.5.EP and GroupWise 6.0).
GW.CLIENT.WINDOW.ATTACHMENT CONTROL.EDITABLEFILE	Implements behavior when you right click in an outgoing attachment to a message (in GroupWise 5.5.EP and GroupWise 6.0).
GW.CLIENT.WINDOW.ATTACHMENT CONTROL.STATICFILE	Implements behavior when you right click on an incoming attachment to a message (in GroupWise 5.5.EP and GroupWise 6.0).

For example, to register a context menu item for the calendar, the C3PO must be registered under GW.CLIENT.WINDOW.CALENDAR.

Registry Definition

GroupWise refers to these registry entries to invoke the C3PO handlers. All registry entries appear as subkeys of the following entry:

```
HKEY_LOCAL_MACHINE\Software\Novell\GroupWise\5.0\C3PO\DataTypes\
```

Beneath this key appear entries for supported C3PO contexts, such as:

```
...\GW.MESSAGE.MAIL ...\GW.MESSAGE.MAIL.X ...\GW.CLIENT.WINDOW
```

Each registry section for a specific context contains subkeys used for naming the object server. The subkeys at this level are the PROGIDs of the COM servers for the C3PO. The value of each subkey is a description of the C3POServer, such as:

```
...\GW.MESSAGE.MAIL.X\Vendor.ObjName.Version = "My Description"
...\GW.MESSAGE.MAIL.X\Vendor2.ObjName2.Version = "Your Description"
```

These examples indicate that Vendor.ObjName.Version is the PROGID of the COM server supporting the C3PO. Beneath each PROGID key are further subkeys that clarify the use of the C3POServer. The subkeys are Objects and Events.

Objects

This subkey is required. It identifies the objects that are supported by the C3PO. This is an optimization to allow loading of the C3PO to be deferred. This is also a multi-valued key. Each value names a particular object that is supported by the C3POServer, such as:

```
\ProgID\Objects = "CommandFactory" = "IconFactory"
```

Events

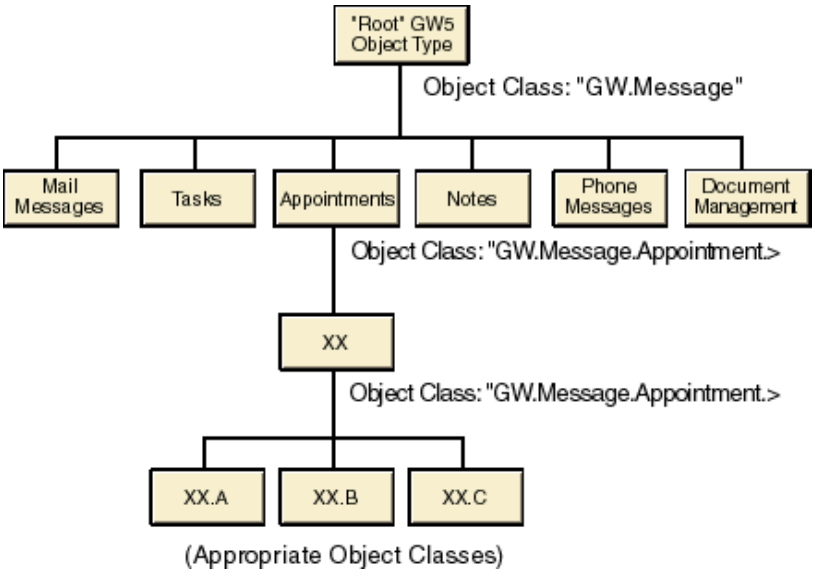
This subkey identifies the events for which that the C3PO is interested in receiving notification messages. This is a multi-valued key. Each value names a particular event, such as:

```
\ProgID\Events = "OnDelivery" = "OnReady" = "OnShutdown" = "OnOverflow" =
<Persistent Command ID>
```

Record Type Taxonomy

The GroupWise environment (both client and data-access components) binds to the registered C3PO providers as the support mechanism for custom data types. The data types are implemented via the data store in GroupWise.

Figure 1-1 *C3PO Object Class Hierarchy*



As illustrated, each C3PO data type is associated with a specific object class designation. These object classes are hierarchical in nature, representing a subtyping relationship. All behavior of an object, whether predefined, subclassed, or a newly instantiated type, is fully inheritable. For example, new methods introduced for new objects in the data store are fully inherited by any subtyped object.

Aggregations, delegation, and use of ancestor methods for implementation are fully supported. This is achieved by each C3PO writer having access to the instances of the C3POServers for supertypes. The C3PO writer is thereby free to invoke ancestor methods for any reason, such as delegation, aggregation, and behavior modification.

2 Tasks

This section contains instructions on how to write a GroupWise Custom 3rd-Party Object (C3PO), hints for developers, and the C3PO Customer Tracking Application sample.

How to Write a C3PO

GroupWise Custom 3rd-Party Object (C3PO) allows you to alter the GroupWise client user interface. You can add buttons to the toolbar, add menus and menu items to a menu, and create custom functionality for predefined client commands, such as Open or Compose. There are over 20 predefined GroupWise commands whose functionality you can either replace or enhance.

This section provides generic instructions on how to write C3PO. You can also use the "Tools" on page 41 section to create your C3PO.

Message Types

GroupWise has six message types. Each message type is defined as GW.MESSAGE. The types are:

GW.MESSAGE.APPOINTMENT
GW.MESSAGE.DOCUMENTREFERENCE
GW.MESSAGE.MAIL
GW.MESSAGE.NOTE
GW.MESSAGE.PHONE
GW.MESSAGE.TASK

In addition, GroupWise defines the following two subtypes of the GW.MESSAGE.MAIL context:

GW.MESSAGE.MAIL.Internet
GW.MESSAGE.MAIL.NGW.DISCUSS

Your C3PO can subclass any of the GroupWise message types to create a new custom class of your own. To do this, you pick one of the existing classes that fits the class you wish to create, then you subclass that class. For example:

GW.MESSAGE.NOTE.MYMESSAGE

You can use any name in place of MYMESSAGE. A new message of this type will have all the properties of a NOTE plus any others you give it.

Your C3PO can associate a custom icon with your custom class so that the user can distinguish between your custom message and others. Your C3PO can tell the C3PO Manager that you wish to be notified of certain events that occur. You can be notified of startup (eGW_CMDEVTID_READY), shutdown (eGW_CMDEVTID_SHUTDOWN), delivery of a certain message (eGW_CMDEVTID_DELIVERY), or overflow (eGW_CMDEVTID_OVERFLOW) conditions.

To summarize, with a C3PO you can create a custom message class of GW.MESSAGE.NOTE.MYMESSAGE and take control of the Open function of that class. Then, any time a user opens a message of your class, your C3PO will handle the Open.

You can control as much or as little of the manipulation of your custom class as you wish. You can add a menu item to File > New in the browser that, when selected, calls your C3PO to create a new message of your custom class type. Your C3PO can be seamlessly integrated into the GroupWise client user interface.

OLE COM Server

A C3PO is an OLE COM server. You can write a C3PO using any language that supports OLE or COM. But because a C3PO is a server, it makes writing a C3PO a little different than using most other APIs. You must create objects that the GroupWise C3PO Manager is looking for and that perform functions required by the manager.

The first C3PO OLE COM server object is C3POServer. It is the only object that is required.

In Visual Basic you create a C3POServer class with the following properties and methods:

```
Public Property Get CommandFactory( ) As CommandFactory
Public Property Get Description( ) As String
Public Property Get EventMonitor( ) As EventMonitor
Public Property Get IconFactory( ) As IconFactory
Public Function CanShutdown( ) As Boolean
Public Sub DeInit( )
Public Sub Init(objGWManager As Object)
```

In Delphi you define a class and then instantiate it. The class would look like the following:

```
C3POServer = class(TAutoObject)
     private
          { Private declarations }
          function GetCmdFact: Variant;
          function GetDescription : string;
          function GetEventMonitor : variant;
          function GetIconFactory : variant;
     automated
          { Automated declarations }
          property CommandFactory : Variant read GetCmdFact;
          property Description: string read GetDescription;
          property EventMonitor: variant read GetEventMonitor;
          property IconFactory: variant read GetIconFactory;
          function CanShutdown: TOleBool;
          procedure DeInit;
          procedure Init(Manager: variant);
     end;
```

Make sure that you create the routines associated with the new class.

Properties and Methods

The following are C3PO methods:

Method	Description
Init	This is called first by the C3PO Manager. The main purpose is to pass in the Manager object. One of the properties of the Manager object is the ClientState object, which object is used to find out the current state of the GroupWise client.
CanShutdown	This is called when the GroupWise client needs to shut down. You need to return a TRUE or FALSE. You are telling the Manager if it is all right to shut down. If you return a TRUE value, the Client will proceed with the shutdown. If you return a FALSE value, the Manager will poll you until you return a TRUE value.
DeInit	This is called to allow the C3PO to release any holds that still exist on any objects. DeInit terminates the relationship of the C3PO Manager with the C3PO.

The following are C3PO properties:

Property	Description
Description	This contains a short description of the C3PO.
CommandFactory	This contains the CommandFactory object. If this property is NULL, the Manager assumes that the C3PO does not want to have any CommandFactory functionality. CommandFactory is the object that allows you to add menus and menu items, add buttons to the toolbar, or take over predefined GroupWise commands.
EventMonitor	This contains the EventMonitor object. If this property is NULL, the Manager assumes that the C3PO does not want to have any EventMonitor functionality. EventMonitor allows the C3PO to handle events such as Ready, Shutdown, Delivery, and Overflow.
IconFactory	This contains the IconFactory object. If this property is NULL, the Manager assumes that the C3PO does not want to have any IconFactory functionality. IconFactory allows the C3PO to associate icons with custom message classes.

Custom Icons

Suppose you want to have a custom icon associated with your custom message class. You would create an IconFactory object and return it to the Manager when it calls your Get IconFactory function. The only method in the GetIcons object is the GetIcons method. In Visual Basic it would look like the following:

```
Public Sub GetIcons(sGWObjClass As String,
                    psGWIconFile As String,
                    plGWUnOpenIcon As Long,
                    plGWOpenIcon As Long)
psGWIconFile = "icons.dll"    ' set the icon file name
plGWUnOpenIcon = 1            ' set the unopen icon index
plGWOpenIcon = 0                ' set the open icon index
```

The variable passed in sGWObjClass is a string that is the class of the message that the client is going to paint. You must do the following:

1. Check to make sure it is the class you want to have your new icon associated with.
2. Pass back to the Manager the full path name of the .EXE or .DLL that contains the icon information for your custom icon.
3. Define 16x16 and 32x32 icons.
4. Return the index of the icon you want associated with both opened and unopened messages.

You also need to register your C3PO so that the Manager knows you want to have a custom icon associated with a custom message class. You need to register under:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Novell\GroupWise\5.0\C3PO\DataTypes\
GW.MESSAGE.MAIL.NEWCLASS\MYC3PO\Objects]
```

GW.MESSAGE.MAIL.NEWCLASS is the name of the new class that you want to have your icon associated with. MYC3PO is the name of your C3PO. The string value you need for this key is IconFactory.

C3PO Events

To set up your C3PO to handle GroupWise events, create an EventMonitor object and pass it back to the Manager in your GetEventMonitor function. EventMonitor has only one method: Notify. In Visual Basic it would look like the following:

```
Public Sub Notify(sGWContext As String, objGWEvent As Object)
Dim res
     Select Case objGWEvent.PersistentID
          Case eGW_CMDEVTID_READY
               'Check for Ready Event
               'This is were you put Ready code.
          res = MsgBox(objGWEvent.PersistentID, vbOKOnly, sGWContext)
          Case eGW_CMDEVTID_SHUTDOWN
               'Check for Shutdown Event
               'This is were you put Shutdown code.
          res = MsgBox(objGWEvent.PersistentID, vbOKOnly, sGWContext)
          Case eGW_CMDEVTID_OVERFLOW
               'Check for Overflow Event
               'This is were you put Overflow code.
          res = MsgBox(objGWEvent.PersistentID, vbOKOnly, sGWContext)
          Case eGW_CMDEVTID_DELIVERY
               'Check for Delivery Event
               'This is were you put Delivery code.
               If sGWContext = "GW.MESSAGE.MAIL.XXXX" Then
                    'Check for correct context
               res = MsgBox(objGWEvent.PersistentID, vbOKOnly,
                                  sGWContext)
               End If
          Case Else
               MsgBox "Unsupported Case"
          End Select
     End Sub
```

This Notify routine handles all four possible C3PO events: Ready, Shutdown, Overflow, and Delivery.

Notice in the preceding Visual Basic example that Notify passes in sGWContext, which is the class of the message that was delivered. Notify also passes in the objGWEvent object, which has a PersistentID property that tells you which events your C3PO is being called for, so it can handle it. For delivery, the context is also checked to make sure it is the desired message class.

Adding Menus, Menu Items, Buttons and Predefined Commands

The CommandFactory object handles these functions. You must create a CommandFactory object and return it in your GetCommandFactory function in the C3POServer object. The CommandFactory object has six methods you need to support. They are:

```
Public Function Init(lGWLCID As Long) As Long

Public Function WantCommand(sGWContext As String,
                            sGWPersistentID As String)
                            As Boolean

Public Function BuildCommand(sGWContext As String,
                             sGWPersistentID As String,
                             objGWBaseCommand As Object,
                             objGWParameter As Object)
                             As Object

Public Function CustomizeMenu(sGWContext As String,
                              objGWMenu As Object)
                              As Boolean

Public Sub CustomizeContextMenu(sGWContext As String,
                                objGWMenu As Object)

Public Function CustomizeToolbar(sGWContext As String,
                                 objGWToolbar As Object)
                                 As Boolean
```

Init

The Init routine is called first. It is an optimization to help speed up your C3PO. You pass back a set of flags that tell the Manager what to do. To only change menus, do the following:

```
Init = eGW_CMDINIT_MENUS
```

To add menus and buttons, use the following:

```
Init = eGW_CMDINIT_MENUS + eGW_CMDINIT_TOOLBARS
```

The possible flags are:

eGW_CMDINIT_MENUS
eGW_CMDINIT_TOOLBARS
eGW_CMDINIT_CONTEXT_MENUS
eGW_CMDINIT_NO_PREDEFINED

WantCommand, BuildCommand, and GWCommand

These methods work together. If you have registered your C3PO to handle one of the GroupWise pre-built commands your WantCommand method will be called. For a complete list of the possible pre-built commands, see C3PO Data Type Related Identifiers.

WantCommand

If you create a new custom message class of NEWCLASS that is sub-classed from the MAIL message class, then when a user opens any message of this class type, you will want to handle the open function. If you name your C3PO as MYC3PO, you would register your C3PO in the following manner:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Novell\GroupWise\5.0\C3PO\DataTypes\
GW.MESSAGE.MAIL.NEWCLASS\MYC3PO\Objects]
```

Under the Objects key you would have a string value of CommandFactory.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Novell\GroupWise\5.0\C3PO\DataTypes\
GW.MESSAGE.MAIL.NEWCLASS\MYC3PO\Events]
```

Under the Events key you would have a string value of GW#C#OPEN.

With your system registry set up like this, when a user opens a message of class GW.MESSAGE.MAIL.NEWCLASS, your WantCommand will be called. The WantCommand passes in two variables:

sGWContext	The class of the message.
sGWPersistentID	The type of predefined command that the user is attempting to perform.

A check is made to ensure that sGWContext contains the message class you are interested in and that sGWPeristentID is eGW_CMD_OPEN. If both of these conditions are TRUE, a TRUE value is returned for the method. By doing this, the Manager is informed that you are going to take over the Open function for this message.

BuildCommand

The BuildCommand method is called next. Again, sGWContext and sGWPeristentID are checked to ensure that the correct message and command are being used. You then need to build a GWCommand object and return it to the Manager. The code to build a GWCommand in the BuildCommand method would look like the following:

```
Dim GWCmd As New GWCommand
     ' Check for correct context before creating GWCommand object
     If sGWContext = COMMANDCONTEXT0 Then
    ' Check for the correct persistent ID to create GWCommand object
          If sGWPersistentID = eGW_CMD_OPEN Then
                    ' Set persistent ID for GWCommand object
               Let GWCmd.PersistentID = 1
                    ' Save base GWCommand for later use
               Set GWCmd.BaseCmd = objGWBaseCommand
               Set BuildCommand = GWCmd
                    ' Return GWCommand created
          End If
     End If
```

GWCommand

The GWCommand object is the object that the Manager uses to perform a GroupWise command. It looks like the following:

```
Public Property Get BaseCmd( ) As Object
Public Property Set BaseCmd(objNewBaseCmd As Object)
Public Property Get LongPrompt( ) As String
Public Property Let LongPrompt(sNewLongPrompt As String)
Public Property Get Parameters( ) As Object
Public Property Get PersistentID( ) As String
Public Property Let PersistentID(sNewPersistentID As String)
Public Property Get ToolTip( ) As String
Public Property Let ToolTip(sNewToolTip As String)
Public Sub Execute( )
Public Sub Help( )
Public Sub UnDo( )
Public Function Validate( ) As Long
```

BaseCmd: A property containing the GWCommand for client functionality.

LongPrompt: The string that will be displayed in the client when the user puts the cursor on a menu item.

ToolTip: The string that will be displayed in the client when the user puts the cursor on the toolbar button.

PersistentID: The ID of the GWCommand.

CustomizeMenu

The CustomizeMenu method allows your C3PO to add menus and menu items to the GroupWise client menus. If in the Init method you have set the bit telling the Manager that you want to change menus, this routine will be called any time the menu that you have registered to change is built.

You first need to decide were you want to place the menu. There are several options:

Option	Description
GW.CLIENT	All client views.
GW.CLIENT.WINDOW.ATTACHVIEWER	Attachment viewer.
GW.CLIENT.WINDOW.BROWSER	Browser window.
GW.CLIENT.WINDOW.CALENDAR	All calendar views.
GW.CLIENT.WINDOW.DOCUMENTLIST	Document list window.
GW.CLIENT.WINDOW.FINDRESULTS	Query results window.
GW.CLIENT.WINDOWS.PROPERTIES	Properties window.
GW.MESSAGE	All message types and their windows.
GW.MESSAGE.APPOINTMENT[.xx]	Appointments and their windows.
GW.MESSAGE.DOCUMENTREFERENCE	Document references.
GW.MESSAGE.MAIL[.xx]	Mail messages and their windows.
GW.MESSAGE.NOTE[.xx]	Notes and their windows.
GW.MESSAGE.PHONE[.xx]	Phone messages and their windows.

Option	Description
GW.MESSAGE.TASK[.xx]	Tasks and their windows.

If you want to place a new client menu item under the File > New menu, you would register it in the following manner:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Novell\GroupWise\5.0\C3PO\DataTypes\
GW.CLIENT.WINdOW.BROWSER\MYC3PO\Objects]
```

The string under the Objects key would be CommandFactory. Registered this way, your CustomizeMenu routine will be called. The Manager passes in to the CustomizeMenu routine the context that is being called for, and you need to check to be sure it is the right context. Also, the Manager passes in the GroupWise main menu object. That object is taken and used to find the menu you want to add to, and then the new menu item is added. The following is how to do this in Visual Basic:

```
Public Function CustomizeMenu(sGWContext As String, objGWMenu As Object)
            As Boolean
     Dim Menu As Object
     If sGWContext = "GW.CLIENT.WINDOW.BROWSER" Then
               ' Check for correct context
          Set Menu = objGWMenu
               ' Get Main menu object
          Set Menu = Menu.MenuItems.Item("File")
                ' get menu File
          Dim Cmd01 As New GWCommand
               ' Build GWCommand object
          Let Cmd01.PersistentID = XXXX
               ' Set persistent ID for Custom menu in GWCommand object
          Let Cmd01.LongPrompt = "Create a new message of type
                                          GW.MESSAGE.MAIL.XXXX"
               ' set long prompt for menu item
          Call Menu.MenuItems.Add("Create New XXXX Message", Cmd01)
               ' add menu item to the end of menu
     End If
End Function
```

Notice that once again a GWCommand object is built. Set the Persistent ID and the Long prompt for the new menu item. When a user selects your new menu item, the Manager calls your GWCommand Execute method. You then should check the Persistent ID to see that it is this menu item being clicked. Then you can continue with whatever you want your C3PO to do.

You must provide the Execute method. This method is called by the Manager when a button is clicked, a menu item is selected or a predefined command is executed.

You must provide the Validate method. This method is called to ask the C3PO if the GWCommand is valid under the current conditions. Validate can return:

eGW_CMDVAL_CHECKED	The command has a check mark.
eGW_CMDVAL_DISABLED	The command is disabled.

CustomizeContextMenu

The CustomizeContextMenu method allows your C3PO to add menus and menu items to the GroupWise client context menus. If in the Init method you have set the bit telling the Manager that you want to change context menus, this routine will be called anytime the menu that you have registered to change is built.

You first need to decide were you want to place the menu. There are several options to choose from that are similar to the CustomizeMenu method, except that the GW.CLIENT.WINDOW.PROPERTIES and the GW.CLIENT.WINDOW.ATTACHVIEWER contexts are not available. Also, the following three additional contexts have been added for GroupWise 5.5.EP and GroupWise 6.x:

GW.CLIENT.WINDOW.ATTACHMENTCONTROL is the attachment window.

GW.CLIENT.WINDOW.ATTACHMENTCONTROL.STATICFILE is the incoming attachment in the attachment window.

GW.CLIENT.WINDOW.ATTACHMENTCONTROL.EDITABLEFILE is the outgoing attachment in the attachment window.

Another difference is that CustomizeContextMenu passes in the context menu object instead of the menu object itself.

The execution and validation functions for customizing context menus operate in the same manner as the functions for customizing regular menus, and the registration information is the same.

CustomizeToolbar

To add a button to the toolbar you need to add code to the CustomizeToolbar method. If in the Init method you have set the bit telling the Manager that you want to change the toolbar and you are registered correctly, your CustomizeToolbar method will be called each time that toolbar is built.

You need to decide to which toolbar you want to add a toolbar button. The options are the same as for the CustomizeMenu method, with the addition of the GW.CLIENT.WINDOW.QUICKVIEWER context. The registration for adding a button is also the same. Do the following in Visual Basic:

```
Public Function CustomizeToolbar(sGWContext As String, objGWToolbar As
Object)
          As Boolean
     Dim Button As Object
     Dim FilePath As String
     If sGWContext = "GW.CLIENT.WINDOW.BROWSER" Then
               ' Check for correct context
          Dim Cmd00 As New GWCommand
               ' Build GWCommand object
          Let Cmd00.PersistentID = XXXX
               ' Set persistent ID for GWCommand object
          Let Cmd00.ToolTip = "Create a new message of type
                              GW.MESSAGE.MAIL.XXXX"
               ' Set Button tooltip
          Set Button = objGWToolbar.ToolbarItems.Add("New XXXX Message",
```

```
Cmd00)
               ' Add button to toolbar
          FilePath = App.Path & "\icons.dll"
               ' Set bitmap for Button
               ' icons.dll can be replaced by the full path name of any
               '  .exe or  .dll that contains a 16x16 and a 32x32 pixel
               '  bitmap
               ' BUTTON_1 can be replaced with the name of the bitmap
               '  contained in the .exe or .dll
          Call Button.SetBitmap(FilePath, "BUTTON_1")
               ' set were the bitmap is found and its name
     End If
     CustomizeToolbar = False
End Function
```

In this routine, you check to see if it is the context that you are interested in. Then you create a new GWCommnad object. You set the Persistent ID, set the ToolTip, and the add the button. You then set the bitmap for the button.

Hints for the Developer

The following sections address various hints:

Modeless Dialog Boxes

Modeless dialog boxes are best implemented in an out-of-process server. Because GroupWise traps keyboard events (to turn them into accelerators and eventually into tokens), a .DLL server will experience problems receiving keyboard events when creating a modeless dialog box or window.

Identification of Menu Items, Submenus, and Separators

These items can often be identified in C++ by using the IUnknown pointers to ID the running instances. For example, when an item is added to the menu, the IUnknown pointer is saved and used later to identify the instance when calling GWMenuItems::Item( ).

Refreshing the Client User Interface

A C3PO that replaces custom commands often needs to refresh the client user interface. To do this, given the GWClientState interface, invoke the BuildPersistentID method using the syntax GW#C#xxx where xxx is a function, such as Open. Then call the BuildCommand method, passing the new ID to create a GWCommand. Finally, execute the GWCommand to refresh the Client UI. (See Pre-Built Command ID Encoding.)

Note that if you refresh a query folder, the query will re-execute. The constants in Pre-Built Command ID Encoding provide a text parameter that is associated with each numerical value.

C3PO Sample: the Customer Tracking Application

This section contains the following:

Creating the Customer-Tracking C3PO with Delphi

Start Delphi and create a new project. Rename the project to CusTrack. Rename unit1.pas to exesrv, change the form name to CTS, and set the form caption to Customer Tracking.

C3POServer

Add a new OLE object called C3POServer. This creates a new unit; rename it to servobj.pas. In addition, an object class named C3POServer will be created.

The C3POServer object is the first object that needs to be created for the C3PO. Change C3POServer so that it subclasses from TC3POServer instead of TAutoObject. Then add functions, procedures, and properties as shown in the following:

```
C3POServer = class(TC3POServer)
private
{ Private declarations }
function GetCmdFact: Variant;
function GetDescription : string;
automated
{ Automated declarations }
property CommandFactory : Variant read GetCmdFact;
property Description: string read GetDescription;
function CanShutdown: TOleBool;
procedure Init(Manager: variant);
end;
```

The C3POServer object is required and all properties must be set up. TC3POServer is defined in C3POin.pas, and is a complete C3POServer object with all C3PO objects and routines necessary for these objects. The C3POServer that you defined subclasses TC3POServer, and overrides only the routines that you need to use. TC3POServer will handle the others. This keeps your servobj.pas simple and easy to read.

Add C3POin.pas and ObjApiin.pas from the SDK to your project. These files are similar to C header files. They contain information you need for using C3PO software and the Object API. In servobj.pas, add C3POin, OLE2, and Windows to the uses directive as shown in the following:

```
uses
OLEAuto, OLE2, Windows, C3POin;
```

Add routines for the functions and procedures defined in the C3POServer object. This is shown in servobj.pas. Notice that C3POServer.Init( ) saves "manager" into a global variable called g_C3POManager, which is used later to get the ClientState.

C3POServer.GetCmdFact returns g_CommandFactory.OLEObject, which means you need to define a CommandFactory object. This is done in the sample under the vars directive. Make g_CommandFactory a global variable of type CommandFactory. Then, under initialization, give g_CommandFactory to CommandFactory.Create. This creates a CommandFactory object. Finally, under finalization, release g_CommandFactory to free up memory.

CommandFactory

A CommandFactory class must be defined. To do this, create a new unit called C3PO.pas, then add C3PO to the uses directive in servobj.pas. Next, create a CommandFactory class in C3PO.pas as shown in the following:

```
CommandFactory =
class(TCommandFactory)
private
ContextMenuID: integer;
public
public
Constructor Create;     // Used to create CommandFactory Object
automated
function CustomizeMenu( Context: string;
GWMenu: variant): TOleBool;
function Init(lcid : longint): longint;
end;
```

You are subclassing from TCommandFactory defined in C3POin.pas. You are going to change only the menus at this time, so all you need to define are the CustomizeMenu and Init functions.

Customize Menus

Create the CommandFactory.Init( ) function as in the sample. Return eGW_CMDINIT_MENUS to say that you are customizing the menus.

Create the CommandFactory.CustomizeMenu( ) function. This is the routine that will change the GroupWise client menus. See the sample.

Each menu item must have a GWCommand object. To do this, you need to define a Command class as in the following example:

```
Command = class(TGWCommand)
private
{ Private declarations }
LongPrmt : string;
ToolTp : string;
function GetLongPrompt : string;
function GetToolTip : string;
public
m_nCmd : longint;     // Command ID information
Constructor Create(nCmd: longint);     // Used to create Command Object
automated
property LongPrompt: string read
GetLongPrompt;
property ToolTip: string read
GetToolTip;
procedure Execute;
function Validate: longint;
end;
```

Create the functions and procedures as in the sample. As a command object for each menu item is created, a unique ID is stored in m_nCmd. This ID is used to determine which menu item has been chosen by the user when Execute is called.

Compile and register CusTrack.

Customize ContextMenus

CusTrack also customizes the context menu by overriding CommandFactory.CustomizeContextMenu( ) in CommandFactory. To do this, add the routine shown in the following example. Change CommandFactory.Init( ) to look like this example:

```
result := eGW_CMDINIT_MENUS or     // modify menus
eGW_CMDINIT_CONTEXT_MENUS;         // modify context menus
```

Customize Toolbars

CusTrack customizes the toolbar by overriding CommandFactory.CustomizeToolBar in CommandFactory. To do this, add the routine as shown in the following example. Change CommandFactory.Init to look like this example:

```
result := eGW_CMDINIT_MENUS or     // modify menus
eGW_CMDINIT_CONTEXT_MENUS or       // modify context menus
eGW_CMDINIT_TOOLBARS;              // modify toolbars
```

Predefined Commands

The CusTrack C3PO defines three new objects:

CTS_COMPANY_OBJ = ’GW.MESSAGE.MAIL.NGWCOMPANY’;

CTS_CONTACT_OBJ = ’GW.MESSAGE.MAIL.NGWCONTACT’;

CTS_ACTION_OBJ = ’GW.MESSAGE.MAIL.NGWACTION’;

These objects subclass from GW.MESSAGE.MAIL.

CusTrack supports the Open predefined command for these three objects. To do this, you need to set up the registry. You also need to override CommandFactory.WantCommand and CommandFactory.BuildCommand as shown in C3PO.pas. WantCommand checks context and the PersistentID to see if the predefined command that is being set up will be supported. A check is made to see if it is the Open command, and then the context is checked to make sure it is one of the new objects. TRUE is then returned to tell the C3POManager that we will handle the open of these new objects. In BuildCommand a check is again made for the PersistentID and Context for the proper objects. A GWCommand is then built to handle Open.

IconFactory

To show your own icons for your custom objects, you need to support IconFactory. Override the IconFactory property in C3POServer as shown in the following example:

```
C3POServer = class(TC3POServer)
private
{ Private declarations }
function GetCmdFact: Variant;
function GetDescription : string;
function GetIconFactory : variant;
automated
{ Automated declarations }
property CommandFactory : Variant read GetCmdFact;
property Description: string read
GetDescription;
property IconFactory: variant read GetIconFactory;
function CanShutdown: TOleBool; procedure Init(Manager: variant);
end;
```

Under the vars directive in servobj.pas, add the following:

```
g_IconFactory : IconFactory;     // Create global IconFactory object
```

Under the initialization directive in servobj.pas, add the following:

```
g_IconFactory := IconFactory.Create;
```

Under the finalization directive in servobj.pas, add the following:

```
g_IconFactory.Release;
```

In C3PO.pas, define IconFactory as shown below.

```
IconFactory = class(TIconFactory)
private
{ Private declarations }
public
automated
procedure GetIcons(
ObjClass: string;
var pIconFile: string;
var plUnOpenIcon: longint;
var plOpenIcon:longint);
end;
```

Create the GetIcons procedure as shown in the sample.

EventMonitor

CusTrack uses two events, OnReady and OnShutdown. To do this, override the EventMonitor property in C3POServer as shown in the following example:

```
C3POServer = class(TC3POServer)
private
{ Private declarations }
function GetCmdFact: Variant; function GetDescription : string;
function GetEventMonitor : variant;
function GetIconFactory : variant;
automated
{ Automated declarations }
property CommandFactory : Variant read GetCmdFact;
property Description: string read
GetDescription;
property EventMonitor: variant read GetEventMonitor;
property IconFactory: variant read GetIconFactory;
function CanShutdown: TOleBool;
procedure Init(Manager: variant);
end;
```

Under the vars directive in servobj.pas, add the following:

```
g_EventMonitor :EventMonitor;     // Create global EventMonitor object
```

Under the initialization directive in servobj.pas, add the following:

```
g_EventMonitor :=EventMonitor.Create;
```

Under the finalization directive in servobj.pas, add the following:

```
g_EventMonitor.Release;
```

In C3PO.pas, define EventMonitor as shown in the following:

```
EventMonitor =  class(TEventMonitor)
private
public
automated
procedure Notify(Context: string; evt: variant);
end;
```

Create the Notify procedure as shown in the sample.

Creating the Customer-Tracking C3PO with C++

The Customer tracking sample application was created as a in-process server (DLL) using Microsoft Visual C++ 4.0. A C3PO in C/C++ must use a COM interface. Begin by creating a DLL project called C3PO.

Add a new file to the project called C3PO.cpp. In C3PO.cpp add new routines DllGetClassObject, DllCanUnloadNow, and BuildIUnkDispatch as shown in the C3PO.cpp sample. Next, use GUIDGEN.EXE to define a new GUID for the COM server. Put the definition in C3PO.h. For example:

```
DEFINE_GUID(CLSID_SAMPLEC3PO,
            0xd49, 0ce00, 0x8bb, 0x11cf, 0xbb, 0xf3,
            0x0, 0x20, 0xaf, 0xe0, 0x28, 0x9c);
```

The next step is to build a Class Factory. In C3PO.h define MyFactory as shown in the following:

```
class MyFactory :
public IClassFactory
{
public:
/* IUnknown methods */
STDMETHOD(QueryInterface)(
THIS_ REFIID riid,
LPVOID FAR*ppvObj);
STDMETHOD_(ULONG, AddRef)(THIS);
STDMETHOD_(ULONG, Release)(THIS);
STDMETHODIMP CreateInstance(IUnknown *, REFIID, void**);
STDMETHODIMP LockServer(BOOL);
MyFactory( );
~MyFactory( );
private:
ULONG m_cRef;
```

Next, you need to build routines for the methods defined in MyFactory as shown in C3PO.cpp. In MyFactory::CreateInstance, build a C3POServer called CC3PO.

```
STDMETHODIMP MyFactory::CreateInstance(
IUnknown *pUnkOuter,
REFIID riid,
void** ppv)
{
if(NULL != pUnkOuter)
return CLASS_E_NOAGGREGATION;
CC3PO *pIC3PO = new CC3PO;
if(pIC3PO == NULL)
return E_OUTOFMEMORY;
pIC3PO->Create( );
HRESULT hr = pIC3PO->QueryInterface(riid, ppv);
if(FAILED(hr))
delete pIC3PO;
else
g_cObj++;
return hr;
}
```


“C3POServer” on page 34

“CommandFactory” on page 35

“CustomizeMenus” on page 36

“CustomizeContextMenu” on page 37

“CustomizeToolBars” on page 37

“Predefined Commands” on page 37

“IconFactory” on page 38

“EventMonitor” on page 39

C3POServer

You now need to define a C3POServer object. Every C3PO must support this interface. It is used to initialize the C3PO. Add a new Class called CC3PO in C3PO.h. Subclass it from IC3POServer. This is the object that was created in MyFactory::CreateInstance.

```
class CC3PO : public
IC3POServer
{
public:     // IUnknown methods
STDMETHOD(QueryInterface)(
     THIS_ REFIID riid,
     LPVOID FAR* ppvObj);
STDMETHOD_(ULONG, AddRef)(THIS);
STDMETHOD_(ULONG, Release)(THIS);
/* IC3POServer methods */
STDMETHOD(get_CommandFactory)(
     THIS_ IDispatch * FAR*ppIDispCommandFactory);
STDMETHOD(get_Description)(
     THIS_ BSTR FAR* pbstrDescription);
STDMETHOD(get_EventMonitor)(
     THIS_ IDispatch * FAR*ppIDispEventMonitor);
STDMETHOD(get_IconFactory)(
     THIS_ IDispatch * FAR*ppIDispIconFactory);
STDMETHOD(CanShutdown)(
     THIS_ VARIANT_BOOL FAR* pbCanShutdown);
STDMETHOD(DeInit)(THIS);
STDMETHOD(Init)(
     THIS_ IDispatch * pIDispManager);     // Constructor
CC3PO( );     // Destructor
virtual ~CC3PO( );     // RefCount required method
BOOL Create( );
private:
ULONG m_cRef;
CCommandFactory  *m_pICmdFact;
EventMonitor  *m_pIEventMonitor;
IconFactory  *m_pIIconFactory;
IUnknown *m_pIUnkDispServ;
};
```

All IC3POServer methods must be declared. If a method is not needed, it simply does a return. The Init( ) method is the first one called and passes in the C3POManager. The Manager object is saved to get the Client State and is valid until a future DeInit( ) call. If the server fails the call, the C3POServer is unloaded.

The Customer Tracking C3PO uses the functionality from the CommandFactory, EventMonitor and IconFactory methods. In Create( ) they are created and in QueryInterface( ) a check of the riid is made for these interfaces and the appropriate object is returned. In each get the interface for the object is returned. See C3PO.cpp.

CommandFactory

This interface is used to manage commands in GroupWise. Commands are located in menus and toolbars. Since menu, context menu, and the toolbar are to be modified in Customer Tracking , this interface must be supported. Begin by defining a new class in Setupcmd.h called CCommandFactory.

```
class CCommandFactory :
public ICommandFactory
{
public:
/* IUnknown methods */
STDMETHOD(QueryInterface)(
     THIS_ REFIID riid, LPVOID FAR*ppvObj);
STDMETHOD_(ULONG, AddRef)(THIS)
{ return m_pUnkOuter-&gt;AddRef( ); }
STDMETHOD_(ULONG, Release)(THIS)
{ return m_pUnkOuter-&gt;Release( ); }
/* ICommandFactory methods */
STDMETHOD(BuildCommand)(
     THIS_ BSTR bstrContext,
     BSTR bstrPersistentID,
     IDispatch *pIDispBaseCommand,
     IDispatch * pIDispParameters,
     IDispatch *FAR* ppIDispGWCommand);
STDMETHOD(CustomizeContextMenu)(
     THIS_ BSTR bstrContext,
     IDispatch * pIDispGWMenu);
STDMETHOD(CustomizeMenu)(
     THIS_ BSTR bstrContext,
     IDispatch *pIDispGWMenu,
     VARIANT_BOOL FAR* pbChanged);
STDMETHOD(CustomizeToolbar)(
     THIS_ BSTR bstrContext,
     IDispatch *pIDispGWToolbar,
     VARIANT_BOOL FAR* pbChanged
STDMETHOD(Init
     THIS_ long lcid,
     long FAR* plCmdFlags
STDMETHOD(WantCommand
     THIS_ BSTR bstrContext,
     BSTR bstrPersistentID,
VARIANT_BOOL FAR*pbChanged
CCommandFactory(IUnknown *pUnk
CCommandFactory
private
IUnknown   *m_pUnkOuter;
IUnknown *m_pIUnkDispFact;
};
```

All methods in CommandFactory must be supported. After QueryInterface( ), the first method to be called is the Init( ) routine.

```
STDMETHODIMP CCommandFactory::Init(long lcid,
                                   long FAR* plCmdFlags)
{
*plCmdFlags = eGW_CMDINIT_MENUS     // modify menus
eGW_CMDINIT_CONTEXT_MENUS |         // modify context menus
eGW_CMDINIT_TOOLBARS;               // modify toolbars
return NOERROR;
}
```

A flag is returned in *plCmdFlags indicating if modify menus, context menus and/or toolbars are to be modified. In this case, all of them will be modified.

CustomizeMenus

The CustomizeMenu method must be supported. If you do not want to customize a menu, simply return NOERROR. To support the method, you must first define a new class named CGWCommand in Setupcmd.h.

```
class CGWCommand :
public IGWCommand
{
public:
/* IUnknown methods */
 STDMETHOD(QueryInterface)(
THIS_ REFIID riid, LPVOID FAR* ppvObj);
STDMETHOD_(ULONG,AddRef) ( );
STDMETHOD_(ULONG,Release) ( );
/* IGWCommand methods */
STDMETHOD(get_BaseCmd)(
     THIS_ IDispatch * FAR* ppIDispBaseCmd);
STDMETHOD(get_LongPrompt)(
     THIS_ BSTR FAR* pbstrLongPrompt);
STDMETHOD(get_Parameters)(
     THIS_ IDispatch * FAR* ppIDispBaseCmd);
STDMETHOD(get_PersistentID)(
     THIS_ BSTR FAR* pbstrPersistentID);
STDMETHOD(get_ToolTip)(
     THIS_ BSTR FAR* pbstrToolTip);
STDMETHOD(Execute)(THIS);
STDMETHOD(Help)(THIS);
STDMETHOD(Undo)(THIS);
STDMETHOD(Validate)(
     THIS_ long FAR* plValidate);
CGWCommand(int nID);
~CGWCommand( );
BSTR bstrLongPrompt;
BSTR bstrToolTip;
private:
ULONG  m_cRef;
int   m_nID;
IUnknown *m_pIUnkDisp;
};
```

Once again, all methods must be supported. LongPrompt is only for menu items and Tooltip is only for toolbar items. One command is distinguished from another with the m_nID. As each command is created it is given a unique ID, so that you can tell each command apart. Validate returns whether the command is enabled, disabled, or checked for menu items and toolbar items. Execute is called when the item has been selected by the user. See Setupcmd.cpp.

With GWCommand set up, you can now build or modify a menu item. CCommandFactory::CustomizeMenu( ) adds another menu to the New menu of the File menu. In the new menu, called Customer Tracking, three new menu items are included. See CCommandFactory::CustomizeMenu( ) in Setupcmd.cpp.

In CGWCommand::Validate( ), a check is made to see if any customer tracking message has been selected by the user. If no customer tracking message has been selected, the contact and action menu items are disabled and company menu item is enabled. If the user has selected a company message, then the company and contact menu items are enabled. If the user has selected a contact message, then the company and action menu items are enabled.

CustomizeContextMenu

For customer tracking, you want to add a new item to the context menu only when the user has selected a customer tracking message item, and then right-clicks to bring up the context menu. To do this the value of bstrContext is checked to see if it is a CTS_COMPANY_OBJ, CTS_CONTACT_OBJ or CTS_ACTION_OBJ message object. If the user has selected a company object that then a new menu item is set for creating a contact message. If the user has selected a contact message, then a new menu item is set for creating an action message. If the user has selected an action message, a new menu item is set for creating another action message. This is done in CCommandFactory::CustomizeContextMenu( ) in Setupcmd.cpp.

CustomizeToolBars

For customer tracking, three new buttons are set up on the toolbar, one each to create a new company, contact, and action message. CCommandFactory::CustomizeToolbar( ) makes the modifications to the toolbar. Since the same command IDs in CustomizeToolbar( ) are used as in CustomizeMenu( ), the buttons on the toolbar will act the same as the items on the custom menu that was created.

Predefined Commands

The CusTrack C3PO defines three new objects:

CTS_COMPANY_OBJ = ’GW.MESSAGE.MAIL.NGWCOMPANY’;
CTS_CONTACT_OBJ = ’GW.MESSAGE.MAIL.NGWCONTACT’;
CTS_ACTION_OBJ = ’GW.MESSAGE.MAIL.NGWACTION’;

These objects subclass from GW.MESSAGE.MAIL.

CusTrack supports the Open predefined command for these three objects. To do this, enter the registry entries shown in Cts.reg. After the C3PO is registered, CCommandFactory::WantCommand( ) will be called when the user attempts to open one of the custom objects. In WantCommand, check bstrPersistentID see if the Open command is being called. If so then *pbChanged = TRUE is returned.

If not, FALSE is returned. If TRUE is returned, then CCommandFactory::BuildCommand( ) is called. Here you once again check for the correct persistent ID, and if it is correct build a GWCOMMAND for this Open. See Setupcmd.cpp.

IconFactory

This interface is used to retrieve icons that represent the state of C3PO records. Use IconFactory to change the icons for custom objects.

```
class IconFactory : public
IIconFactory
{
public:
/* IUnknown methods */
STDMETHOD(QueryInterface)(
     THIS_ REFIID riid, LPVOID FAR*ppvObj);
STDMETHOD_(ULONG, AddRef)(THIS)
{ return m_pUnkOuter-&gt;AddRef( ); }
STDMETHOD_(ULONG, Release)(THIS)
{ return m_pUnkOuter-&gt;Release( ); }
/* IconFactory methods */
STDMETHOD(GetIcons)(
     THIS_ BSTR bstrObjClass,
     BSTR FAR*pbstrIconFile,
     long FAR* plUnOpenIcon,
     long FAR*plOpenIcon);
IconFactory(IUnknown *pUnk);
~IconFactory( );
private:
IUnknown *m_pUnkOuter;
IUnknown *m_pIUnkDispFact;
};
```

In GetIcons( ), bstrObjClass is checked to see which custom object is being called for, then the icon file and the icon index for both an opened and unopened state of the object are returned. See Setupcmd.cpp. In addition, IconFactory needs to be registered; an example is found in Cts.reg.

EventMonitor

Customer Tracking uses two events, OnReady and OnShutdown.

```
class EventMonitor :
public IEventMonitor
{
public:
/* IUnknown methods */
STDMETHOD(QueryInterface)(
     THIS_ REFIID riid, LPVOID FAR* ppvObj);
STDMETHOD_(ULONG, AddRef)(THIS)
{ return m_pUnkOuter->AddRef(  ); }
STDMETHOD_(ULONG, Release)(THIS)
{ return m_pUnkOuter->Release( ); }
/* IEventMonitor methods */
STDMETHOD(Notify)(
     THIS_ BSTR bstrContext,
     IDispatch *pIDispGWEvent);
EventMonitor(IUnknown *pUnk);
~EventMonitor( );
private:
IUnknown *m_pUnkOuter;\
IUnknown *m_pIUnkDispFact;
};
```

In the EventMonitor::Notify( ), bstrPersistentID is checked to see which event is calling and then it performs the desired function. See Setupcmd.cpp.

3 Tools

This section documents the GroupWise Custom 3rd-Party Object (C3PO) tools, such as the C3PO Wizard.

The C3PO Wizard provides a menu-driven interface that helps you create a C3PO framework for the Visual Basic, Delphi, and C++ development languages. The Wizard walks you through:

- Creating a C3PO name and defining a path.
- Determining how you want your C3PO to function.
- Provide additional information based on your function selections.

Once you have provided all the necessary information to create the C3PO, the Wizard will finish the creation process.

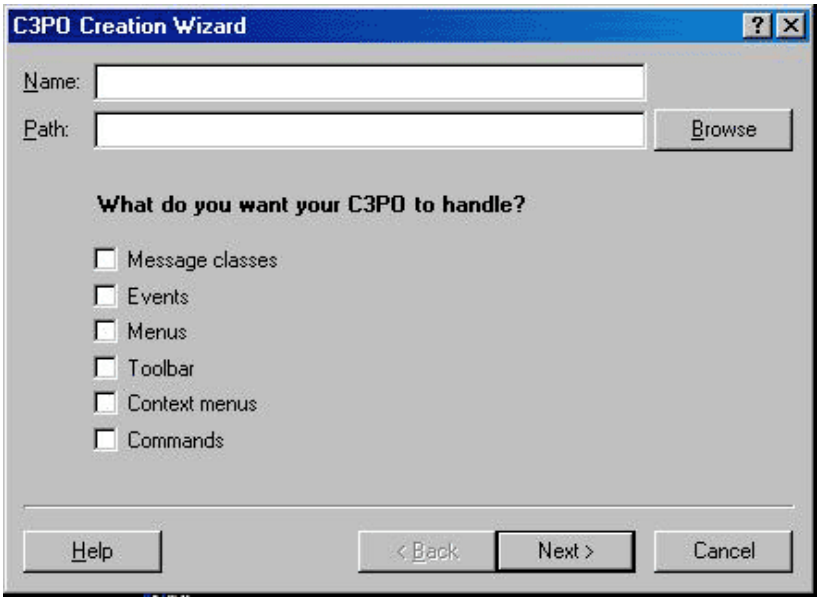
This guide consists of the following sections.

Starting the Wizard

1 Start the wizard by running `C3POWIZARD.EXE`.

After you have read the initial box which explains how to navigate through the wizard, click *Next* to display the following dialog box

Figure 3-1 *C3PO Creation Wizard Screenshot*



The following step describes the fields in this dialog box.

2 Fill in the fields:

Name: Specify the name of the .WIZ file you want to create.

Path: Specify the path to the directory where you want the .WIZ file created. Click *Path* to browse for the directory. The path must already exist.

Options: These options let you perform certain tasks with your C3PO. After selecting a check box, you will be prompted to provide additional information. For example, if you select the option to use GroupWise C3PO Events, you will be prompted to choose which events to use: Ready, Shutdown, Delivery, or Overflow.

- *I want to define a custom message class in GroupWise.* Check this box if you want your C3PO to define a customer message class that GroupWise will recognize.
- *I want to use GroupWise C3PO events.* C3POs allow you to trap four GroupWise events:
  - *Ready*—Occurs when the GroupWise client has painted the screen and the client dialog box is ready to receive user input.
  - *Shutdown*—Occurs when the user has exited the GroupWise client and the client is ready to shut down.
  - *Delivery*—Occurs when a GroupWise message of the specified Class type has been delivered to the In Box.
  - *Overflow*—Occurs when more messages are delivered to the In Box than the Delivery event can handle.
- *I want to customize the GroupWise menus.* Check this box if you want your C3PO to modify the GroupWise client menus.
- *I want to customize the GroupWise Toolbar.* Check this box if you want your C3PO to add buttons to the GroupWise client toolbar.

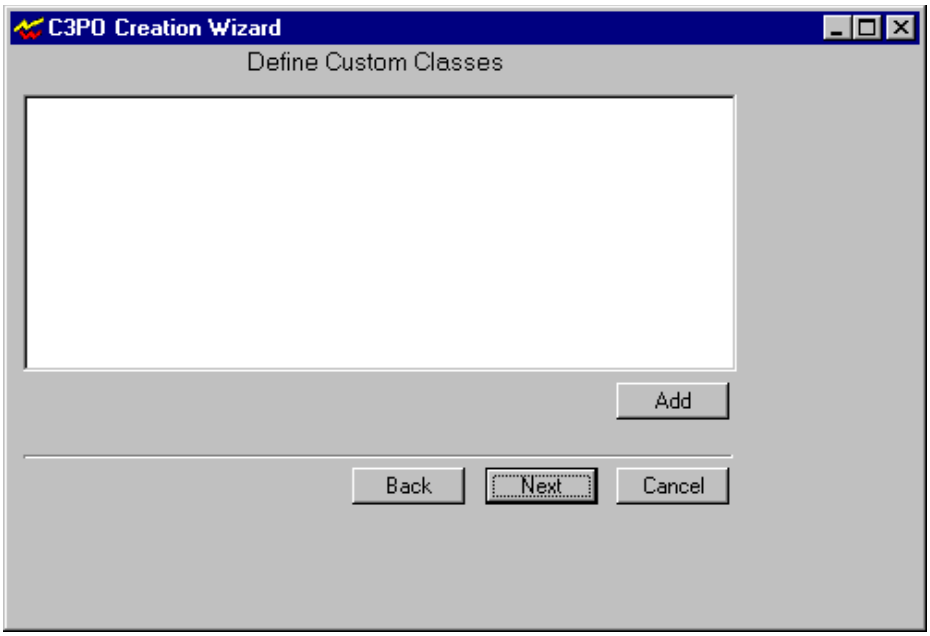
- *I want to customize the GroupWise context menus.* Check this box if you want your C3PO to modify the GroupWise client context menus.
- *I want to handle standard GroupWise C3PO commands.* Check this box if you want your C3PO to use certain predefined client functions, such as Open, Save, and Save As.

3 Click *Next*. For each option you selected, a dialog box will appear, prompting you to provide additional information about your selections.

Defining Custom Message Classes

If you selected *I want to define a custom message class in GroupWise*, the following dialog box is displayed.

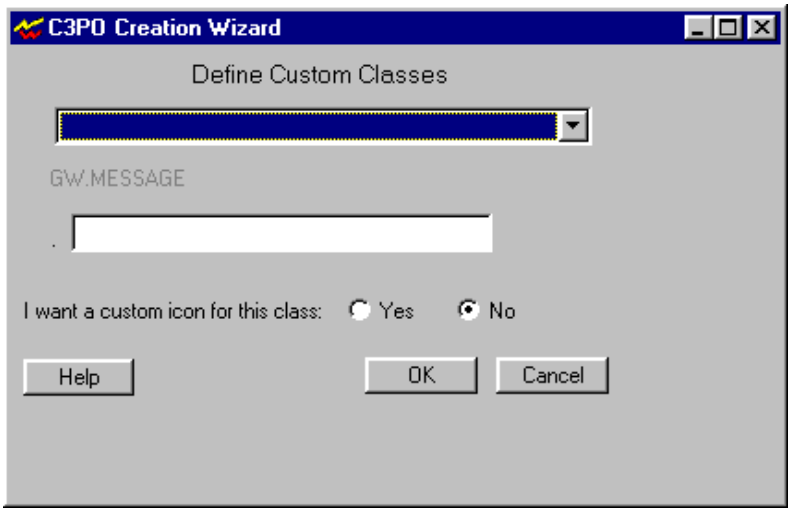
Figure 3-2 *C3PO Custom Messages Screenshot*



This dialog lists defined custom classes.

1 Click *Add* to display the following dialog box.

Figure 3-3 *C3PO Defining Custom Classes Screenshot*



2 Click the down arrow, then select one of the six base message classes. All custom classes must be subclasses of the six base classes:

GW.MESSAGE.APPOINTMENT
GW.MESSAGE.DOCREFERENCE
GW.MESSAGE.MAIL
GW.MESSAGE.NOTE
GW.MESSAGE.PHONE
GW.MESSAGE.TASK

3 Select the base class that best fits your custom class, then enter the name of your custom class in the text box.

4 If you want the wizard to set up an icon for your custom class, select *Yes*. Otherwise, select *No*.

5 Click *OK*.

6 Repeat Steps 2-5 for each class you want to add. When you are finished, click *Next*.

Using Events

If you selected *I want to use GroupWise events*, the following dialog box is displayed.

Figure 3-4 *Events Screenshot*

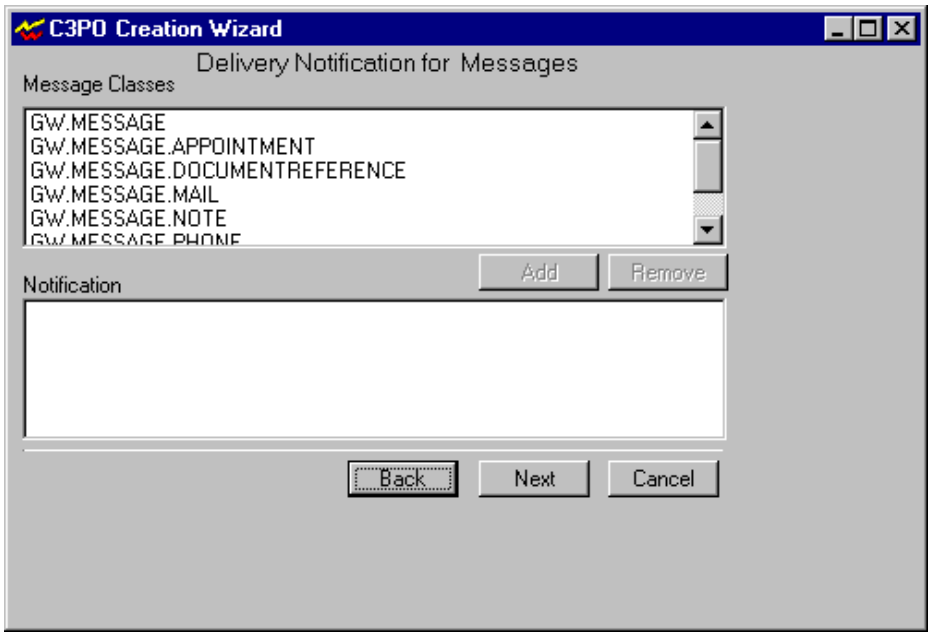
This dialog allows you to select from Ready, Shutdown, Delivery, and Overflow events.

1 Select an event, then click *Next*.

If you selected *Ready*, *Shutdown*, or *Overflow*, no further Event information is required. Skip to Step 4.

If you selected *Delivery*, the following dialog box is displayed.

Figure 3-5 *Delivery Notification Screenshot*



2 Select the message classes you wish to receive notification for, then click *Add*.

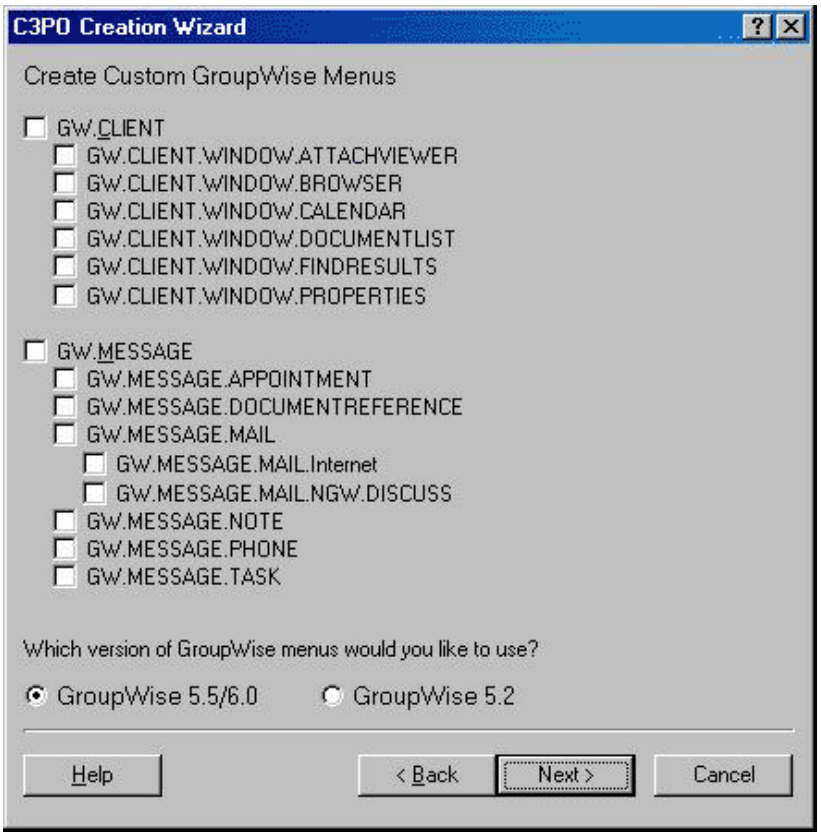
3 Repeat Step 2 for each message class you want to add.

4 Click *Next*.

Customizing Menus

If you selected *I want to customize the GroupWise menus*, the following dialog box is displayed.

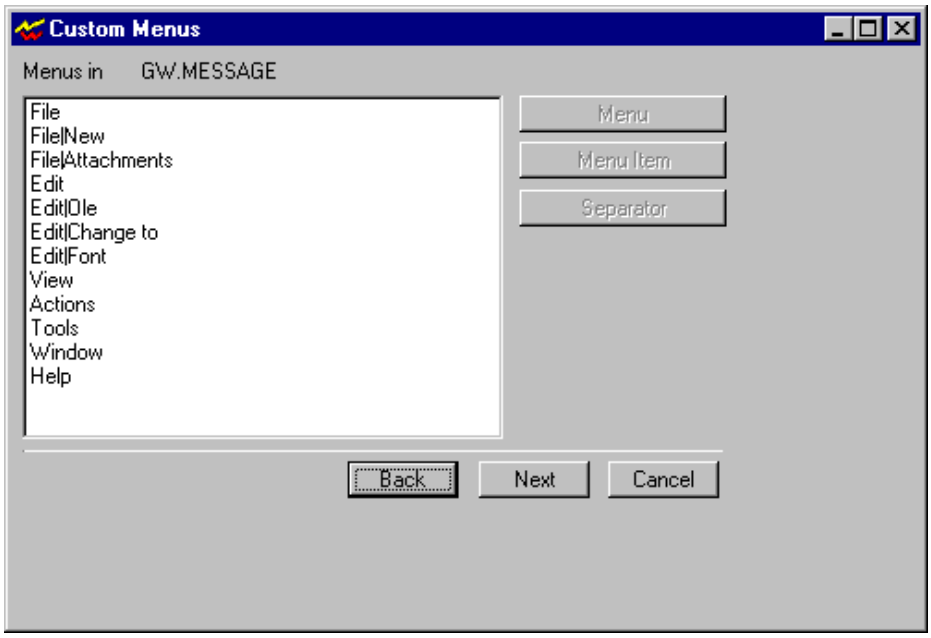
Figure 3-6 *Menu Customization Screenshot*



1. Select the GroupWise views to which you want to add custom menus. If you want your menus added to all GroupWise views, select *GW.CLIENT*.
2. Select the message creation views to which you want to add custom menus. If you want your menus added to all message creation views, select *GW.MESSAGE*.
3. Select the GroupWise version that you are using.

4 Click *Next* to display the Custom Menus dialog box.

Figure 3-7 *Custom Menus Screenshot*



5 Select the menu you want to modify.

6 Click *Add Menu*, *Add Menu Item*, or *Add Separator*. Each option has an associated dialog box that requires information specific to that option.

- If you clicked *Add Menu*, the New Menu dialog box appears. In the Menu Text field, enter the name of the menu, then click *OK*.
- If you clicked *Menu Item*, the New Menu Item dialog box appears. Fill in the fields as explained below, then click *OK*.

  Menu Item Text - The name of the menu item.

  Long Prompt - The text to appear in the long prompt.

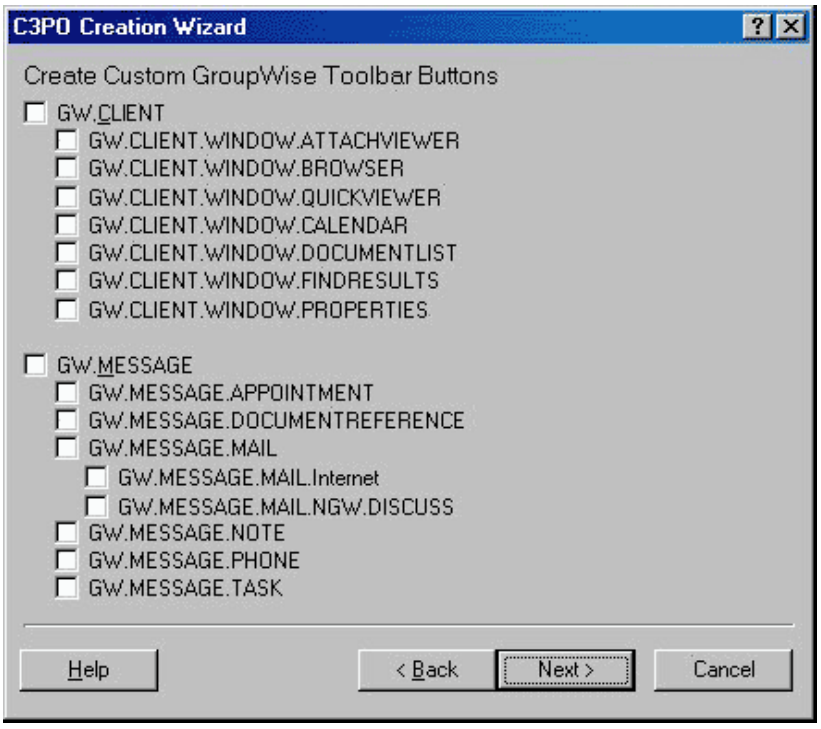
  Variable Name for Command ID Constant - The constant name the wizard will use to identify this menu in the generated code.
- If you clicked *Add Separator*, an information dialog box appears. Click *OK*.

7 Click *Next*.

Customizing the Toolbar

If you selected *I want to customize the GroupWise Toolbar*, the following dialog box is displayed.

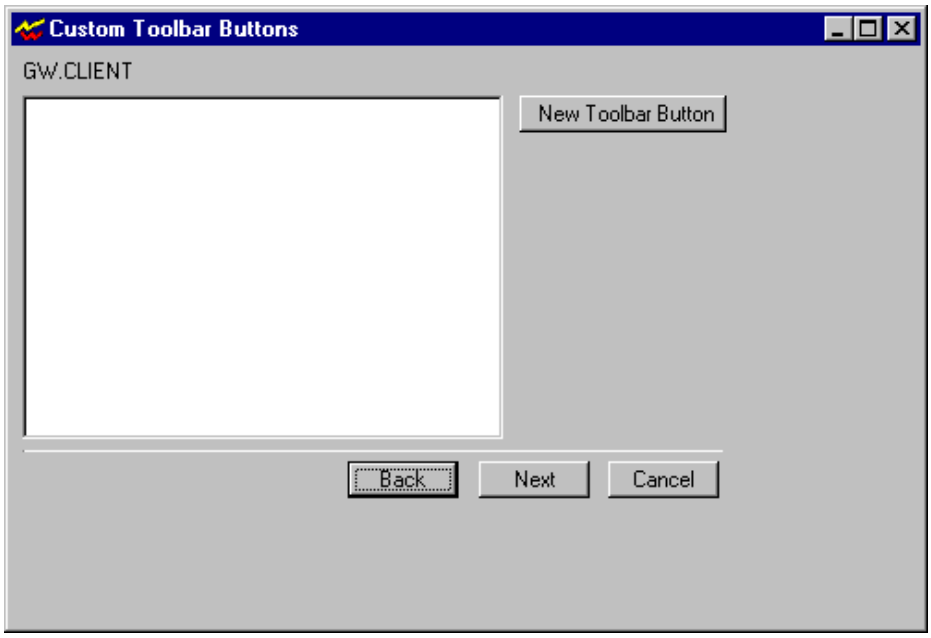
Figure 3-8 *Toolbar Customization Screenshot*



1. Select the GroupWise views to which you want to add Toolbar buttons. If you want your Toolbar buttons added to all GroupWise views, select *GW.CLIENT*.
2. Select the message creation views to which you want to add Toolbar buttons. If you want your Toolbar buttons added to all message creation views, select *GW.MESSAGE*.

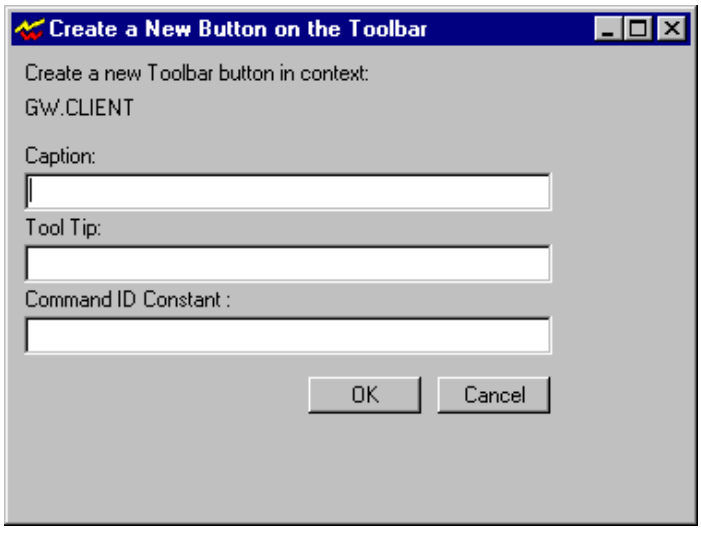
3 Click *Next* to display the Custom Toolbar Buttons dialog box.

Figure 3-9 *Custom Toolbar Buttons Screenshot*



4 Click *New Toolbar Button* to display the following dialog box.

Figure 3-10 *Toolbar Button Creation Screenshot*



5 Fill in the fields as explained below, then click *OK*.

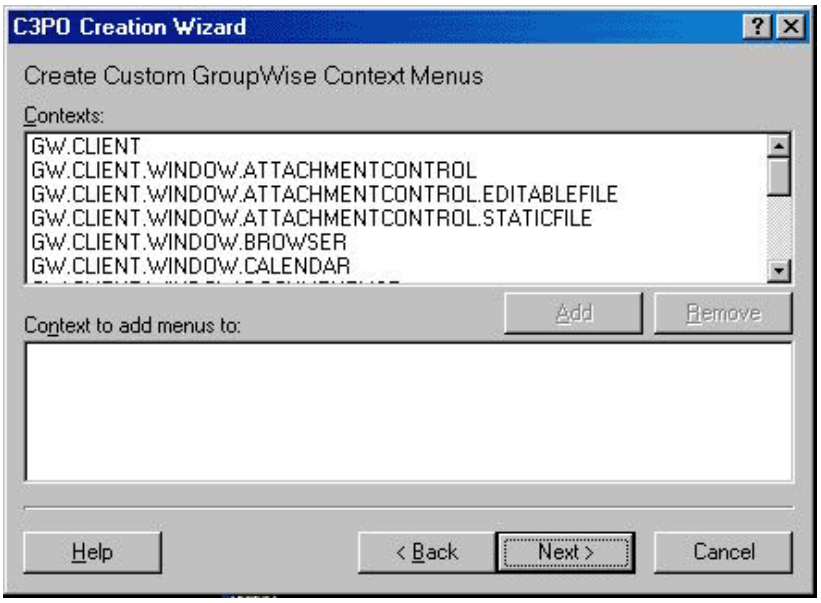
- *Caption* - The name of the menu item.
- *Tool Tip* - The text to appear in the tool tip (long prompt).
- *Command ID Constant* - The constant name the wizard will use to identify this button in the generated code.

6 Click *Next*.

Customizing Context Menus

If you selected *I want to customize the GroupWise context menus*, the following dialog box is displayed. The Contexts list shows all available contexts for the GroupWise client, including any new custom classes that you have defined.

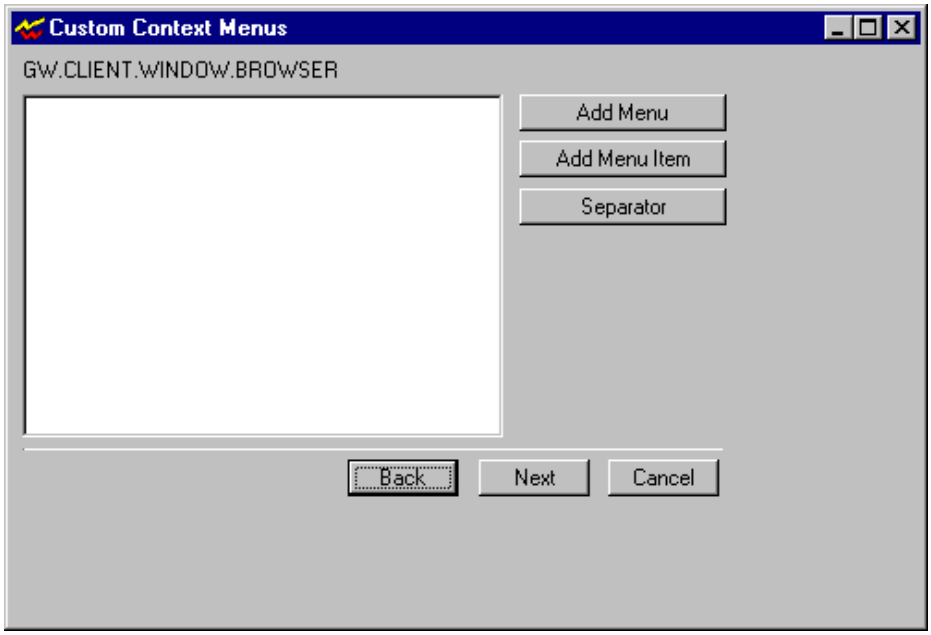
Figure 3-11 *Context Menus Screenshot*



1 To modify a context menu, select the context, then click *Add*.

2 Repeat Step 1 for the context of each menu you want to modify, then click *Next* to display the Custom Context Menus dialog box.

Figure 3-12 *Custom Context Menus Screenshot*



3 Select the menu you want to modify.

4 Click *Add Menu*, *Add Menu Item*, or *Add Separator*. Each option has an associated dialog box that requires information specific to that option.

- If you clicked *Add Menu*, the New Menu dialog box appears. In the Menu Text field, enter the name of the menu, then click *OK*.
- If you clicked *Menu Item*, the New Menu Item dialog box appears. Fill in the fields as explained below, then click *OK*.

  Menu Text - The name of the menu.

  Long Prompt - The text to appear in the long prompt.

  Command ID Constant - The constant name the wizard will use to identify this menu in the generated code.
- If you clicked *Add Separator*, an information dialog box appears. Click *OK*.

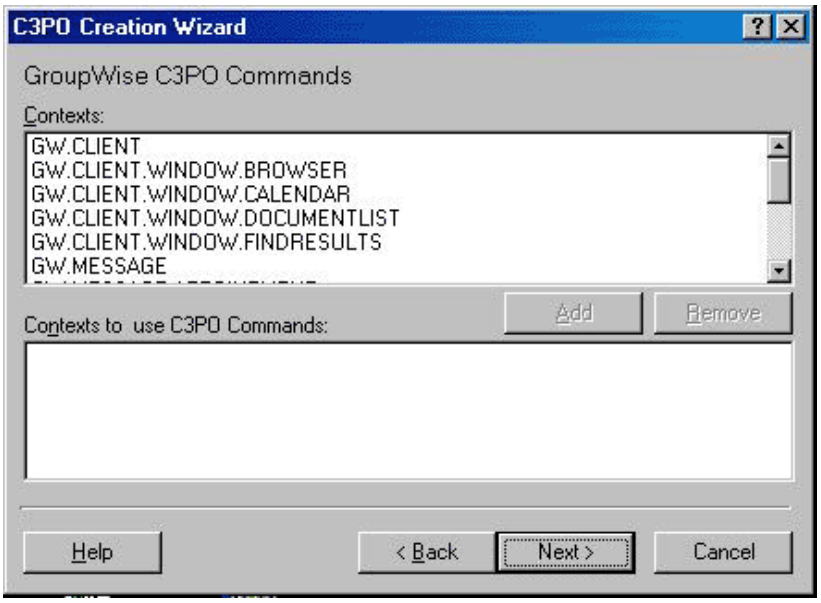
5 Click *Next*. The wizard displays a dialog box for each context selected in Step 1.

6 Repeat Steps 4 and 5 for each context.

Using Commands

If you selected *I want to handle standard GroupWise C3PO* commands, the following dialog box is displayed. The Contexts list shows all available contexts in which you can use predefined GroupWise commands.

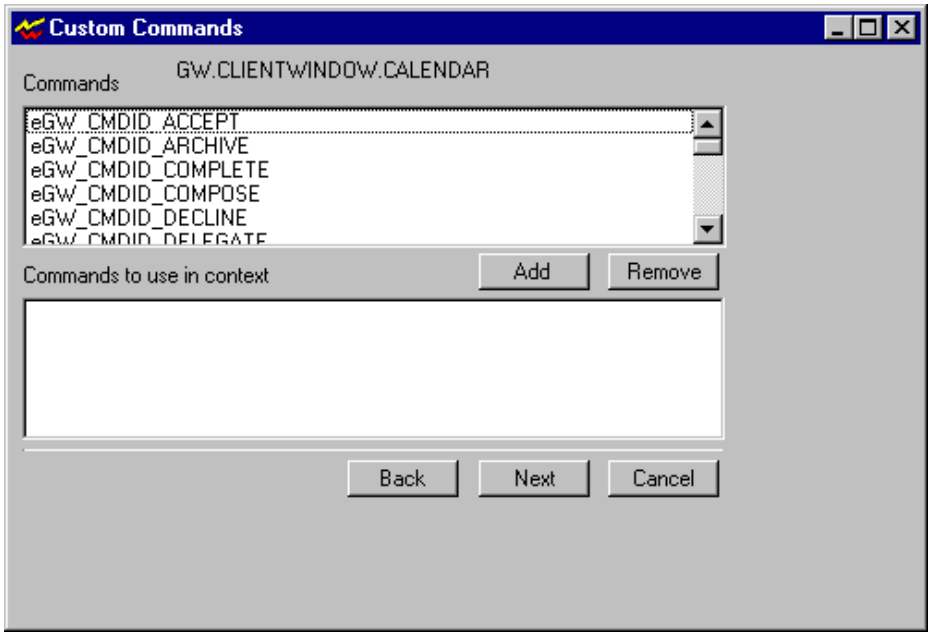
Figure 3-13 *Selecting Contexts Screenshot*



1 Select the context of the command you want to use, then click *Add*.

2 Repeat Step 1 for the context of each command you want to use, then click *Next* to display the Custom Commands dialog box. The Commands list shows the constant names for the predefined commands that you can control.

Figure 3-14 *Selecting Predefined Commands Screenshot*



3 Select a command, then click *Add*.

4 Repeat Step 3 for each command you want to use from the displayed context.

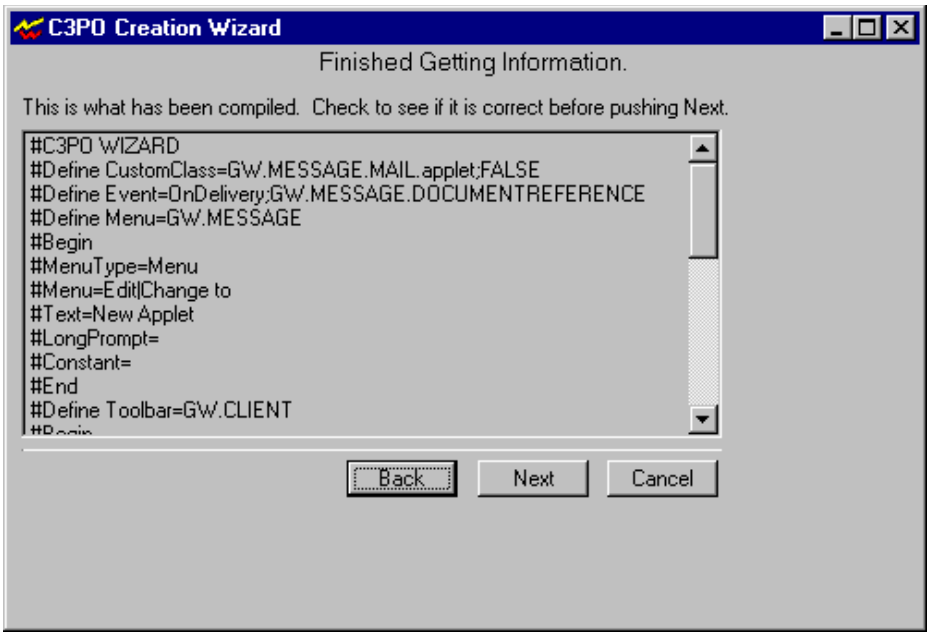
5 Click *Next*. A dialog box is displayed for each context selected in Step 1.

6 Repeat Steps 3 and 4 for each context.

Completing the C3PO

When you have finished entering information, the wizard displays the following dialog box. The list shows a text representation of the information the wizard has compiled.

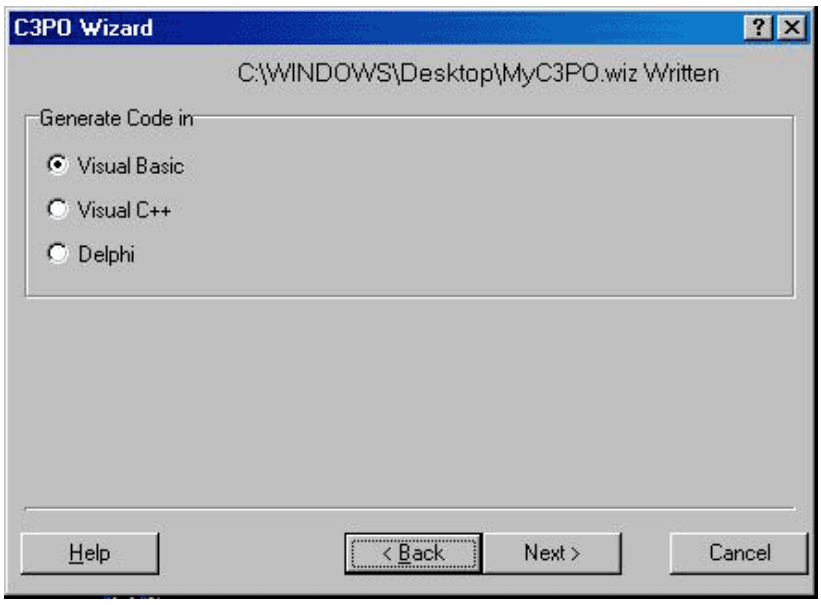
Figure 3-15 *Compiled Information Screenshot*



1 If the information is not correct, click *Back* to make corrections. If it is correct, click *Next* to continue.

A .WIZ file is created in the directory you specified in Step 1 of Starting the Wizard. The C3PO Wizard dialog box is displayed.

Figure 3-16 *Code Type Selection Screenshot*



2 Select the type of code you want to generate: Visual Basic, Delphi, or C++.

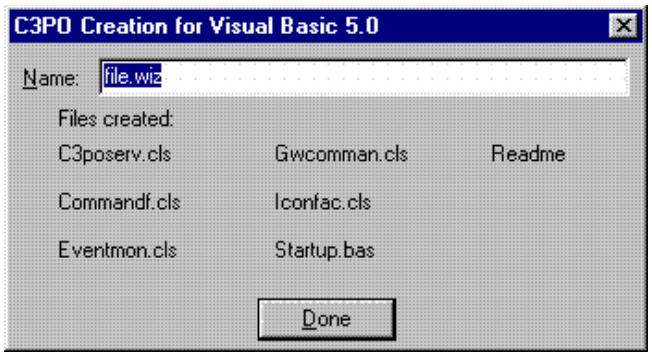
3 Click *Next*.

- If you selected *C++* in Step 2, skip to Step 5.
- If you selected *Visual Basic* or *Delphi* in Step 2, a dialog box appears asking if you want the wizard to create a project file. (Answering *Yes* will make it easier to create an .EXE file in Visual Basic or a .DLL file in Delphi).

4 Click *Yes* if you want the wizard to create the project file. Otherwise, click *No*.

5 The wizard creates the C3PO files for the platform you selected and displays the following dialog box. (The information will vary according to platform.)

Figure 3-17 *Completion Screenshot*



6 Click *Done*.

Now that you have created a C3PO, click an option below for specific instructions.

- *Visual Basic* instructions. The link you choose here depends on your response to Step 4.

  If you told the wizard to create a project file (you answered *Yes* in Step 4), click here for Using Visual Basic.exe.

  If you told the wizard *not* to create a project file (you answered *No* in Step 4), click here for Using Visual Basic .dll

- *Delphi* instructions. The option you choose here depends on your response to Step 4.

  If you told the wizard to create a project file (you answered Yes in Step 4), click here for Using Delphi .dll.

  If you told the wizard to create a project file (you answered Yes in Step 4), click here for Using Delphi .exe.

Using Visual Basic.exe

IMPORTANT: Use these procedures only if you chose to have the wizard create a Visual Basic project (.VBP) file. That is, you answered *Yes* in Step 4 of Completing the C3PO.

(If you answered *No*, Click here for Using Visual Basic .dll.)

This section includes:

New Files

Your new C3PO consists of the following files, which are located in the directory you specified in Step 1 of Starting the Wizard.

- C3POSERV.CLS
- COMMANDF.CLS
- EVENTMON.CLS
- GWCOMMAN.CLS
- ICONFAC.CLS
- README.TXT
- STARTUP.BAS
- XXX.VBP
- XXX.WIZ

Creating an .EXE File

Now that you have created a C3PO for Visual Basic, you need to perform the following steps to make your C3PO work for you. Unless otherwise specified, all files you create using these procedures should be placed in the same directory as the new C3PO files listed above.

1. Start Visual Basic.
2. Load your new C3PO project (XXX.VBP).
3. To enable breakpoints, go to the Project>Properties>Component tab and change the start mode from "Standalone" to "Active X Component."
4. Create an executable file (File Make).

Registering Your C3PO

1. Switch to your Windows desktop.
2. Register the C3PO with GroupWise and Windows by running it from Windows (Start Run) using the syntax: `XXXC3P0.EXE /R`
   - XXXC3PO represents the unique name of your C3PO.
   - The C3PO STARTUP.BAS file calls the sub main procedure.

- sub main calls the RegC3PO command.
- RegC3PO registers the C3PO with both GroupWise and Windows.

(Click here for information about Unregistering a C3PO with Windows.)

Testing Your C3PO

1. Copy the ICONS.DLL file (downloaded with the wizard) into the same directory as your new C3PO files.
2. Switch to Visual Basic.
3. Place a breakpoint in the sub main procedure at the RegC3PO call.
4. Run the C3PO from Visual Basic (Run Start). (The C3PO will not execute because GroupWise has not yet been started.)
5. Start GroupWise.
   - If your C3PO handles the Ready event, or if you created a new menu item or toolbar button which should be visible, the C3PO should now stop at your breakpoint.
   - If your C3PO is waiting for another event, you should cause that event to occur. The C3PO should then stop at your breakpoint.
6. You can now place breakpoints in the other .CLS files to ensure that the appropriate procedures are called and that your C3PO is being implemented correctly.

Unregistering a C3PO

Should you ever need to unregister your C3PO, follow these steps:

1. Switch to your Windows desktop.
2. Unregister the C3PO from GroupWise and Windows by running the C3PO from Windows (Start Run) using the syntax: XXXC3PO /U
   - The C3PO STARTUP.BAS file calls the sub main procedure.
   - sub main calls the RegC3PO command.
   - RegC3PO unregisters the C3PO in both GroupWise and Windows.

Sample C3PO

The GroupWise SDK includes a sample C3PO named C3POBYWIZ.WIZ that you can copy and use as you wish. This sample is located in the following directory:

`Novell/Ndk/GroupWise/Gw_5/C3po/Tools/C3poWizard/Vb/Samples/`

The C3POBYWIZ.WIZ sample:

- Handles Ready and Shutdown events
- Creates a new menu item under the File/New menu
- Creates a new context menu item
- Creates a custom class
- Handles the Open command of that custom class

Using Visual Basic .dll

IMPORTANT: Use these procedures only if you chose *not* to have the wizard create a Visual Basic project (.VBP) file. That is, you answered *No* in Step 4 of Completing the C3PO.

(If you answered *Yes*, click here for Using Visual Basic.exe.)

These procedures assume you chose not to create a .VBP file because you want to create a .DLL file for your C3PO.

This section includes

New Files

Your new C3PO consists of the following files, which are located in the directory you specified in Step 1 of Starting the Wizard.

- C3POSERV.CLS
- COMMANDF.CLS
- EVENTMON.CLS
- ICONFACT.CLS
- GWCOMMAN.CLS
- README.TXT
- STARTUP.BAS
- XXX.WIZ

Creating a .DLL File

Now that you have created a C3PO for Visual Basic, you need to perform the following steps to make your C3PO work for you. Unless otherwise specified, all files you create using these procedures should be placed in the same directory as the new C3PO files listed above.

1. Start Visual Basic.
2. Create a new ActiveX DLL project (File New Project).
3. Remove the class file from the project that automatically was inserted into your new project. Do not save the class file.
4. Insert the STARTUP.BAS file into your project (Project Add Module).

5 Insert the .CLS files into your project (Project Add Class Module).

6 Name your project (use Project>Properties menu item) the same name that you used in the C3PO Wizard.

7 Create your .DLL (File Make).

8 Save your project to the same directory.

9 Remove your project from the Visual Basic environment.

Registering Your C3PO with Windows

1 Register the C3PO with Windows by using the following syntax at a DOS prompt or by running it from Windows (Start Run).

```
C:\WINDOWS\SYSTEM\REGSVR32.EXE XXXC3PO.DLL
```

The REGSVR32.EXE is required for in-process servers. This utility places important information about the server into the systems registry.

- XXXC3PO represents the unique name of your C3PO.
- This syntax assumes the REGSVR32.EXE program on your system is located in the C:\WINDOWS\SYSTEM directory. If it is not, adjust the syntax accordingly.

(Click here for information about Unregistering a C3PO with Windows.)

Registering Your C3PO with GroupWise

1 Switch to Visual Basic.

2 Create a new Standard .EXE project (File New Project).

3 Double-click the form

4 In the form load handler, call the REGC3PO procedure.

5 Add the STARTUP.BAS file to your project (Project Add Module). STARTUP.BAS contains the REGC3PO procedure.

6 Run your project. This will register your C3PO with GroupWise.

(Click here for information about Unregistering a C3PO with Windows and Unregistering Your C3PO with GroupWise.)

Testing Your C3PO

1 Copy the ICONS.DLL file (downloaded with the wizard) into the same directory where your other C3PO files are located.

2 Remove the new .EXE project, and open the .DLL project you created under Creating a .DLL File.

3 Insert breakpoints in your C3PO procedures. To enable breakpoints, go to the Project>Properties>Component tab and change the start mode from "standalone" to "ActiveX component." This will ensure that the appropriate procedures are called and that your C3PO is being implemented correctly.

4 Run the C3PO from Visual Basic. (The C3PO will not execute because GroupWise has not yet been started.)

5 Start GroupWise.

- If your C3PO handles the Ready event, or if you created a new menu item or toolbar button which should be visible, the C3PO should now stop at your breakpoint.
- If your C3PO is waiting for another event, you should cause that event to occur. The C3PO should then stop at your breakpoint.

Unregistering a C3PO with Windows

If you ever need to unregister your C3PO with Windows or DOS, follow these steps:

1 Go to Windows Start>Run or a DOS prompt.

2 Unregister the C3PO by using the syntax:

```
c:\WINDOWS\SYSTEM\REGSVR32.EXE /U XXXC3PO.DLL
```

Unregistering Your C3PO with GroupWise

If you ever need to unregister your C3PO with GroupWise, follow these steps:

1 Switch to Visual Basic.

2 Create a new Standard .EXE project (File New Project).

3 Double-click the form

4 In the form load handler, call the UNREGC3PO procedure.

5 Add the STARTUP.BAS file to your project (Project Add Module). STARTUP.BAS contains the UNREGC3PO procedure.

6 Run your project. This will unregister your C3PO with GroupWise.

Sample C3PO

The GroupWise SDK includes a sample C3PO named C3POBYWIZ.WIZ that you can copy and use as you wish. This sample is located in the following directory:

`Novell/Ndk/GroupWise/Gw_5/C3po/Tools/C3poWizard/Vb/Samples/`

The C3POBYWIZ.WIZ sample:

- Handles Ready and Shutdown events
- Creates a new menu item under the File/New menu
- Creates a new context menu item
- Creates a custom class
- Handles the Open command of that custom class

Using Delphi .dll

IMPORTANT: Use these procedures only if you chose to have the wizard create a Delphi project file (.DPR). That is, you answered *Yes* in Step 4 of Completing the C3PO.

(If you answered *No*, Click here for Using Delphi .exe.)

This section includes

New Files

Your new C3PO consists of the following files, which are located in the directory you specified in Step 1 of Starting the Wizard.

- XXXC3PO.PAS
- XXXREADME.TXT
- XXXSERV.PAS
- XXX.DPR
- XXX.WIZ

Creating a .DLL File

Now that you have created a C3PO for Delphi, you need to perform the following steps to make your C3PO work for you. Unless otherwise specified, all files you create using these procedures should be placed in the same directory as the new C3PO files listed above.

1. Copy the C3POINC.PAS file (downloaded with the wizard) to the same directory as your new C3PO files.
2. Start Delphi.
3. Open the project (XXX.DPR) that was created by the wizard (File Open).
4. Build the project (Project Build).
    - If your C3PO does not use constants, a compile error may occur. If this happens, note which CONST statement is causing the error, comment it out, then recompile.

Registering Your C3PO

1. Switch to your Windows desktop.
2. Register the C3PO by running it from Windows (Start Run) using the syntax:

   ```
   C:\WINDOWS\SYSTEM\REGSVR32.EXE xxxC3PO.DLL
   ```

NOTE: xxxC3PO represents the unique name of your C3PO. Also, this syntax assumes the REGSVR32.EXE program on your system is located in the C:\WINDOWS\SYSTEM directory. If it is not, adjust the syntax accordingly.

3 Registration of your C3PO takes place automatically in GroupWise. No further registration steps are necessary.

(Click here for information about Unregistering a C3PO.)

Testing Your C3PO

1 Switch to Delphi.

2 Open your .DLL project.

3 Select Run Parameters, then enter the path to your GroupWise application (GRPWISE.EXE).

4 Insert breakpoints in your C3PO procedures. This will ensure that the appropriate procedures are called and that your C3PO is being implemented correctly.

5 Run your C3PO. (The C3PO will not execute because GroupWise has not yet been started.)

6 Start GroupWise.

- Depending on which events or menu items your C3PO is programmed to handle, your application should stop at your breakpoints or immediately after you cause your events to occur in GroupWise.

Unregistering a C3PO

Should you ever need to unregister your C3PO, follow these steps:

1 Switch to your Windows desktop.

2 Unregister the C3PO from GroupWise and Windows by running the C3PO from a DOS prompt using the syntax:

```
C:\WINDOWS\SYSTEM\REGSVR32\XXXC3PO /U
```

- The C3PO STARTUP.BAS file calls the sub main procedure.
- sub main calls the RegC3PO command.
- RegC3PO unregisters the C3PO from both GroupWise and Windows.

Sample C3PO

The GroupWise SDK includes a sample C3PO named C3POBYWIZ.WIZ that you can copy and use as you wish. This sample is located in the following directory:

```
Novell/Ndk/GroupWise/Gw_5/C3po/Tools/C3poWizard/Delphi/Samples/
```

The C3POBYWIZ.WIZ sample:

- Handles Ready and Shutdown events
- Creates a new menu item under the File/New menu
- Creates a new context menu item

- Creates a custom class
- Handles the Open command of that custom class

Using Delphi .exe

IMPORTANT: Use these procedures only if you chose not to have the wizard create a Delphi project file (.DPR). That is, you answered *No* in Step 4 of Completing the C3PO.

(If you answered *Yes*, click here for Using Delphi .dll.)

These procedures assume you chose not to create a Delphi project file (.DPR) because you want to create an .EXE file for your C3PO.

This guide consists of the following sections:

New Files

- XXXC3PO.PAS
- XXXREADME.TXT
- XXXSERV.PAS
- XXX.WIZ

Creating an .EXE File

Now that you have created a C3PO for Delphi, you need to perform the following steps to make your C3PO work for you. Unless otherwise specified, all files you create using these procedures should be placed in the same directory as the new C3PO files listed above.

1. Copy the C3POINC.PAS file (downloaded with the wizard) to the same directory as your new C3PO files.
   - You should now have three .PAS files: XXXC3PO.PAS, XXXSERV.PAS, and C3POINC.PAS.
2. Start a new Application project (File New).
3. Save the project (File Save All).
4. Build the project (Project Build).
   - If your C3PO does not use constants, a build error may occur. If this happens, note which CONST statement is causing the error, comment it out, then rebuild.

Registering Your C3PO

1. Switch to your Windows desktop.
2. Register the C3PO with GroupWise and Windows by running the C3PO from Windows (Start Run) using the syntax: XXXC3P0.EXE /R, where XXXC3P0 represents the unique name of your C3PO. The C3PO code will then execute and insert the proper information into the Windows Registry.

   (See Registering Your C3PO for information on unregistering a C3PO.)

Testing Your C3PO

1. Switch to your Windows desktop.
2. Insert breakpoints in your C3PO procedures. This will ensure that the appropriate procedures are called and that your C3PO is being implemented correctly.
3. Run your C3PO. (The C3PO will not execute because GroupWise has not yet been started.)
4. Start GroupWise.
   - Depending on which events or menu items your C3PO is programmed to handle, your application should stop at your breakpoints or immediately after you cause your events to occur in GroupWise.

Unregistering Your C3PO

Should you ever need to unregister your C3PO, follow these steps:

1. Switch to your Windows desktop.
2. Unregister the C3PO from GroupWise and Windows by running the C3PO from Windows (Start Run) using the syntax: XXXC3P0.EXE /U, where XXXC3P0 represents the unique name of your C3PO. The C3PO code will then execute and remove the registry information for this C3PO from the Windows registry.

Sample C3PO

The GroupWise SDK includes a sample C3PO named C3POBYWIZ.WIZ that you can copy and use as you wish. This sample is located in the following directory:

`Novell/Ndk/GroupWise/Gw_5/C3po/Tools/C3poWizard/Delphi/Samples/`

The C3POBYWIZ.WIZ sample:

- Handles Ready and Shutdown events
- Creates a new menu item under the File/New menu
- Creates a new context menu item
- Creates a custom class
- Handles the Open command of that custom class

Using C++

This section includes:

New Files

Your new C3PO consists of the following files, which are located in the directory you specified in Step 1 of Starting the Wizard.

C3POSERVER.CPP	GWCOMMAND.CPP
C3POSERVER.H	GWCOMMAND.H
CLASSFACTORY.CPP	ICONFACTORY.CPP
CLASSFACTORY.H	ICONFACTORY.H
COMMANDFACTORY.CPP	README.TXT
COMMANDFACTORY.H	UTIL.CPP
EVENTMONITOR.CPP	UTIL.H
EVENTMONITOR.H	XXX.WIZ

Standard Files

You also need to import several files (available in the GroupWise C3PO download) to your project.

The following files are available from the c:\novell\ndk\groupwise\gw_5\c3po\tools\c3powizard\samples\cpp\c3pobywiz directory:

- C3PODLL.CPP
- C3PODLL.DEF
- C3PODLL.H

The following files are available from the c:\novell\ndk\groupwise\gw_5\c3po\tools\c3powizard\samples\cpp\include directory:

- GWC3PO.H
- GWC3CMD.H

There are several versions of the last needed file, depending on how many times it has been updated. As of October 2004, the latest version is the gwoapi6.h file, which should be available in several places in your GroupWise download directory.

NOTE: If you are creating a customized button to add to one of your toolbars, add the DLL file for your customized button to the directory where your C3PO source files are located. For example, if you want to add the customized button, which is a picture of some flowers, that comes with the GroupWise download; add the icons.dll file to the same directory where your C3PO source files are located. Otherwise, your button will never appear.

Creating a .DLL File

Now that you have created a C3PO for C++, you need to perform the following steps to make your C3PO work for you. Unless otherwise specified, all files you create using these procedures should be placed in the same directory as the new C3PO files listed above.

The options in parentheses refer to Visual C++ 5.0. If you are using another application builder, your options may differ.

1. Start your C++ application builder.
2. Create a new empty Win32 DLL project (File New). Name the project name the same name you named your project in the C3PO wizard.
3. Insert all new .CPP and .H files created by the C3PO wizard into your project (Project Add to Project Files).
4. Insert the c3podll.def file into your project.
5. Update the wizard to use the newest version of the gwoapi.h file.

   There are several places in the C3PO code that is created by the wizard where the code tries to include the gwoapi.h header file. If you are using a later version of gwoapi.h, be sure to change all references to use the version of the file that you are using (for example, gwoapi6.h).
6. Edit the c3podll.cpp file.

   Go to the DLLMain function. Change the line that states

   ```
   CTShInstance = hModule
   ```

   to read

   ```
   CTShInstance = (HINSTANCE)hModule
   ```
7. If you are adding your own customized button, edit the commandfactory.cpp file to change the references to icons.dll to use your new dll name.

   You also need to change the reference to the gwcommand.cpp file to the new ID of your customized button.
8. Build your .DLL file (Build Project).

Registering Your C3PO

1. Switch to your Windows desktop.
2. Register the C3PO with GroupWise and Windows by running it from Windows (Start Run) using the syntax:

   ```
   C:\WINDOWS\SYSTEM\REGSVR32.EXE XXXC3PO.DLL
   ```

   NOTE: xxxC3PO represents the unique name of your C3PO. Also, this syntax assumes the REGSVR32.EXE program on your system is located in the C:\WINDOWS\SYSTEM directory. If it is not, adjust the syntax accordingly.

   (Click here for information about Unregistering a C3PO.)

Testing Your C3PO

1. Switch to your C++ application builder.
2. Open the .DLL project you created under Creating a .DLL File.
3. Insert breakpoints in your C3PO procedures. This will ensure that the appropriate procedures are called and that your C3PO is being implemented correctly.
4. Run the C3PO from your C++ IDE. (The C3PO will do nothing because GroupWise has not yet been started.)
5. Start GroupWise.
   - Depending on which events or menu items your C3PO is programmed to handle, your application should stop at your breakpoints or immediately after you cause your events to occur in GroupWise.

Unregistering a C3PO

Should you ever need to unregister your C3PO, follow these steps:

1. Switch to your Windows desktop.
2. Unregister the C3PO from GroupWise and Windows by running the C3PO from a DOS prompt using the syntax:

   ```
   C:\WINDOWS\SYSTEM\REGSVR32\XXXC3PO /U
   ```

Sample C3PO

The GroupWise SDK includes a sample C3PO named C3POBYWIZ.WIZ that you can copy and use as you wish. This sample is located in the following directory:

```
Novell/Ndk/GroupWise/Gw_5/C3po/Tools/C3poWizard/Cpp/Samples/
```

The C3POBYWIZ.WIZ sample:

- Creates a new menu item under the File menu
- Creates a new button

4 Reference

This section contains information for the C3PO objects, available contexts, prebuilt command IDencoding, predefined GroupWise client identifiers, and sample applications.

Objects

Available objects include the following:

AttachmentControl

Provides information about an attachment to a message. Valid for GroupWise 5.5 EP and GroupWise 6.x.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
AttachDisplayName (Long index)	R/O	String. Display name of the attachment.
AttachFileName (Long index)	R/O	String. Name of the file.
AttachType (Long index)	R/O	Long. Type of the attachment.
CurrentAttach	R/O	Long. Attachment being accessed.

Methods

None.

Remarks

None.

C3POServer

Used to initialize the C3PO. Every C3PO must support this interface.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
CommandFactory	R/O	CommandFactory object. Invoked to obtain the command source for the C3PO. The C3POServer may return NULL. If the C3POServer returns NULL (or returns an error), the C3POManager will skip the C3PO. That is, the Manager will consider the NULL (or error) to be a skip signal. Such a signal is not considered a system error by the Manager and will not propagate the error outside of the Manager.
Description	R/O	String. Returns a readable description of the C3POServer.
EventMonitor	R/O	EventMonitor object. Invoked to obtain the event monitor for the C3PO. If the C3POServer returns NULL (or an error), the C3POManager will skip the C3PO. Such a signal is not considered a system error and will not propagate the error outside of the C3POManager.
IconFactory	R/O	IconFactory object. Invoked to obtain the icon source for the C3PO. If the C3POServer returns NULL (or an error), the C3POManager will skip the C3PO. Such a signal is not considered a system error and will not propagate the error outside of the C3POManager.

Methods

`Boolean CanShutdown( )`

Invoked to query whether or not the C3POServer can shut down. Typically used when the C3POServer has a window open on the UI screen and shutdown is possible. Each C3POServer is guaranteed to have CanShutdown queried at least once before the C3PO system shuts down.

CanShutdown is a request from the C3POManager to a C3PO. It is not a request to shut down a C3PO, but to shut down the C3POManager and disconnect the C3PO. Typically a C3PO shuts down with the C3POManager.

No particular C3PO order can be relied on for querying the CanShutdown property.

A returned error is interpreted by the C3POManager to be a positive response (that is, the C3PO can be shut down). This prevents a an errant C3PO from causing an interference, making it impossible to exit the client.

`DeInit( )`

Terminates the relationship of the C3POManager with the C3POServer. This is a separate issue from the shutdown sequence (including shutdown events). For example, a C3PO can be unloaded from memory as a runtime optimization in which case the C3POServer first receives calls to CanShutdown followed by DeInit. The C3PO is unloaded, but the client application has not necessarily terminated.

The C3POManager pointer passed in to C3POServer::Init is still valid while calling DenInit. However, when DeInit returns, the C3POManager pointer is not guaranteed to be valid. The C3POServer must release all holds to the C3POManager during the DeInit( ) call.

One call to DeInit is issued to the C3POServer for each C3PO client using the C3PO system. C3PO software that wants to be capable of loading into multiple clients (irrespective of process boundaries) should be multiple-instance OLE servers. That is, a new C3POServer object should be created for each C3POManager that wants to use the services of the C3PO. By tracking the Manager pointer for each C3POServer, the C3PO can determine which requests are being issued from which clients.

Errors returned from this method are ignored by the C3POManager.

```
Init( C3POManager Manager )
```

The Manager object is valid until a future DeInit call is made. That is, the C3PO does not need to AddRef this object, but can simply store it for the life of the C3PO until DeInit is called.

This method is the first method invoked in the C3POServer object when loading a C3POServer. If the server fails this method (via HRESULT), the C3POServer is unloaded.

One call to Init is issued to the C3POServer for each C3PO client using the C3PO system. C3PO software that wants to be capable of loading into multiple clients (irrespective of process boundaries) should be multiple-instance OLE servers. That is, a new C3POServer object should be created for each C3POManager that wants to use the services of the C3PO. By tracking the Manager pointer for each C3POServer, the C3PO can determine which requests are being issued from which clients.

An error returned from this method will abort the loading of the C3POServer.

Remarks

None.

C3POServer2

Used to provide additional methods to support presence and telephony.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
PresenceFactory	R/O	PresenceFactory object. Returns the factory object to create presence objects.

Methods

```
DisplayName(
string *bsName
)
```

Returns a string that names the C3PO, such as could be displayed in a combo box in the interface. Used for the default telephony provider selection.

Remarks

None.

CalledPhoneNumber

Provides the phone number to be dialed as the result of a user action. Valid for GroupWise 8.x.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
CalledContact	R/O	DIGWAddressBookEntry7. The contact that owns this phone number. May be NULL if unknown.
CalledPhoneNumber	R/O	String. The phone number that was called.
CalledPhoneNumberType	R/O	Long. The type of the phone number (home, office, fax, etc).
IsDefaultPhoneNumber	R/O	Boolean. TRUE if this is the default phone number fo the user

Methods

None.

Remarks

None.

CalledPhoneNumbers

Provides a list of phone numbers to be dialed. Valid for GroupWise 8.x.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Count	R/O	Returns the number of phone numbers called.

Methods

```
Item (lIndex,
      CalledPhoneNumber retval);
```

Returns the called phone number at the given index (0 based).

Remarks

Currently, only one phone number can be called at a time.

ClientState

Passed to the C3POServer during UI modification operations. The C3PO can call back through this interface to discover UI contextual information. ClientState serves as the base class for client-specific subclasses. The GroupWise client issues the GWClientState object as the specific ClientState derivative. Other clients have their own subclasses. A C3PO should never assume a specific ClientState subclass. Rather, the ClientName, MajorVersion, and MinorVersion properties should be checked to identify the appropriate ClientState subclass.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
CallDescription	R/O	String. An optional description of why the Dial command is being called
CalledPhoneNumbers	R/O	Object. A list of called phone numbers, if the event being handled is a Dial event.
ClientName	R/O	String. The name of the C3PO client. Because each C3PO client typically provides a subclass of the ClientState object, the ClientName, MajorVersion, and MinorVersion are used to differentiate them. The ClientName is assumed to be non-translatable and alike for all versions of a client.
CurrentAccount	R/O	Object. Returns account object that represents the currently selected account in the client.
MajorVersion	R/O	Long. The major version number for the client.
MinorVersion	R/O	Long. The minor version number for the client.

Methods

```
GWCommand BuildCommand( String Context, String PersistentID, GWCommand
BaseCommand, Parameters parms )
```

Exposing this method allows a C3PO to build commands from persistent IDs available to it. Building a GWCommand through this interface implies that the command will call out to applicable C3PO software. For example, a C3PO must not bind to and throw a command as part of executing that command. Also, circular chains of commands are not detected. There is no built-in protection against infinite loops.

Rather than a Flag Not Set token-based command, the client builds a Flag Set token-based command followed by a call to BuildCommandAggregate. This implies that the GWMenu object knows what context to support when it calls the build object. The applicable context appears as a parameter on the BuildGWMenu and BuildGWToolbar items.

The context in this command refers to the context for the command. The context string contains either the class of a GW data object (such as GW.MESSAGE.MAIL.X) or the class of a specific GW user interface (such as GW.CLIENT.WINDOW.BROWSER).

The persistent ID can be either a predefined ID or an ID provided by a C3PO in a previous session of GroupWise. (See Predefined GroupWise Client Identifiers.)

BaseCommand is the command being composed (overridden).

Parms is the parameter list for the command. Some clients may return NULL for any invocation of this method.

```
String BuildPersistentID( long ID, [variant ObjFrom] )
```

Used to build a persistent ID from another integer ID in the system. For example, the C3POManager invokes this method to change a menu ID into the associated persistent ID.

ID in this command is the integer identifier that is being permuted into a persistent ID.

ObjFrom is optional, and is the object that identifies the ID. For IDs from the toolbar, the ObjFrom is the IDispatch pointer for the GWToolbar object. For menu IDs, ObjFrom is the IDispatch pointer for the GWMenu object. Because GroupWise 5.x and 6.x always omits this parameter, C3PO clients should be able to uniquely distinguish a persistent ID from any integer ID in the system.

Returning an empty string, NULL, or an error are all interpreted to mean no persistent ID is available.

```
PresenceUpdated(DIGWPresence2 *pDispPresence)
```

Call this method whenever your presence state changes, so that it can be reflected in the client user interface

Remarks

None.

ClientState Implementation (Subclass)

The subclass of ClientState that the GroupWise client passes to a C3PO.

Properties

The following table lists the properties for this class, along with access and descriptions:

Property	Access	Description
AttachmentControl	R/O	AttachmentControl. Available beginning with the ClientState3 interface. Valid for GroupWise 6.0 and support packs for GroupWise EP.
ClientName	R/O	String. "GroupWise."
CommandMessage	R/O	Message. The Message object associated with the command object being executed. For example, when a predefined command is issued (such as Forward, Open, or Delegate), this property identifies the message associated with the command. CommandMessage isn't available for new messages. If you try to access Manager.ClientState.CommandMessage from a new message, GroupWise 6.5 throws an “Unspecified error” exception.
CurrentAccount	R/O	Account. The account object currently being accessed. Available beginning with the ClientState2 interface.
HighlightedFolder	R/O	Returns the currently highlighted folder in the folder tree. Usually, the highlighted folder is identical to the selected folder. However, they differ when the user is right-clicking another folder. The folder that is being right-clicked will be highlighted, but the previous folder still remains selected.
HTMLEditor	R/O	Provides editing access to the IHTMLDocument interface provided by the Internet Explorer browser.
HTMLViewer	R/O	Provides viewing access to the ITHMLDocument interface provided by the Internet Explorer browser.
MajorVersion	R/O	Long. 5.
MinorVersion	R/O	Long. 0.
SelectedFolder	R/O	Folder. The currently selected folder. For open message items, this property is NULL. For the browser window, this is the folder selected in the folder tree. Calendar windows have the ability to multi-select the folders that are displayed, in which case, this property is the "caret" folder (or focus selection) in the folder tree.
SelectedMessages	R/O	MessageList. The currently selected set of messages.

Methods

```
RevokeDeliveryFolder( string MessageClass, [Variant Target] )
```

Reverses the effect of calling SetupDeliveryFolder. Messages of the given class will no longer be linked to the target folder. If multiple servers are monitoring a given message class, this method will remove the delivery folder for all servers. There is no type of "reference count" associated with calls to SetupDeliveryFolder.

MessageClass names a specific message type.

Target names the target folder. Target can be a Folder object from the Object API or a string. A string value is interpreted to be a full folder path. For example, \\user_full_name\folder A\Folder B would indicate that Folder B is the target folder. Note that for GroupWise 5.x and 6.x, this parameter must always be omitted.

```
SetupDeliveryFolder( string MessageClass, [Variant Target])
```

Causes messages of the specified class to be automatically linked to the named folder. This is accomplished by creating a rule that causes items of the designated class to be linked to the folder upon delivery of the item.

MessageClass names a specific message type.

Target names the target folder. Target can be a Folder object from the Object API or a string. A string value is interpreted to be a full folder path. For example, \\user_full_name\folder A\Folder B would indicate that Folder B is the target folder. Note that for GroupWise 5.x and 6.x, this parameter must always be omitted.

A C3PO would invoke this method to facilitate the management of messages. For example, processing of delivered items can be accomplished by using this method. The C3POServer would query the folder when the deliver event is passed to the EventMonitor.

If more than one C3PO invokes this method with the same parameters, the GWClientState object will attempt to resolve to the same rule. This is done through the naming of the rule.

Each C3POServer should take care to invoke this method only once. The server must keep a one-time init flag (perhaps in the registry) to guarantee that it invokes the method only once. The startup overhead with this API is substantial and should be avoided.

Remarks

GWClientState is not intended to replace the functionality of the Object API. The GroupWise 5.x and 6.x implementations of GWClientState is intentionally minimal because adequate functionality is available in the Object API.

CommandFactory

Used to manage commands in GroupWise. A command occurs in multiple locations, including the menu and toolbar. This object has no properties.

Properties

None.

Methods

```
BuildCommand( Context: string;
PersistentID: string;
BaseCommand: variant;
parms: variant): variant;
```

This method is invoked to create a Command object. The C3PO cannot assume that Command::WantCommand has been called before this method is invoked.

In

Context: string is the context for the command. The context string contains either the class of a GW data object (for example, GW.MESSAGE.MAIL.X) or the class of the specific user interface (for example, GW.APPLICATION.BROWSER).

PersistentID: string can be either a "pre-defined" ID or is an ID provided in a previous session of GroupWise.

BaseCommand: variant is the command being overridden.

parms: variant is the parameter list of the command.

Out

GWCommand: variant is the command object of the command being overridden. Each time the BuildCommand is invoked, the C3PO should return a new instance of a GWCommand object. This is neccessary because BuildCommand can be called recursively

```
CustomizeContextMenu( Context: String;
GWMenu: Variant );
```

Method to customize a Context Menu.

In

Context: string is the class of the object owning the menu. For example, ObjClass would contain "GW.MESSAGE.MAIL" for the mail messages.

GWMenu: variant is the context menu to be modified.

Out

None.

```
CustomizeMenu( Context: string;
GWMenu: variant ): TOleBool;
```

Method to customize the menu.

In

Context: string is the class of the containing window. For example "GW.APPLICATION.BROWSER"

GWMenu: variant is the main menu object being modified.

Out

TOleBool. The return value indicates whether the modifications to the menu were "volatile." If the return value is TRUE, CustomizeMenu( ) will continue to be called each time the menu may need to be updated, such as at each popup creation). Otherwise, C3POServers can safely assume that CustomizeMenu is called only once for any single instance of a menu. This simplifies the resulting code because there is no need to check for commands that might already be on the menu. That is, the C3PO commands are guaranteed to be absent from the menu at the time of the first CustomizeMenu call.

```
CustomizeToolBar( Context: string;
GWToolbar: variant ): TOleBool;
```

Method to customize the toolbar. The C3PO must assume that its commands are absent from the toolbar. For example, when a toolbar has been saved and restored in another session of GroupWise, the commands can be put back onto the toolbar without any intervening calls to CustomizeToolbar. For that reason, the C3PO should query the toolbar first before adding anything to the toolbar window.

In

Context: string contains the class of the containing window.

GWToolbar: variant contains the GWToolbar object to be modified.

Out

TOleBool. Returning TRUE from this method indicates that the toolbar modifications were "volatile." In that case, each time the toolbar is validated in the UI, the CommandFactory::CustomizeToolbar method will be called.

```
Init( lcid: longint ): longint;
```

This method is the first method of Command::Factory called. It is used to tell GroupWise what UI items will be modified.

In

lcid: longint. This is the local ID of the application driving the C3PO.

Out

The C3POServer returns any combination of the following bits:

eGW_CMDINIT_MENUS	The C3PO intends to modify the menus (CustomizeMenu Interface).
eGW_CMDINIT_TOOLBARS	The C3PO intends to modify the toolbar (CustomizeMenu Interface).
eGW_CMDINIT_CONTEXT_MENUS	The C3PO intends to modify the context menus (CustomizeMenu Interface)
eGW_CMDINIT_NO_PREDEFINED	Optimization flag that indicates the C3PO will never respond to predefined commands. Returning this flag will suppress calls to the WantCommand method for predefined commands.

```
WantCommand( Context: string; PersistentID: string ): TOleBool;
```

This method is used to query the C3PO for its intention to support a predefined command.

In

Context: string is the actual object class associates with the command.

PersistentID: string can be either a pre-defined ID or an ID provided in a previous session of GroupWise.

Out

TRUE. I do want to support this predefined command.

FALSE.. I do not want to support this predefined command.

Remarks

None.

EventMonitor

Passed to monitor low-level actions that occur in GroupWise. This method has no properties.

Properties

None.

Methods

```
Notify( String Context, Event evt )
```

Notification that an event occurred. The evt parameter represents one of these events:

Event	Description
eGW_EVT_DELIVERY	Indicates that an item has been delivered to the GroupWise mailbox. A C3PO will typically invoke the GWClientState::SetupDeliveryFolder method and query the corresponding folder when this event is broadcast.
eGW_EVT_READY	This notification occurs immediately after GroupWise is initialized. The following services are running at the time this method is called: DDE server; OLE objects registered and placed in the COM running object table.
eGW_EVT_SHUTDOWN	A notification to the C3POServer that shutdown is occurring. System services are still available when this notification is received. Immediately after returning from this method call, the C3PO should assume that the calling client has shut down. If the calling client is the last client to use the C3POManager, calls to the Manager will fail after this call has returned.
eGW_EVT_OVERFLOW	This notification indicates that too much information has been returned at the same time. When this occurs, you should query the corresponding folder to determine if the information you requested is there.

Remarks

None.

GWCommand

Defines an instance of a command.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
BaseCmd	R/O	GWCommand. Returning an error is equivalent to returning NULL, which is legal.
LongPrompt	R/O	String. Only commands generated and placed on the menu/toolbar with CommandFactory::CustomizeXXX call this method. Returning an error is equivalent to returning NULL, which is legal.
Parameters	R/O	Always NULL.
PersistentID	R/O	String.
ToolTip	R/O	String. Only commands generated and placed on the toolbar with CommandFactory::CustomizeToolbar (or explicitly added by the user) call this method. Returning an error is equivalent to returning NULL, which is legal.

Methods

`Execute( )`

`Help( )`

Invokes help for the GWCommand. Though the GroupWise client does not invoke this method, C3POServers are nevertheless required to implement it because other (add-on) C3PO clients may invoke it.

`Undo( )`

Required method. Not called in GroupWise 5.x and 6.x. Though the GroupWise client does not invoke this method, C3POServers are nevertheless required to implement it because other (add-on) C3PO clients may invoke it.

`Long Validate( )`

Called to determine the state of a C3PO command. The command is assumed by default to be in an enabled, unchecked, visible state. The C3PO provider can then modify that assumption by returning flags. The available flags are:

eGW_CMDVAL_CHECKED	The command has a check mark.
eGW_CMDVAL_DISABLED	The command is disabled.

Validate applies only to commands placed on the toolbar or menu by the C3POServer. Predefined commands do not cause this method to be invoked.

If the C3PO returns an error, the command is assumed to be disabled and unchecked.

Remarks

When GWCommand objects are associated with menu items, such as when CommandFactory::ModifyMenu is called, the lifetime of the command will parallel that of the menu.

GWEvent

Defines an instance of an event.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
PersistentID	R/O	String. The persistent ID.

Methods

None.

Remarks

None.

GWMenu

Manages a client application's menus. The C3PO uses this object to modify the menu.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Caption	R/W	String.
GWCommand	R/W	GWCommand( ) . For pre-built items, the CommandFactory( ) in the ClientState( ) object is used to build the value of this property. Otherwise this property will be NULL for pre-built items.
IsModified	R/O	Boolean.
MenuItems	R/O	GWMenuItems( ).
ObjType	R/O	enumeration (always eGW_GWMENU). Indicates the proper subtype of GWMenuItem( ) being referenced. For example, the GWMenuItems::Item( ) method returns either a GWMenuAction( ) or a GWMenu( ) object, both of which are subtypes of GWMenuItem( ). The ObjType can be used to distinguish which object type has been returned.
Parent	R/O	GWMenu( ).

Methods

`Delete( )`

Deletes GWMenu. This method removes the item from the UI menu and releases the AddRef on GWCommand. Root menus cannot be deleted.

Remarks

None.

GWMenuAction

Represents a simple menu item, which is not a submenu.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Caption	R/W	String.
GWCommand	R/W	GWCommand( ). For pre-built items, the CommandFactory( ) in the ClientState( ) object is used to build the value of this property. Otherwise this property will be NULL for pre-built items.
MenuID	R/O	Long. The menu ID associated with the menu. When a C3PO adds items to a GWMenu, the MenuIDs are allocated automatically. The C3PO can read this property to obtain the allocated ID.
ObjType	R/O	enumeration (always eGW_GWMENUACTION). Indicates the proper subtype of GWMenuItem( ) being referenced. For example, the GWMenuItems::Item method returns either a GWMenuAction( ) or a GWMenu( ) object, both of which are subtypes of GWMenuItem( ). The ObjType can be used to distinguish the object type that has been returned. For the GWMenuAction( ) object, this property is always set to eMenuAction.
Parent	R/O	GWMenu( ).

Methods

`Delete( )`

Deletes GWMenuAction, which removes the item from the UI menu and releases the AddRef on the GWCommand object.

Remarks

None.

GWMenuItem

Associates a GWCommand( ) object with a menu item.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Caption	R/W	String.
GWCommand	R/W	GWCommand. For pre-built items, the CommandFactory in the ClientState object is used to build the value of this property. Otherwise, this command will be NULL for pre-built items. A client can also implement the GWCommand factory, but still return NULL for certain menu items. Hence, C3POServers should never assume they will receive non-NULL GWCommand objects for every menu item.
ObjType	R/O	enumeration (eGW_GWMENU, eGW_GWMENUACTION, eGW_GWMENUSEPARATOR) Indicates the proper subtype of GWMenuItem being referenced. For example, the GWMenuItems::Item method returns either a GWMenuAction or a GWMenu object, both of which are subtypes of GWMenuItem. The ObjType can be used to distinguish the object type that has been returned.
Parent	R/O	GWMenuItems.

Methods

`Delete( )`

Deletes GWMenuItem, which removes the item from the UI menu and releases the AddRef on the GWCommand object.

Remarks

None.

GWMenuItems

Enumerates the items on a menu. The menu items collection maintains references to the commands placed on the menu (GWMenuItems::Add). When the corresponding GWMenu is destroyed, the GWCommand objects placed on the menu are released.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Count	R/O	Long.
Parent	R/O	GWMenu.
_NewEnum	R/O	Enumeration object Implements IEnumVARIANT. Fo´r Windows only.

Methods

```
GWMenuAction Add( String caption, GWCommand item [,Variant AddBefore] )
```

The AddBefore parameter refers to either a string (display name) of a menu item or to an index (ordinal).

```
GWMenu AddMenu( String caption [, GWCommand item ] [, Variant AddBefore] )
```

The AddBefore parameter refers to either a string (display name) of a menu item or to an index (ordinal).

The item parameter is optional, but it is useful for long prompt functions for the submenu.

```
GWMenuSeparator AddSeparator( [ Variant AddBefore] )
```

The AddBefore parameter refers to either a string (display name) of a menu item or to an index (ordinal).

```
GWMenuItem Item( Variant item )
```

Returns either GWMenu, GWMenuAction, or GWMenuSeparator. The return value can be identified by querying the ObjType property.

The item parameter refers to either a string (display name) of a menu item or to an index (ordinal).

```
GWMenuItem FindByHMenu( Long hMenu )
```

Returns the GWMenu object that contains a Command property with the menu handle.

This method returns NULL when AddMenu is called without providing a GWCommand object. In this case, the client (caller of this method) should assume that the menu is enabled and has no long prompt.

This method is recursive. If the hmenu is contained in a menu that is inside the MenuItems collection, FindByHMenu locates it.

```
GWMenuItem FindByID( Variant ID )
```

Returns the GWMenuItem object that corresponds to the menu identifier. The ID parameter refers to the actual Menu ID, not to a PersistentID. Separators cannot be found within this method.

The ID parameter refers to the number (actual menu ID).

This method is recursive. If the ID is contained in a menu that is inside the MenuItems collection, FindByIDl locates it.

Remarks

C3PO software generally adds items to (or remove them from) the menu while calling the CommandFactory methods, such as CustomizeMenu. In this case the C3PO does not redraw the menu bar. However, if the C3PO maintains a reference to GWMenuItems and adds (or removes) items at another time, there is no guarantee that the menu bar will be redrawn properly.

The Item and AddBefore parameters do not allow you to specify a persistent ID. The workaround is to call FindByID method to obtain GWMenuItem, then pass the menu ID or display string available from that object into the Item method or as the AddBefore parameter.

GWMenuSeparator

Indicates a menu separator.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Caption	R/W	String. Always empty.
GWCommand	R/W	GWCommand. Always NULL.
ObjType	R/O	enumeration (always eGW_GWMENUSEPARATOR). Indicates the proper subtype of GWMenuItem( ) being referenced. For example, the GWMenuItems:Item method returns either a GWMenuAction or a GWMenu object, both of which are subtypes of GWMenuItem. The Objtype can be used to distinguish which object type has been returned.
Parent	R/O	GWMenuItems.

Methods

`Delete( )`

Deletes the GWMenuSeparator (which removes the item from the UI menu) and releases the AddRef on the GWCommand object.

Remarks

None.

GWToolbar

Manages GroupWise toolbars.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
ToolbarItems	R/O	GWToolbarItems.
IsModified	R/O	Boolean.

Methods

None.

Remarks

None.

GWToolbarItem

Associates a GWCommand object with a toolbar item. When a C3PO adds items to the toolbar, the GWToolbar maintains a reference to the GWCommand object placed onto the toolbar. When the toolbar is destroyed, the GWCommand object is released.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
GWCommand	R/W	GWCommand. For pre-built items, CommandFactory in the ClientState object is used to build the value of this property. Otherwise, GWCommand is NULL for pre-built items. A client can implement the GWCommand factory, but still return NULL for certain toolbar items. Hence, C3POServers should never assume they will receive non-NULL GWCommand objects for each toolbar item.
Parent	R/O	GWToolbarItems.
ToolbarID	R/O	Long. The actual ID assigned to the toolbar item. When GWToolbarItems::Add is invoked, the new GWToolbarItem is automatically assigned an ID number. This property can be queried by the C3PO to retrieve the ID.

Methods

```
Delete( )
```

Deletes the toolbar item (which removes the item from the toolbar) and releases AddRef on the GWCommand object that is associated with the toolbar item.

```
SetBitmap( String Filename, Variant ResID )
```

Sets the bitmap image for the toolbar item. When GWToolbarItems::Add is called, the returned GWToolbarItem has been assigned a default image. Calling this method will change that image. In GroupWise 5.x and 6.x, the new item is not displayed on the toolbar by default; you must call SetBitmap to display the item on the toolbar.

Filename is the name of the file that contains the bitmap.

ResID can be either a resource ID (long) or a resource name (string).

GroupWise client bitmaps are associated with a color map which a C3PO can use for transparency and highlight effects. The map is as follows:

Table 4-1 *GroupWise Color Map*

Color Mapped From	Color Effect
RGB(0, 255, 0)	Transparent
RGB(192, 192, 192)	COLOR_3DLIGHT (Win32 color)
RGB(255, 255, 255)	COLOR_3DHILITE (Win32 color)
RGB(128, 128, 128)	COLOR_3DSHADOW (Win32 color)

In GroupWise, color mapping is only functional when specifying a bitmap by its resource ID (a number). Resource strings are not mapped.

Remarks

None.

GWToolbarItem2

Allows you to create a toolbar button with a drop-down menu. Valid for GroupWise 2012.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
HMenu	R/W	The windows HMENU associated with this toolbar button.
Menu	R/O	The menu associated with this toolbar button.

Methods

```
BuildToolbarButtonMenu( string bstrContext,  GWMenu DispMenu)
```

Creates an empty toolbar drop-down menu. The caller can then add items and submenus as necessary. This method must be called before `GWToolbarItem.SetBitmap` to have any effect.

Remarks

None.

GWToolbarItems

Enumerates the items on a toolbar.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Count	R/O	Long.
_NewEnum	R/O	Enumeration object. Implements IEnumVARIANT. For Windows only.

Methods

```
GWToolbarItem Add( String Caption, GWCommand item [Variant AddBefore] )
```

AddBefore refers to the index (ordinal) into the collection.

```
GWToolbarItem Item( Variant item )
```

Returns GWToolbarItem. The item parameter refers to an index (ordinal).

```
GWToolbarItem FindByID( Variant ID )
```

Returns the GWToolbarItem object with the command ID assigned to a toolbar button (Number). This method can be used by a C3PO to override the behavior of other C3PO commands. That is, a GWCommand object can be obtained, aggregated with a new GWCommand object, and the result assigned back to the GWToolbarItem.GWCommand property. Items on the menu that were not constructed by the C3PO return NULL.

Remarks

While a toolbar is being loaded, the client calls CommandBlock::CreateCommandAggregate and invokes GWToolbarItems::Add.

Unlike menus, the Item and AddBefore parameters allow you to specify a persistent ID.

IconFactory

Retrieves icons that represent the state of C3PO record types.

Properties

None.

Methods

```
GetIcons( String ObjClass, String *pIconFile, long *plUnOpenIcon, long
*plOpenIcon )
```

Remarks

The plUnOpenIcon and plOpenIcon parameters are icon index values within the module (.EXE or .DLL) named by the pIconFile parameter. Returning a value of -1 indicates that a default icon should be used. Any other negative index value indicates a resource ID by taking the absolute value of the number. The IconFactory object does not support inheritance. There is no mechanism for two C3PO COM Servers to offer icons for the same record. The C3PO system simply uses the first C3PO it finds.

The icons retrieved are presumed to contain multiple image sizes. The appropriate size is used by GroupWise as needed.

MessageBlock

Aggregates the manipulation of C3PO message instances.

Properties

None.

Methods

```
GetIcons( String ObjClass, String *pIconFile, long *plUnOpenIcon, long
*plOpenIcon )
```

Remarks

None.

PresenceFactory

Used to manage presence information displayed in the GroupWise client. Valid for GroupWise 2012.

Properties

None.

Methods

```
WantPresence(
   GW_PRESENCE_CONTEXT GWContext,
   DIGWAddressBookEntry7 pContactDisp,
   long ContactOrPhoneNumberType,
   VARIANT_BOOL *pbCanGivePresence
   )
```

Asks the C3PO if it wants to provide presence for the given contact, in the given context. ContactOrPhoneNumberType is used if the user is clicking on a particular phone number or IM address, otherwise it is 0. The return value is set to VARIANT_TRUE if the object can give presence.

Represents one of the following constants:

eGW_CONTACT_ADDRESS_CARD_VIEW	Presence is being requested by an item in the address card view.
eGW_CONTACT_ADDRESS_CARD_TOOLTIP	Presence is being requested by an item in a tool tip.
eGW_CONTACT_ITEM_LIST	Presence is being requested by an item in a contact list.
eGW_MAIL_ITEM_LIST	Presence is being requested in the mail item list.
eGW_CONTACT_MAIL_ITEM_VIEWER	Presence is being requested while viewing a mail item.
eGW_CONTACT_CONTACT_ITEM_VIEWER	Presence is being requested while viewing a contact.
eGW_CONTACT_HTML_PAGE_EMBEDDED	Presence is being requested from within an HTML contact view.

```
BuildPresence(
   GW_PRESENCE_CONTEXT GWContext,
   DIGWAddressBookEntry7 pContactDisp,
   long ContactOrPhoneNumberType,
   DIPresence2 **ppIDispPresence
   )
```

Requests that the C3PO return a Presence object given the same parameters as WantPresence.

Remarks

None.

Presence2

Defines an instance of an object containing presence information for a particular contact. Valid for GroupWise 2012.

Properties

The following table lists the properties for this class, along with access and descriptions.

Property	Access	Description
Contact	R/O	DIGWAddressBookEntry7 Returns the contact associated with this presence object.

Methods

```
GetBitmap(
   string *bstrBitmapDllname,
   VARIANT *vResID,
   long *nImages
   )
```

Returns the DLL and resource ID of a bitmap that lists all of the possible presence states for this provider. nImages is the number of images in the bitmap.

```
GetPresence(
   GW_PRESENCE_CONTEXT GWContext,
   [string *bstrStatusString,
   GW_PRESENCE_TYPE *PredefinedPresence,
   long *nBitmapIndex
   )
```

Returns the presence given the context. bstrStatusString is a description of the state (offline, available, etc).

PredefinedPresence is the constant that best describes the user's presence, and nBitmapIndex is the index of the appropriate icon in the bitmap returned by GetBitmap.

The following presence type constants are returned:

eGW_PRS_AVAILABLE	The user is available.
eGW_PRS_AWAY	The user is away.
eGW_PRS_BUSY	The user is busy.
eGW_PRS_IDLE	The user is idle.
eGW_PRS_INVALID	Unknown presence state.
eGW_PRS_OFFLINE	The user is offline.
eGW_PRS_ON_THE_PHONE	The user is on the phone.
eGW_PRS_UNKNOWN	The user's status is unknown.

```
GetAvailableActions([out] GW_PRESENCE_PROVIDER_ACTIONS *pavailableActions)
```

Called to ask the presence object what actions it can perform on the given contact.

The following presence provider action constants are returned:

eGW_LEAVE_VOICE_MESSAGE	The provider can leave a voice message.
eGW_NONE	The provider can't do any actions.
eGW_PHONE_CALL	The provider can dial the user.
eGW_SEND_IM	The provider can send an IM to the user.
eGW_SEND_SMS_MESSAGE	The provider can send a SMS message.
eGW_SEND_EMAIL	The provider can send an e-mail.
eGW_SEND_FILE	The provider can send a file.
eGW_VIDEO_CALL	The provider can create a video chat/call with the user.

```
ButtonLClicked(long hWnd)
```

Called if the user left clicks on the presence icon.

```
ButtonRClicked(long hWnd)
```

Called if the user right clicks on the presence icon.

```
DisablePresence();
```

Called when the user is done using the presence for this user (switched views, closed a contact, etc).

```
GetSubmenu(
   GW_PRESENCE_PROVIDER_ACTIONS action,
   VARIANT_BOOL bOnlyProvider,
   long *phMenu
   )
```

Called when the client is aggregating several presence providers using the “'little man” icon.

Action is set when the client is requesting providers for a specific action. Otherwise, it is 0.

bOnlyProvider is set to VARIANT_TRUE if this is the only provider that indicated it can handle the action.

phMenu is expected to return a windows HMENU, which will be combined with other menus for the right click action.

```
MenuClicked(long MenuId,
   string MenuText
   )
```

Called if the user clicks on any menu returned by GetSubMenu.

MenuId is the Menu id that was clicked.

MenuText is the text that was displayed in the menu.

Remarks

None.

Available Contexts

The C3POManager will accept any context string that adheres to the following form:

```
Indicator.Key[.Subey[.Subkey[.etc]]]
```

There must be at least one dot (.) separating the Indicator from a key. When traversing such a context string, the C3POManager will pick up C3POServers registered anywhere in the key up to (but not including) the Indicator.

All application contexts produced by Open Text begin with an indicator key of GW. Contexts that are specific to Open Text applications insert a subkey for the application. For example, all contexts specific to the GroupWise client begin with GW.CLIENT.

This space is then further partitioned for particular uses:

GW.CLIENT.WINDOW.ATTACHVIEWER	Attachment viewer.
GW.CLIENT.WINDOW.BROWSER	Browser window.
GW.CLIENT.WINDOW.CALENDAR	All calendar views.
GW.CLIENT.WINDOW.DOCUMENTLIST	Document list window.
GW.CLIENT.WINDOW.FINDRESULTS	Query results window.
GW.CLIENT.WINDOW.PROPERTIES	Properties window
GW.CLIENT.WINDOW.QUICKVIEWER	Quickviewer window

Contexts not specific to clients do not include a client indication as part of the context string:

GW.MESSAGE	All message types and their windows.
GW.MESSAGE.APPOINTMENT[.xx]	Appointments and their windows.
GW.MESSAGE.DOCUMENTREFERENCE	Document references.
GW.MESSAGE.MAIL[.xx]	Mail messages and their windows.
GW.MESSAGE.MAIL.Internet	Internet mail
GW.MESSAGE.MAIL.NGW.DISCUSS	Discussions and their windows.
GW.MESSAGE.NOTE[.xx]	Notes and their windows.
GW.MESSAGE.PHONE[.xx]	Phone messages and their windows.
GW.MESSAGE.TASK[.xx]	Tasks and their windows.

Pre-Built Command ID Encoding

GWCommand Objects and Events are identified through the use of a "persistent ID" that appears as a property of the command object. While C3PO software that defines new commands assign their own IDs, GroupWise ships with functions that have predefined IDs which adhere to the following convention:

GW#C#XXX	Identifies a GroupWise command. The XXX is replaced with a number or string that represents a command. Numbers are replaced for XXX when the command represents a literal client token; the number is the token ID. Strings are replaced for XXX when new or abstract commands are represented, such as a compose event.
GW#E#XXX	Identifies a GroupWise event. The XXX is replaced with a number that represents an event ID number.

A complete ID list of predefined IDs appears in the GroupWise SDK header files.

Predefined GroupWise Client Identifiers

Identifiers and the GroupWise SDK

The constants listed in this section are simply labels that describe the PersistentID values associated with predefined GroupWise commands. Listed below are the symbolic labels for the predefined commands (effectively #define equivalents) along with their descriptions.

C3PO Data Type Related Identifiers

For C3PO data types (new record types), there is a set of commands (such as Open, Accept, Forward, Delete) that are implicit to the record definitions. When registering a C3PO as supporting a C3PO data type, it is implicitly defined that the C3PO will support (or be asked at runtime if it wants to support) these commands. The commands are:

Event	Description
eGW_CMD_ACCEPT	Accept an instance of the item type.
eGW_CMD_ARCHIVE	Archive an instance of the item type.
eGW_CMD_COMPLETE	Mark an instance of the item type as complete.
eGW_CMD_COMPOSE	Create a new item type.
eGW_CMD_DECLINE	Decline an instance of the item type.
eGW_CMD_DELEGATE	Delegate an instance of the item type.
eGW_CMD_DELETE	Delete an instance of the item type.

Event	Description
eGW_CMD_DIAL	Dial a phone number. Valid for GroupWise 2012.
eGW_CMD_DOC_CHECKIN	Display the check-in time of a document.
eGW_CMD_DOC_CHECKOUT	Display the check-out time of a document.
eGW_CMD_DOC_RESETINUSE	Display when attempting to reset a document's in-use flag.
eGW_CMD_FORWARD	Forward an instance of the item type.
eGW_CMD_OPEN	Open an instance of the item type.
eGW_CMD_OPENFILEATTACH	Open an instance of the item type (attachment).
eGW_CMD_PRINT	Print an instance of the item type.
eGW_CMD_PROPERTIES	Display the properties for a GroupWise object.
eGW_CMD_REPLY	Reply to an instance of the item type.
eGW_CMD_RESEND	Resend an instance of the item type that is currently in the GroupWise Out Box.
eGW_CMD_SAVE	Save an instance of the item type back to the database.
eGW_CMD_SAVEAS	Save an instance to an external file.
eGW_CMD_SAVEATTACHAS	Save an instance of the item type back to the data base (attachment).
eGW_CMD_SEND	Send an instance of the item type.
eGW_CMD_SETALARMS	Set alarms on a GroupWise object.
eGW_CMD_UNDELETE	Remove an instance of the item type from the GroupWise trash.
eGW_CMD_VIEW	View an instance of the item type.
eGW_CMD_VIEWATTACH	View an instance of the item type (attachment).

Events

The GroupWise client issues several events through the EventMonitor:

Event	Description
eGW_EVT_DELIVERY	Indicates that an item has been delivered to the GroupWise mailbox. A C3PO will typically invoke the GWClientState::SetupDeliveryFolder method and query the corresponding folder when this event is broadcast.
eGW_EVT_OVERFLOW	This notification indicates that too much information has been returned at the same time. When this occurs, you should query the corresponding folder to determine if the information you requested is there.
eGW_EVT_READY	This notification occurs immediately after GroupWise is initialized. The following services are running at the time this method is called: DDE server; OLE objects registered and placed in the COM running object table.
eGW_EVT_SHUTDOWN	A notification to the C3POServer that shutdown is occurring. System services are still available when this notification is received. Immediately after returning from this method call, the C3PO should assume that the calling client has shut down. If the calling client is the last client to use the C3POManager, calls to the Manager will fail after this call has returned.

Sample Applications

There are several C3PO sample applications designed to show you how to build a usable C3PO. The code provided with these samples is intended to benefit you as you build your own C3PO software. While these samples have passed a rigorous testing process, they are not necessarily error-free.

"C3POPower" on page 107
"Customer Tracking (C++ and Delphi)" on page 108
"Export" on page 110
"GW Notes (Delphi)" on page 110
"Skeleton" on page 111
"Tip of the Day" on page 111

C3POPower

A test application that uses a cross-section of Object API objects and tokens to show various ways the GroupWise 8 and later clients can be modified to suit developer and user needs in customizing GroupWise.

This project is not meant for production; it is offered to show how objects and tokens can be used together to perform a task.

The C3POPower sample application uses the following C3PO objects:

C3POManager
ClientState
GWMenu
GWMenuItem
GWMenuItems

The C3POPower sample application uses the following Object API objects:

Attachment
Attachments
Document
DocumentVersion
Folder
Message
MessageList

The C3POPower sample application uses the following tokens:

Cancel( )
DateAbsoluteGoTo(’+sDay+’;’+sMonth+’;’+sYear+’)
Delete( )
EnvPrefSavePath( )
FilterCreate(Empty!;0)
FilterDelete(’+Filter+’)
FilterGroupBegin(’+Filter+’;317)
FilterGroupEnd(’+Filter+’)
FilterGroupMarker(’+Filter+’;316)
FilterReset( )
FilterSetSource(’+Filter+’;6;1)
FilterSetText(’+Filter+’;1;"’ + FromString + ’";165)
ItemAttachmentAdd("X00";115;"’ + MsgID + ’")
ItemMessageIDFromView( )
ItemReply(Sender!;Yes!)
ItemSaveMessage("’+Filename+’";"’+t+’"; 269)
ItemSaveViewDlg( )
NewMail( )
EnvUserID( )
QueryExecute(;’+Filter+’;FullUserID:"’+RetString+’")
TextSetSubject("Fwd: ")
ViewHideMenu(Hide!)
ViewOpenFile("Ussm_ap.vew")

Customer Tracking (C++ and Delphi)

Lets you track your customers by building a Customer Tracking folder which contains: 1) a company profile; 2) a company contact; and 3) an action profile.

The Object API was used to create the Customer Tracking folder and the C3PO allows this application to interface with GroupWise.

This SDK contains three versions of the Customer Tracking sample application:

- A DLL written in C++
- An EXE written in Delphi
- A DLL written in Delphi

The Delphi .EXE is provided to show you an example of an out-of-process server, while the Delphi .DLL is an example of an in-process server. Registration instructions for all three versions are provided below.

Note that if you run the .DLL applications from a location other than the SDK, you must edit the .REG file for the application to reflect the correct path to the .DLL. For example, if you copy this program with its file structure to your C: drive, you should find the line that ends "...\InprocServer32" in the .REG file, then change the D: in the next line to a C:.

To register the Customer Tracking C3PO in GroupWise,

1. Open the CTS.REG file and change the path to the C3PO.DLL file to reflect its proper location.
2. Run CTS.REG.
3. Copy C3PO.DLL to the WINDOWS directory on your hard drive. This will enable CommandFactory::CustomizeToolbar and IconFactory::GetIcons to find the bitmap and icons in C3PO.DLL.

To register the Customer Tracking C3PO in C++ (C3PO.DLL),

1. Copy C3PO.DLL to the WINDOWS directory on your hard drive.
2. Run CTS.REG file to register the C3PO.
3. Run C3PO.DLL to execute the C3PO.

To register the Customer Tracking C3PO in Delphi (CUSTRACK.EXE),

1. Copy ICONS.DLL to the WINDOWS directory on your hard drive.
2. Run C3POAUTO.REG to register the C3PO.
3. The first time you run CUSTRACK.EXE, include the command line switch /regserver. (For example, CUSTRACK.EXE /regserver)
4. Copy ICON.DLL to the WINDOWS directory on your hard drive.
5. Succeeding executions do not require the /regserver switch.

To register the Customer Tracking C3PO in Delphi (CTRACK.DLL),

1. Copy ICONS.DLL to the WINDOWS directory on your hard drive.
2. Run CTRACK.REG
3. Run CTRACK.DLL

The Customer Tracking sample application uses the following C3PO objects:

C3POManager
C3POServer
ClientState
EventMonitor
IconFactory
GWMenu
GWMenuAction
GWMenuItem
GWMenuItems
GWMenuSeparator

GWToolbar
GWToolbarItem
GWToolbarItems

The Customer Tracking sample application uses the following Object API objects:

Account
Application
Field
FieldDefinition
FieldDefinitions
Fields
Folder
Folders
FormattedText
Message
Messages

Export

Allows a user to export a message or group of selected messages to a file on disk, and then optionally load that file into a viewer of your choice for quick browsing.

After you run setup, the next time you launch GroupWise an Export... option will appear on the context menu of the message window. Selecting this option will bring up a dialog box in which you should specify a filename. Clicking the Preferences button will provide you with a group of options from which you can specify what message data to export as well as additional options for such things as moving the selected items to a folder when the export is complete, or exporting only unread items. Selecting the Export Attachments option will cause all first-level attachments to be saved in the same directory as the specified filename.

Please note that the default viewer is Notepad, which does not support word wrap by default. You can either specify a no robust viewer or choose Edit|WordWrap to turn the option on.

GW Notes (Delphi)

Implements GroupWise "sticky" notes.

You can use GW Notes to annotate GroupWise messages or as a standalone application to create personal notes separate fromGroupWise. It allows users to drag a GroupWise note to the desktop to be a reminder. When you click the desktop Note icon, it opens the GroupWise Note view if GroupWise is already loaded. If GroupWise is not loaded, it loads GroupWise and then displays the note.

This sample application was written using both the GroupWise Object API and C3PO software with Delphi 2.0. To run this application, you must have Delphi 2.0 running on your system.

This sample application illustrates: 1) how to use an in-process C3PO server to communicate with a C3P0 server running as an out-of-process server; and 2) how to use the GroupWise Token Commander to pass macros to GroupWise.

To create and register the GW Notes C3PO,

1. In Delphi, compile GWNOTES.DPR and GWNTS1.DPR.
2. Run REGISTER path to GWNTS1.DLL.

To use GW Notes within GroupWise, choose a GW Notes menu option (such as New Note).

To use GW Notes independently of GroupWise, execute GWNOTES.EXE from Delphi or from a DOS prompt.

To unregister GWNTS1.DLL, type REGISTER path to WNTS1.DLL /u .

Skeleton

A Visual Basic project with skeleton classes for all of the corresponding C3PO objects. Programmers can add each class individually to their project or load the C3PO project and add functionality to it.

Tip of the Day

Provides a new GroupWise tip the first time a user opens the GroupWise client each day.