Inside UniChat 2™

1. Source Files

Compiler: MS Visual C++ 4.0 or above
Number of Lines = 5,256 (UC2Ani.dll) + 19,094 (UC2.exe) + 3,962 (MapEd.exe) = **28,312** lines

VC++ Project	Description	Module
UC2Ani	UniChat Animation Library	UC2Ani.dll
UC2	UniChat 2 Client System	UC2.exe
MapEd	Map Editor	MapEd.exe
UDSGen	UniChat Data Source Generator	UDSGen.exe
MonTool	Monitoring Tool for counting current members	MonTool.exe

Directory Structure in the UniChat 2 Source CD-ROM

Directory Name	Description
UC2	Root
+AsyncMoniker	Asynchronous Moniker (for files download) sample program
+BMC	Bitmap file manipulation sample code for CDIB (with separate palette file)
--Debug	DEBUG: intermediate files for the compiler
--Doc	Some documents
+GenSDI	Generic SDI: Visual C++ compiler's generated template for an SDI project
--Hlp	Help project files for UniChat 2
+ImageSrc	Image Sources for UniChat 2
+I.Eng	Resource Definition files for English version
--I.Kor	Resource Definition files for Korean version
+I.Mall	Resource Definition files for Mall version
+MapEd	Map Editor Project
+Data	Data files: The default directory for data in UniChat 2 debugging environment
+MonTool	Monitoring Tool project
+Progress	Test project for data downloading with progress control view
--Release	RELEASE: intermediate files for the compiler
--Res	RES: Resource files for UniChat 2
+Setup	Setup project for UniChat 2
--System	Redistributable files for UniChat 2
+Test	Test project
+UC2Ani	Animation library project
+UDSGen	UniChat Data Source file Generator Project
+WBTest	Web Browser Test Project

Project UC2Ani which generates UC2Ani.dll

Class	Base Class	Description	.H	.CPP
CDIBPal	: CPalette	Palette Manager	35	386
CDIB	: CObject	Device Indepent Bitmap manipulation	174	1,548
CSprite	: CDIB	Bitmap Sprite	133	329
CPhasedSprite	: CSprite	Phased Sprite Animation	97	397
COSBView	: CScrollView	Off-Screen Buffered View	87	421
CSpriteList	: CObList	Sprite List structure	38	173
CSpriteNotifyObj	: CObject	Notification Object for Csprite	38	-
CSpriteListNotifyObj	: CSpriteNotifyObj	Notification Object for CSpriteList	36	65
CBubble	: CObject	Bubble Drawing	64	287
CBubbleList	: CObList	Bubble List structure	38	149
CBubbleNotifyObj	: CObject	Notification Object for CBubble	36	-
CBubbleListNotifyObj	: CBubbleNotifyObj	Notification Object for CBubbleList	34	65
CMCIObject	: CObject	Media Control Interface	40	213
CSound	: CObject	Simple Sound Player using MCI	37	131
CPSButton	: Cbutton	Phased Sprite Button	62	143
UniChat Animation Library			949	4,307
				5,256

Project MapEd which generates MapEd.exe

Class	Base Class	Description	.H	.CPP
CAboutDlg	: CDialog	About Dialog	-	-
CActor	: CPhasedSprite	Actor	-	-
CActorDesc		Actor Description	-	-
CBehavior		Behavior of actors	-	-
CCloseDlg	: CDialog	Close Dialog	-	-
CFlatToolBar	: CToolBar	Toolbar for flat buttons	32	245
CGetIntDlg	: CDialog	Input Dialog for Integer	49	60
CGetTextDlg	: CDialog	Input Dialog for Text	48	53
CMainFrame	: CFrameWnd	MainFrame Window for MapEditor	70	239
CMapEdApp	: CWinApp	Application Window for MapEditor	66	201
CMapEdDoc	: CDocument	Document Class for MapEditor	138	589
CMapEdView	: COSBView	View Class for MapEditor	125	1,054
CMapEnvDlg	: CDialog	Input Dialog for Map Environment variable	60	78
CMapListView	: CFormView	View for the list of tiles and sprites	113	489
CMemberInfo		Member Info	89	164
CParser		Line by line parser for the text file	-	-
CResMan	: CObject	Resource Manager	-	-
CStage	: CObject	Stage	-	-
CTextFileBuffer	: CObject	Memory management for text files	-	-
CTileMap	: CObject	Tile and map management	-	-
Map Editor			790	3,172
				3,962

Project UC2 which generates UC2.exe

Class	Base Class	Description	.H	.CPP
CActor	: CPhasedSprite	Actor	91	406
CActorDesc		Actor Description	31	122
CBaseChannel	: CObject	Base Channel Service	101	467
CBaseSocket	: CObject	Base Socket Service	112	621
CBehavior		Behavior of actors	94	138
CBindStatusCallback	: IBindStatusCallback	Callback class for downloading	63	108
CCloseDlg	: CDialog	Close Dialog	71	280
CDownInfo		Download info	25	65
CDownload		Download class using IMoniker	17	308
CEditHistory	: CEdit	Edit control for History Panel	51	–
CEditSend	: CEdit	Edit control for Sending text	36	–
CInputPassword	: CDialog	Input Password Dialog	47	44
CLoginDlg	: CDialog	Login Dialog	144	920
CMainFrame	: CFrameWnd	MainFrame Window for UniChat	127	860
CMemberInfo		Member Info	89	164
CMemberListDlg	: CDialog	Member List Dialog	85	406
CParser		Line by line parser for the text file	135	422
CPPActor	: CPropertyPage	Property Page for actor	76	317
CPPChannel	: CPropertyPage	Property Page for Channel List	89	493
CPPCreateChannel	: CPropertyPage	Property Page for Create Channel	63	166
CPPMemberInfo1	: CPropertyPage	Property Page 1 for Member Info	88	94
CPPMemberInfo2	: CPropertyPage	Property Page 2 for Member Info	–	–
CProgressDlg	: CDialog	Dialog for downloading	82	629
CPSFrame	: CMiniFrameWnd	Frame Window for member property page	51	103
CPSJoinChannel	: CPropertySheet	Property Sheet for Joining Channel	106	375
CPSMemberInfo	: CPropertySheet	Property Sheet for Member Info	52	52
CResMan	: CObject	Resource Manager	156	1,252
CSplashWnd	: CWnd	Splash Window for 256 color image	66	237
CStage	: CObject	Stage	141	1,099
CTextFileBuffer	: CObject	Memory management for text files	136	423
CTileMap	: CObject	Tile and map management	156	1,197
CUC2App	: CWndApp	UniChat 2 Application Window	96	500
CUC2Channel	: CBaseChannel	Channel Service for UniChat	35	148
CUC2Doc	: CDocument	Document class for UniChat	186	1,575
CUC2History	: CDialogBar	History Panel	66	74
CUC2Panel	: CDialogBar	Control Panel	75	271
CUC2Socket	: CBaseSocket	Socket Service for UniChat	65	384
CUC2View	: COSBView	View class for UniChat	127	893
CWhisperDlg	: CDialog	Whisper Dialog	48	46
		UniChat 2 Messages Definition	156	
UniChat 2 Client			3,435	15,659
				19,094

2. Data Files

File Type	Description	Example
RIT	Resource Information Table	U2Res000.rit
SIT	Stage Information Table	0000csin.sit, …
UDS	UniChat Data Source (like a DB)	a00.uds, …

RIT and SIT files are distributed after LZ compression. UniChat client program can cleverly load these files whether they are compressed or not.

RIT – Resource Information Table

RIT contains all the definitions for the UniChat resources. UniChat resources are mainly tiles, static sprites, animated sprites, avatars and animation descriptions. For example, every static sprite needs an information for the bottom center position in the image besides the image file itself.

The client system does not store each file name for the sprite but it sequentially generates serial numbers from 0 and use this number as an identifier to the sprite image.

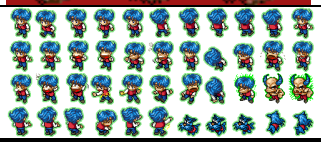
For images, we have the following types:

Object Type	Sample File	Naming and Scripting

Tile
SPRITE_TILE
SPRITE_WALL
		**t**NAMEnnn.bmp
		t00|tcity001=(1,7); // t00 is a unichat data source file
		// cells are arranged in 1 column and 7 rows

Static Sprite
SPRITE_STATIC
		**c**NAMEnnn.bmp
		cs00|ctree009=(29,83);
		// center of the bottom (earth position) – in pixel units
		// the crossing point at the left image indicates this position
		(In Map Editor, this position is shown in this way)

Animated Sprite
SPRITE_PHASED
		**i**NAMEnnn.bmp
		cw00|ircha001=(3,1),(20,36);
		// cell dimension, center of earth position

Avatar
SPRITE_ACTOR
		**a**NAMEnnn.bmp
		a00|aman_001=(11,4),(23,45);
		// cell dimension, center of earth position

- Every image sets its first pixel, the left top pixel of the image, as a transparent color.
- Static sprites and animated sprites can have some linked URLs in the script.

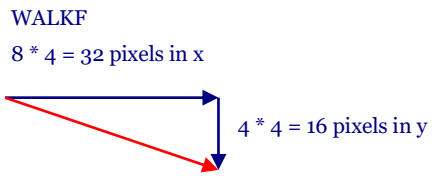
You can assign 3 types of animations for each sprite in Map Editor. CSprite::GetAniType() return the associated animation type of the sprite. In CPhasedSprite::HeartBeat(…), you can see the code for these animations.

Animation Type	Description
SPRITE_ANI_REPEAT	Cell animation with assigned interval
SPRITE_ANI_FADE	Animation effect with fade-in and fade-out of one cell
SPRITE_ANI_RANDOM	REPEAT animation but with random interval between cycles
SPRITE_ANI_ACTOR	Actor keeps animation data in CActorDesc and CBehavior

```
; "u2res00.rit"
;=========================================
; Resource Info Specification
; RIT: Resource Information Table
; Specification written by Soomin Kim
; Feb 1998, Samsung SDS
;=========================================
; filename=[(col,row)][,(cz.x,cz.y)][,nickname];
;   filename = resource type flag(1) + name(4) + serial(3);
;   resource type flag = t(TILE), s(STATIC), i(ANI), a(AVATAR), w(WAVE), m(MIDI)
;   (col,row) = # of cells in (column, row); for STATIC this attribute is omitted
;   (cz.x,cz.y) = center of z-order in (x, y) pixel position from the left-top
;   nickname = nickname for actor
#VERSION=0.99;
// Think of tile as a PhasedSprite
#TIL=        // TILE
{
t00|tcity001=(1,7);        // "tcity001.bmp" in "t00.uds" has 7 cells in one row.
t00|tcity002=(1,5);
t00|tcosm001=(3,7);
…
}
#STT=        // STATIC Sprites – These files comprise of only one cell.
{            // bottom center position (cz.x, cz.y) in pixel
ca00|cadve001=(10,78),http://www.unitel.co.kr/;
ca00|cadve006=(23,115),http://www.macdonald.com/;
ca00|cbill001=(32,98);
…
}
#ANI=        // ANIMATED Sprites
{
cw00|iadve001=(3,1),(110,0),http://a.b.c/unichat/|http://a.b.c/|http://www.naver.com/;
cw00|iadve002=(4,1),(1,6),http://a.b.c/|http://www.samsung.co.kr/|http://a.b.c/|http://www.naver.com/;
cw00|iadve003=(4,1),(1,6),http://a.b.c/|http://a.b.c/|http://a.b.c/unichat/|http://www.naver.com/;
cw00|iarro001=(5,1),(5,0);
…
}
#AVT=        // AVATAR
{
a00|aman_001=(11,4),(23,45);
a00|aman_002=(11,4),(23,45);
a00|aman_003=(11,4),(23,45);
…
```

```
}
#WAV=       // Wave files
{
sagry000;   // sagry000.wav
schng000;
…
}
#MID=       // MIDI files
{
mjazz000;   // mjazz000.mid
mjazz001;
…
}
#STAGE=   // User-creatable stages
{
0000csin;   // 0000csin.sit
0001ctrm;
…
}
#SERVERIP=              // Server IP Addresses
{
203.241.132.83;         // LAN
88.1.26.2;              // MODEM
127.0.0.1;
ComicSrv1.Microsoft.Com;
vchat1.microsoft.com;
}

; ACTOR SECTION
; (*i): - change orientation, * 50%, / 25%, | 12%, ~ 0%, ^ don't USE_COLORKEY
; #ACTOR=id,nickname,nMSPT;      // nMSPT: milliseconds per tick
; {
;               // nMSPT is Milliseconds per Tick
;               // Behaviors
;               0=nRepeatCount,(cell index(, delay ticks(, displacement_x, displacement_y(, sound id))))...;
;               // nRepeatCount = 0 for infinite loop, - value for pendulum movement
;               // delay ticks;  5 = fast; 10 = moderate; 15 = slow
; }

#ACTOR=a00|aman_001,man001,40;              // This first actor definition will be used for the default.
{
// Behaviors
STANDF=1,(0,2);         // Stand forward
STANDB=1,(9,2);         // Stand backward
STANDINGF=1,(|0,2,0,0,schng002)(/0,2)(*0,2)(#0,2)(0,2);    // Appearing in forward stance
STANDINGB=1,(|9,2,0,0,schng002)(/9,2)(*9,2)(#9,2)(9,2);    // sound id is a predefined wave file in RIT
MORPHF=1,(39,2);        // Morphed image
MORPHB=1,(42,2);
```

```
MORPHINGF=1,(|39,2,0,0,schng000)(/39,2)(*39,2)(#39,2)(39,2);          // Morphing sequence
MORPHINGB=1,(|42,2,0,0,schng001)(/42,2)(*42,2)(#42,2)(42,2);
DOZEF=0,(*21,10)(*22,10);
DOZEB=0,(*33,10)(*34,10);
// Movements
WALKF=1,(1,5,8,4,sstep000)(2,5,8,4)(3,5,8,4)(4,5,8,4);
WALKB=1,(5,5,8,4,sstep000)(6,5,8,4)(7,5,8,4)(8,5,8,4);
UPF=1,(1,3)(2,3)(3,3)(4,3,0,0,sjpup000);          // Jump up
UPB=1,(5,3)(6,3)(7,3)(8,3,0,0,sjpup001);
DOWNF=1,(1,3)(2,3)(3,3)(4,3,0,0,sjpdn000);
DOWNB=1,(5,3)(6,3)(7,3)(8,3,0,0,sjpdn001);
MORPHWALKF=1,(40,5,8,4,sturn000)(41,5,8,4)(40,5,8,4)(41,5,8,4);
MORPHWALKB=1,(42,5,8,4,sturn001)(43,5,8,4)(42,5,8,4)(43,5,8,4);
// Gesture Commands
CHAT=3,(10)(11)(12);
ENTER=1,(|0,3,0,0,sstep000)(/0,3)(*0,3)(#0,3)(0,1);
EXIT=1,(0,3,0,0,sstep000)(#0,3)(*0,3)(/0,3)(|0,3);
SMILE=1,(13,0,0,stemp000)(14)(13)(14)(13)(14);
MAD=1,(15,5,0,0,sagry000)(16)(15)(16)(15)(16);
HELLO=1,(17,10)(18);
CRY=1,(19,5,0,0,scrys000)(20)(19)(20)(19)(20);
SCRATCH=1,(23,3,0,0,stemp001)(24,2)(23,3)(24,2);
PICK=1,(29,10,0,0,spick000);
SPECIAL=1,(30,5,0,0,stemp000)(31)(32)(*32)(32)(31)(32);
WIGGLEB=2,(33)(34);
PUNCHF=3,(25,5,0,0,shit_000)(26);
PUNCHB=3,(37,5,0,0,shit_002)(38);
BEATENF=1,(27)(28,2)(*28,3)(28,2)(27,3);
BEATENB=1,(35)(36,2)(*36,3)(36,2)(35,3);
;TURN180
;TURN360
}
#ACTOR=a00|aman_002,man002,20;          // Overload some definitions
{
MORPHWALKF=1,(40,5,8,4,sstep000)(41,5,8,4)(40,5,8,4)(41,5,8,4);
MORPHWALKB=1,(42,5,8,4,sstep000)(43,5,8,4)(42,5,8,4)(43,5,8,4);
SPECIAL=1,(30,5,0,0,stemp000)(31)(32)(31)(32);
}
#ACTOR=a00|aman_003,man003,50; // will use actor definition of the first one a00|aman_001 as a default
{
}
…
```



In the ACTOR section, actor's behaviors are clearly described by some kind of script. For each predefined behavior, we associate a sequence of cell animation and some parameters.

Avatar Image

In UniChat 2, every actor image has the same number (11*4=44) of cells in it and each cell is associated with the predefined behavior.

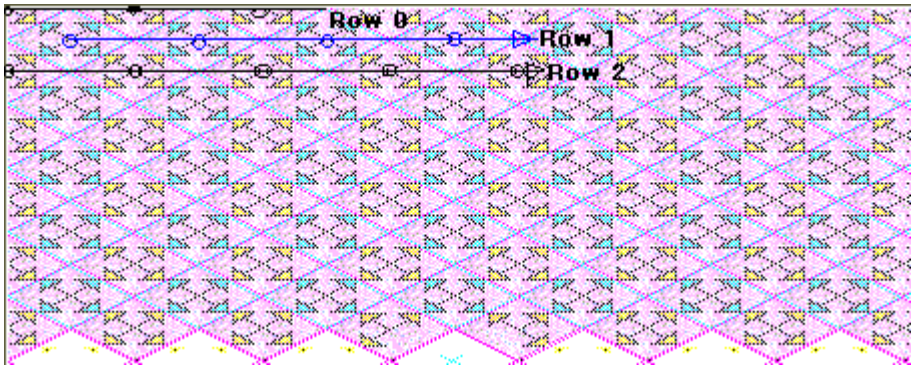
SIT – Stage Information Table

SIT is a file format for storing each background's map and objects on it. It has two sections, one for a list of sprite objects and other for the tile map. One background needs one SIT file. These files are automatically generated by Map Editor tool.

```
; Stage 0030mall.sit
#VERSION=1.01;
// Sprites data
// #STAGE=[room id] Title, Background Music Sequence;
#STAGE=[u2/0030mall]The M@ll-The Biggest Shopping Center on the Earth,5>9>1>4>2>8>;
{               // Cstage::Load()
// Resource ID = Cell ID, LeftTop position, Image Operation, Elevation, Sprite Type, nMSPT;
333=0,(134,-65),0,0,65535,0;
257=0,(454,-69);
335=0,(314,-43),256,0,65535,0;
257=0,(497,-49);
224=0,(616,4),256,0,4,0;
242=0,(106,-57);
257=0,(540,-27);
389=0,(317,-65),256,0,65535,0;
…
}               // 100 sprites.
// MAP data
#TILESIZE=(64,32);      // Tile Size in pixel
#SCREENSIZE=(640,368);          // Dimension
#ROW=(0); // CtileMap::LoadRow()
{
// -1 = Actor Elevation, Direction Attribute;          // for NULL tile (that has no associated tile image)
```

```
// Resource ID = Cell ID, Elevation, Image Operation, Actor Elevation, Sprite Type, nMSPT, Direction Attribute, Hyperlink;
1=3;
1=0;
3=1;
3=1,0,0,0,1,0,13;
1=3,0,0,0,1,0,12;
1=3,0,0,0,1,0,0;
1=3,0,0,0,1,0,0;
3=2,0,0,0,1,0,14;
3=2;
1=0,0,256,0,1,0,15;
0=2;
}
#ROW=(1);
{
…
-1=0,9;
}
…
#ROW=(23);
{
0=1,0,0,0,1,0,15,x:0020mall;          // This tile is an exit with link to “0020mall.sit”.
…
}
; 264 tiles.
```

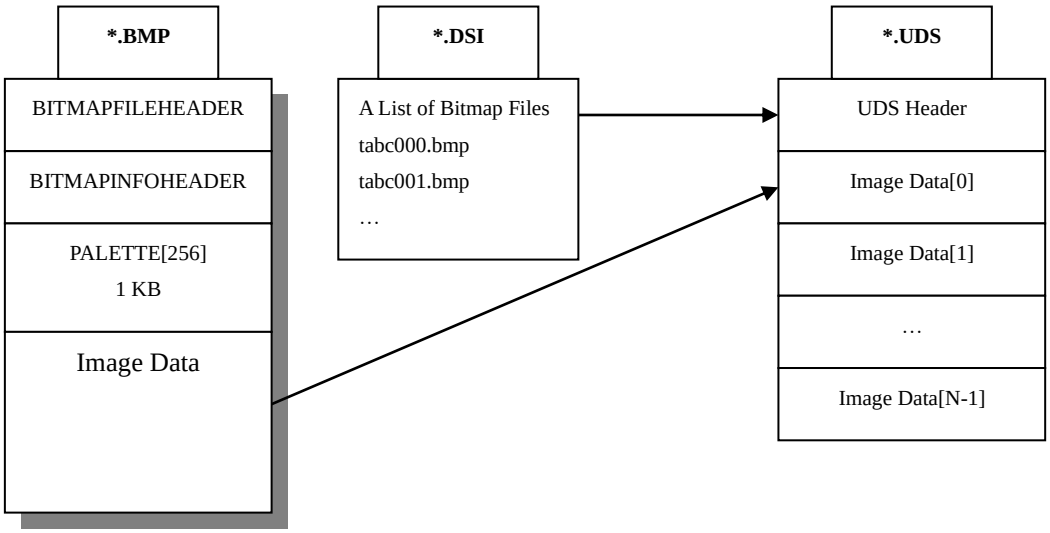
Tiles are managed by rows in the memory and graphically their sequence in the screen looks like this picture.



UDS – UniChat Data Source

UniChat 2 has more than 360 bitmap image files. In order to optimize disk usage by reducing the number of files, we combine multiple files into one data source file. This data source file has its own index table in it and actual bitmap images. And to reduce the file size UDS file does not store each palette table of the image. Since we are using one master palette with 256 colors, we don’t have to have the palette for each image redundantly.

UDSGen.exe is a tool to merge bitmap images into one file.



CDIB class in UC2Ani library knows how to read UDS file and load a specified bitmap image in it.

Naming rule for the resource location:

Data Source(*.uds) | Bitmap Filename(*.bmp)

For example,
CDIB* pDIB;
…
pDIB->Load("**a00** |**atree001**");
// In "a00.uds" file, find "atree001.bmp" image and load into memory

3. Data Model

The model of Classes in UniChat is an analogy of the stage and the actors on it. We can think of animation as a play of the actors on the stage.

Besides the basic classes in UC2Ani.dll, the animation related classes are the following:

Class	Description
CResMan	Resource Manager
CStage	Stage has control for creation of tiles, actors and sprites.
CTileMap	Loads tile data from .SIT and makes an image for the background
CBehavior	Behavior is a sequence of data for cell animations
CActorDesc	Actor Description class keeps a record for each predefined characters
CActor	Actor represents each user in UniChat

Animation Mechanism

Here is the animation mechanism. CStage::TickAll() function is a heart that pumps all the animations in the system. Behavior index and alarm tick in CPhasedSprite and CActor are some

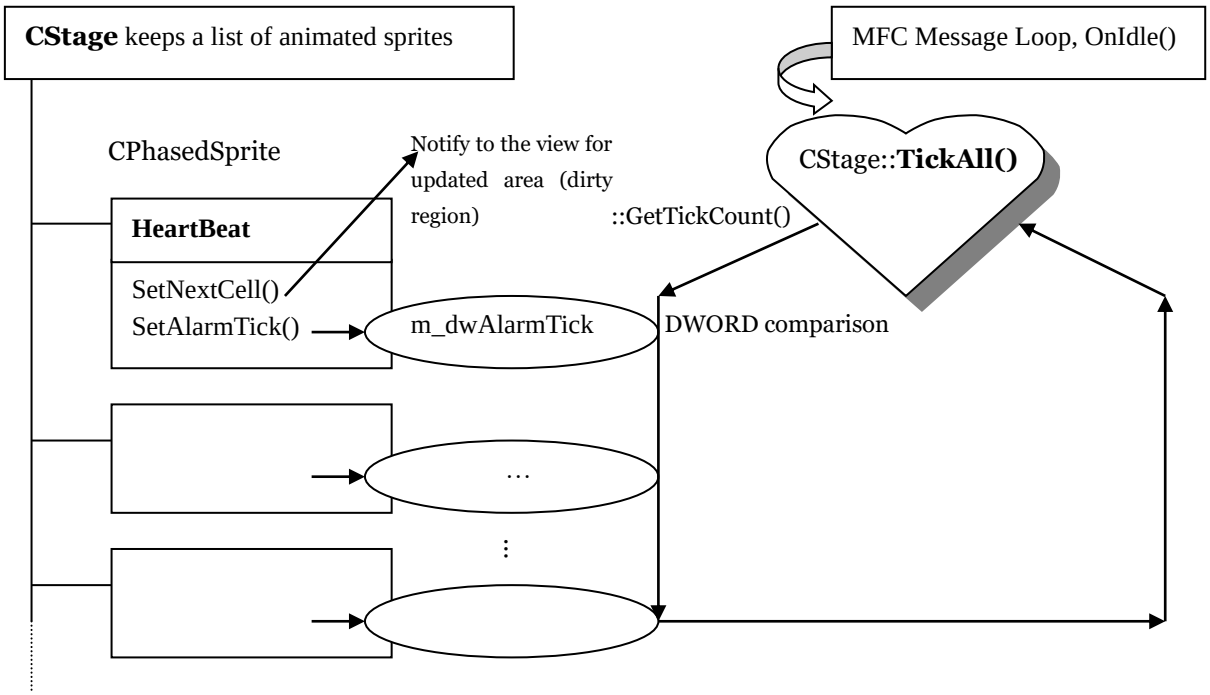
kind of switches. CActor is inherited from CPhasedSprite that has basic animation functions. Since CActor is for actors with various behaviors like walking, smiling, and crying, CActor involves more data and methods to deal with.

CPhasedSprite **CActor**

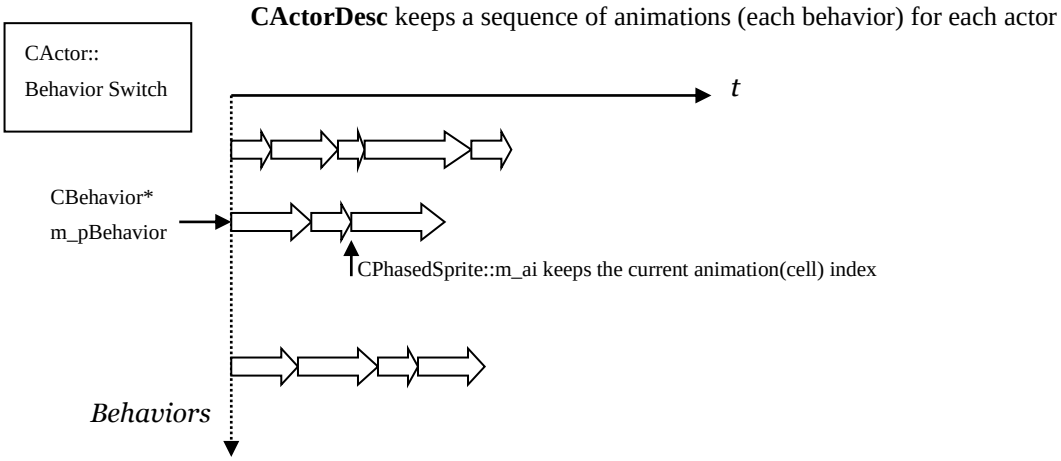
m_dwAlarmTick;
m_ai; // animation index
m_pBeh; // pointer to Behavior
CPhasedSprite::current cell index

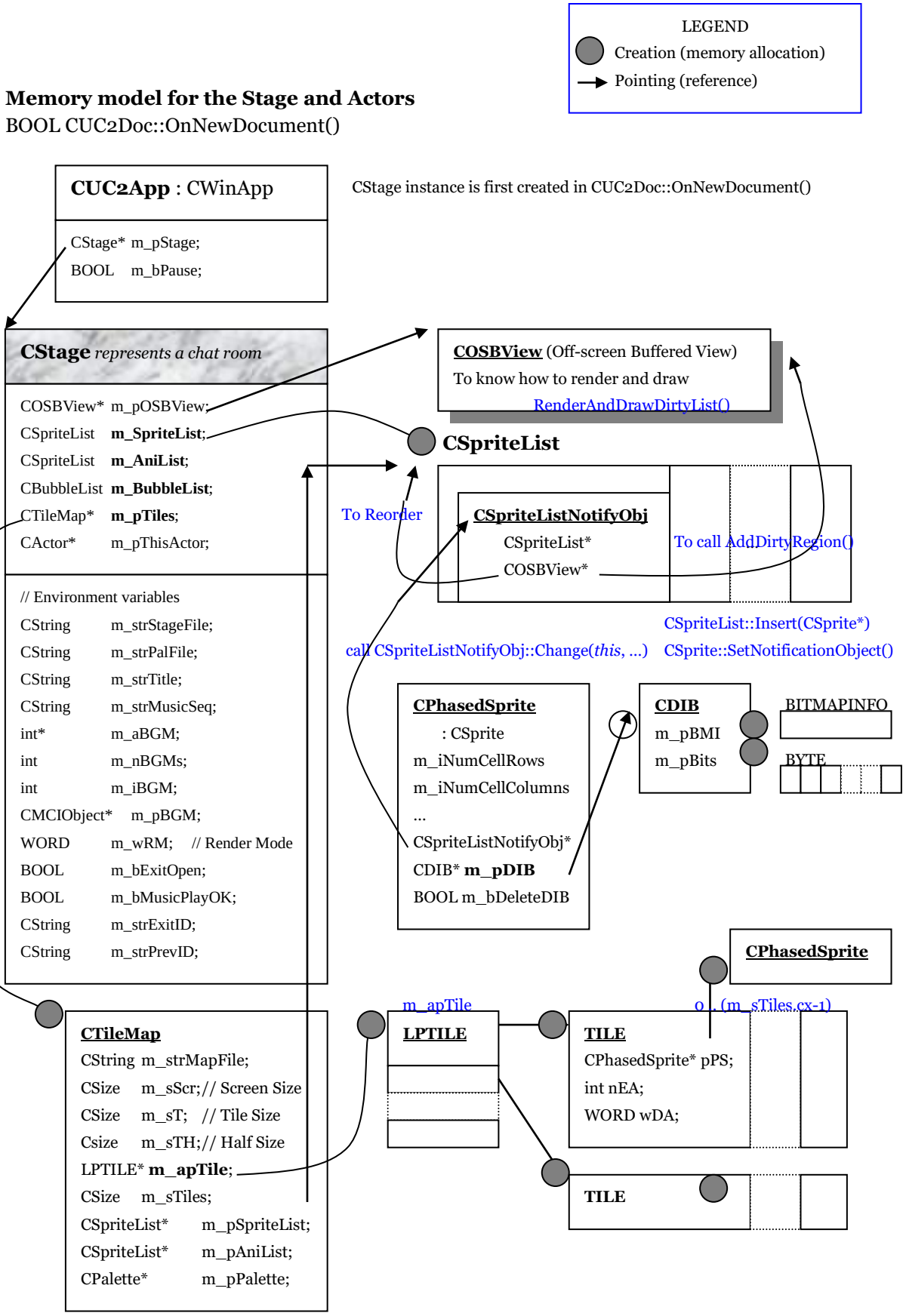
```
void CPhasedSprite::HeartBeat(DWORD dwCurTick)
{
…
    SetNextCell(); // Advance current cell index
    SetAlarmTick(dwCurTick + GetMSPT());    // Set alarm for the next animation
…
}

void CPhasedSprite::SetNextCell()
{   // Notify the change of its image to the current view (COSBView)
    m_pNotifyObj->Change(this, CSpriteNotifyObj::IMAGE, &rcPos);
}

int CStage::TickAll()
{
    DWORD dwCurTick = ::GetTickCount();
    POSITION pos = m_AniList.GetTailPosition();
    while (pos)
    {
        CPhasedSprite* pPS = (CPhasedSprite*)m_AniList.GetPrev(pos); // Increment position.
        if (dwCurTick >= pPS->GetAlarmTick())
        {
            if (pPS->HeartBeat(dwCurTick))
                m_pOSBView->RenderAndDrawDirtyList();
        }
    }
…
}
```

In MFC's OnIdle(long) function, this CStage::TickAll() function is called automatically.



For actors, since each actor has several behaviors involved, we need a kind a switch to designate current behavior.





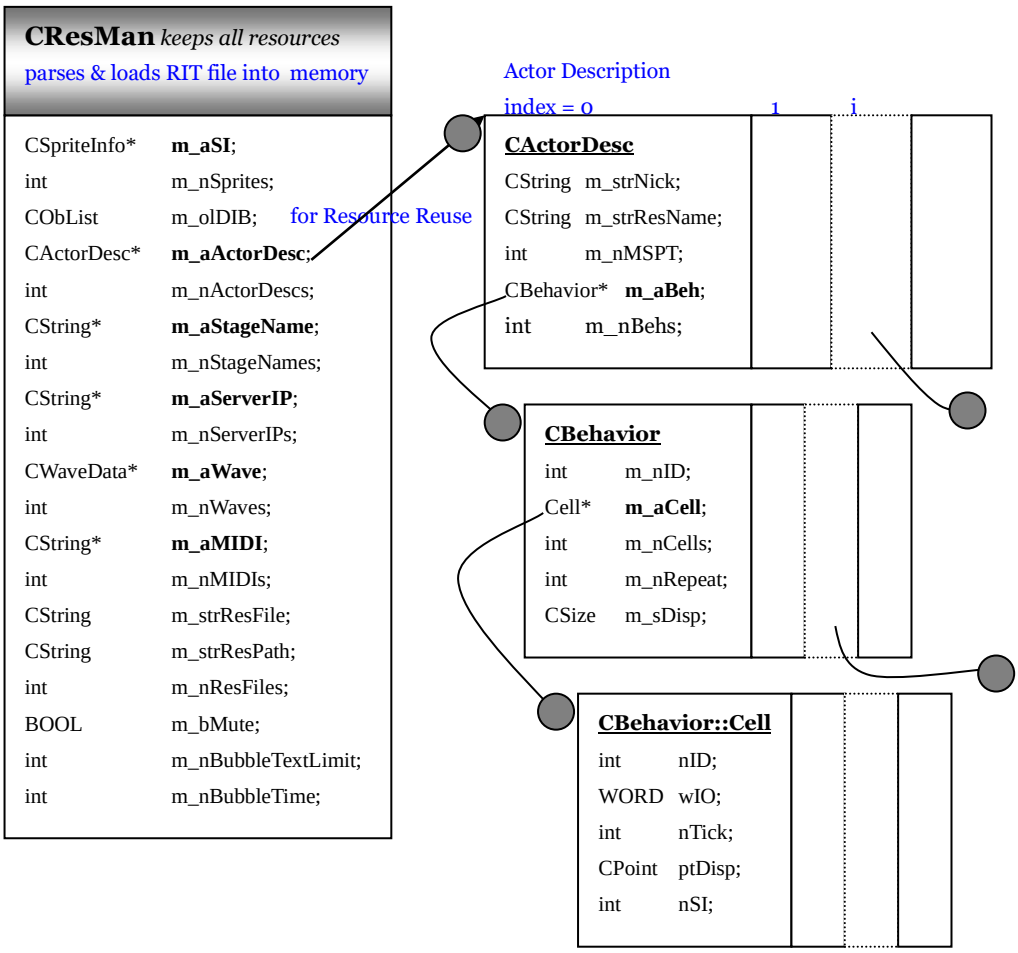
Each sprite notifies its change of state to the view to request redrawing like the following

example code.

```
CSprite::SetZ()
    m_pNotifyObj->Change(this, CSpriteNotifyObj::ZORDER, &rc);

void SpriteListNotifyObj::Change(CSprite*, CHANGETYPE, CRECT*, CRect*)
{
    if (change & CSpriteNotifyObj::ZORDER)
    {
        m_pSpriteList->Reorder(pSprite);
        m_pBufferView->AddDirtyRegion(pRect1);
    }
    ...
}
```

Resource Manager and Actor Description Tables

For actors, CResMan loads actor animation data from RIT file into an array of CActorDesc objects.



Here is an example of Actor Description:

Actor Description

```
#ACTOR=a00|aman_001,man001,40;          // This first actor definition will be used for the default.
{
        // Behaviors
        STANDF=1,(0,2);         // Stand forward
        STANDB=1,(9,2);         // Stand backward
        STANDINGF=1,(|0,2,0,0,schng002)(/0,2)(*0,2)(#0,2)(0,2);   // Appearing in forward stance
        STANDINGB=1,(|9,2,0,0,schng002)(/9,2)(*9,2)(#9,2)(9,2);   // sound id is a predefined wave file in RIT
        ...
}
```

Behavior

Cell

There is a bitmap resource reuse mechanism in CResMan. By using CResMan::LoadDIB() and CResMan::LoadPhasedSprite() one can prevent loading the same bitmap image into memory redundantly. CResMan keeps a list of resource names that are already loaded into memory. So, for any request to load an image with the same resource name, CResMan just returns to the pointer to the memory without actually allocating memory.

4. Communication Architecture using MS ChatSock

MS ChatSock API

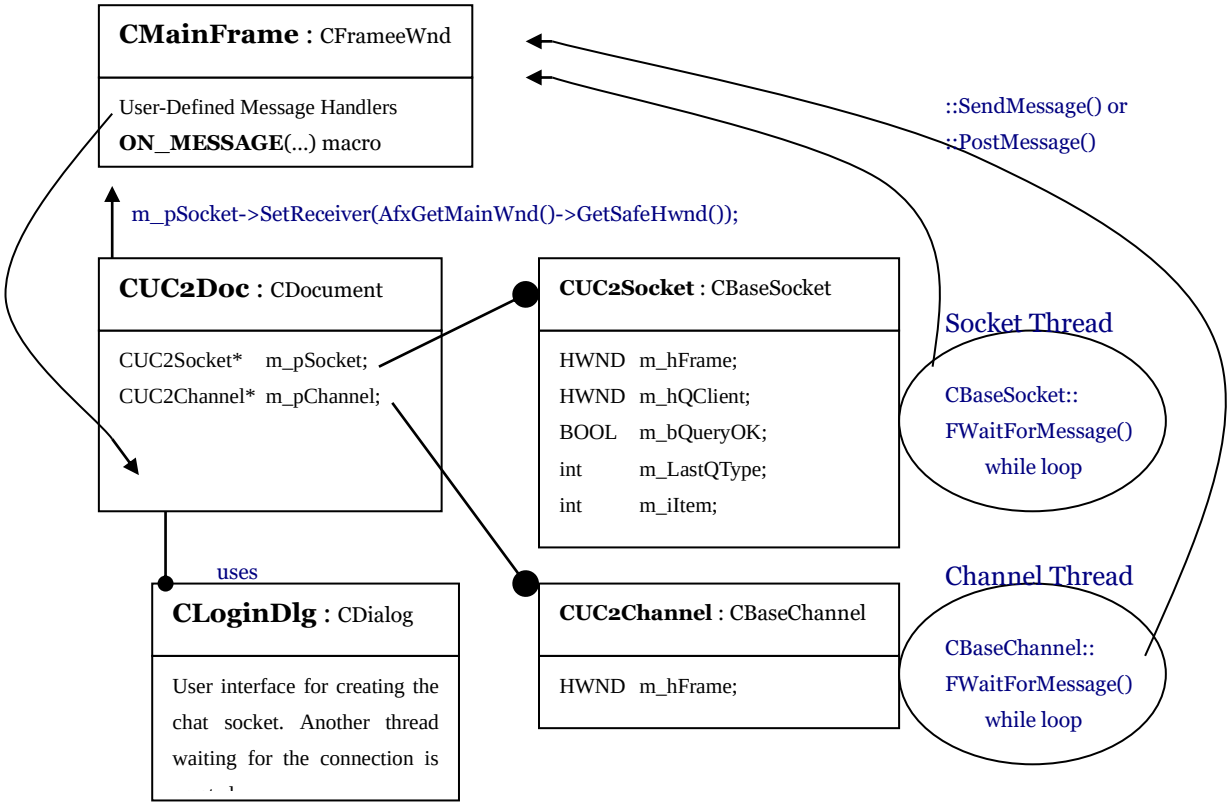
Microsoft provides ChatSock API to support for MIC (Microsoft Internet Chat) protocol. This protocol and APIs are well documented in their SDK. UniChat's communication architecture totally depends on this ChatSock API. For example, our actor in UniChat is implemented in CActor class and there is an interface called ICSMember for each chat client. Then CActor keeps a member for ICSMember to use ChatSock functionalities. So, it is important to identify which object is for ChatSock interfaces and which is for UniChat classes.
Since ChatSock is implemented on the COM (Component Object Model) architecture, programmers have to understand some basic concepts for COM programming, like interfaces, AddRef(), Release().

One thing that confuses the programmers in ChatSock API is the sequence of messages and events. This is not documented so you have to run some sample and trace some events to find out the flow of messages. Here is a diagram that shows how UniChat uses ChatSock messages when a new member joins a channel.

ChatSock-related classes

Class	Description
CBaseSocket	Base Socket functionalities provided by ChatSock
CBaseChannel	Base Channel interface provided by ChatSock
CUC2Socket	Inherited from CBaseSocket, UniChat 2 specific codes
CUC2Channel	Inherited from CBaseChannel, UniChat 2 specific codes

Integration with MFC classes



I made CUC2Doc create and keep the pointer to ChatSock services. When you have a need to use ChatSock services, you have to get the pointer to ChatSock from this document object.

ChatSock knows where to send its messages by *m_hFrame*. This handle is set by CUC2Socket::SetReceiver(const HWND hWnd) in CUC2Doc class. If this is set to NULL, ChatSock doesn't send its message. Because ChatSock is running in another thread, you have to be sure to call SetReciver(NULL) when you're not ready to process the messages.

Socket is created in BOOL CUC2Doc::Connect() through CLoginDlg dialog and channel is created in BOOL CUC2Doc::SetChannel(PICS_CHANNEL picsChannel).

One problem using ChatSock in MFC is that socket services are running in other threads. And in MFC windows, GDI objects, and other objects should be passed between threads by means of handles instead of by means of pointers to MFC objects. That's why I put the handle *m_hFrame* in CUC2Socket and CUC2Channel. ChatSock sends its messages through this handle. But an object of CUC2Doc class that is inherited from CDocument is not a normal window – CDocument has no HWNDs.

So we have to use CMainFrame object as a receiver for socket services. You can see many user-defined message handlers in this class just calling relevant handlers in CUC2Doc. Here is a segment of that code.


```
// MainFrm.h
afx_msg LRESULT OnCsData(WPARAM, LPARAM);
…
// MainFrm.cpp
ON_MESSAGE(CSMSG_CMD_DATA,          OnCsData)
…
LRESULT CMainFrame::OnCsData(WPARAM wParam, LPARAM lParam)
{
    CUC2Doc* pDoc = (CUC2Doc*)GetActiveDocument();
    return pDoc->OnCsData(wParam, lParam);
}
```

It is recommended to read the section on "Sharing MFC Objects Among Threads" in Chap. 14 of Prosise's book. (Programming Windows 95 with MFC, Microsoft Press)

Here is a list of message handlers triggered by the ChatSock service.

```
// MainFrm.h
afx_msg LRESULT OnCsAddChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsPrivateMsg(WPARAM, LPARAM);
afx_msg LRESULT OnCsQueryData(WPARAM, LPARAM);
afx_msg LRESULT OnCsInvite(WPARAM, LPARAM);
afx_msg LRESULT OnCsGotMemList(WPARAM, LPARAM);
afx_msg LRESULT OnCsAddMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsDelChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeMember(WPARAM, LPARAM);
afx_msg LRESULT OnCsModeChannel(WPARAM, LPARAM);
afx_msg LRESULT OnCsTextA(WPARAM, LPARAM);
afx_msg LRESULT OnCsData(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperText(WPARAM, LPARAM);
afx_msg LRESULT OnCsWhisperData(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewTopic(WPARAM, LPARAM);
afx_msg LRESULT OnCsNewNick(WPARAM, LPARAM);
afx_msg LRESULT OnChannelFullRetry(WPARAM, LPARAM);
```

User-defined messages for the communication between ChatSock classes and our application windows are included in "UC2Messages.h" file.

Action Commands

Action commands for character movements or gestures are also enumerated in "UC2Messages.h" file.

I made a simple protocol to send and receive commands between clients. Each client packs some data into a NULL-terminated string as in the following syntax.

X`n`(additional data each separated by `)

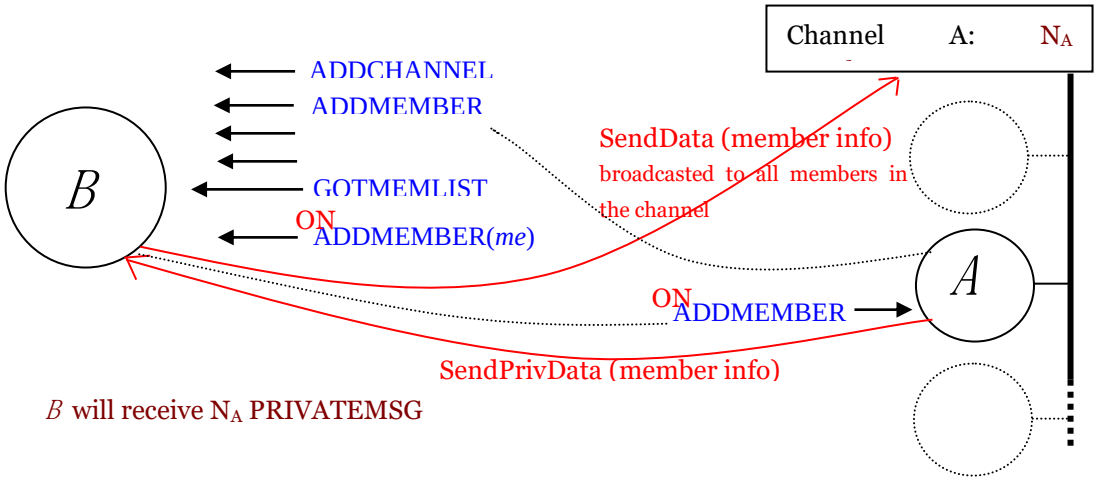
For example,
“X`5`(3,19)`256” … X: flag, 5: command id, (3,19): tile id, 256: state value

This is the command CMD_MOVEF=5. So the client that receives this data will parse the string and move the corresponding character. Since the receiving client already knows where the message came from, we don’t need to add an identifier for the sender. The tile id is included to verify the position after movement. So each time for action commands character positions have a chance to synchronize their positions with other users.

```
enum ACTOR_COMMANDS
{    // Command Enumerators
CMD_BEGIN = 0,
// Management
    CMD_MEMBER_INFO,     //
X`nCmd`nVersion`nCharID`nBubbleKind`strHandle`strRealName`strProfile`ptTID`wState
    CMD_MEMBER_ACTOR, // X`nCmd`nCharID`nBubbleKind        (changed his Actor)
    CMD_NEWS,            // X`nCmd`message
// Position Move
CMD_MOVE,
    CMD_MOVEF,           // X`nCmd`ptTID`wState
    CMD_MOVEB,
    CMD_RES_MOVE,
// State Change
CMD_STATE,
    CMD_STAND,           // State
    CMD_MORPH,
    CMD_DOZE,
    CMD_TURNL,
    CMD_TURNR,
    CMD_RES_STATE,       // reserved
// Actions              // CMD_ACTION is for Just repositioning message
CMD_ACTION,              // Y`nCmd`ptTID`wState` (to verify synchronization)
    CMD_CHAT,
    CMD_ENTER,
    CMD_EXIT,
    CMD_SMILE,
    CMD_MAD,
    CMD_HELLO,
    CMD_CRY,
    CMD_SCRATCH,
    CMD_PICK,
    CMD_SPECIAL,
    CMD_PUNCH,           // Y`nCmd`ptTID`wState`NickTo`
    CMD_BEATEN,
    CMD_RES_ACTION,
```



```
    CMD_END
};
```



CUC2Doc calls CBaseChannel::FSendData() when the client needs to broadcast its message to all the members in the channel. This is used for sending command data which should be shared by all members in the same channel.

CBaseSocket:: FSendPrivData() is for sending data only to a designated user. I used this method to send each client's data to a new member in the channel. Then the new comer will have these messages from other guys already in the channel so that he can gather informations of others.

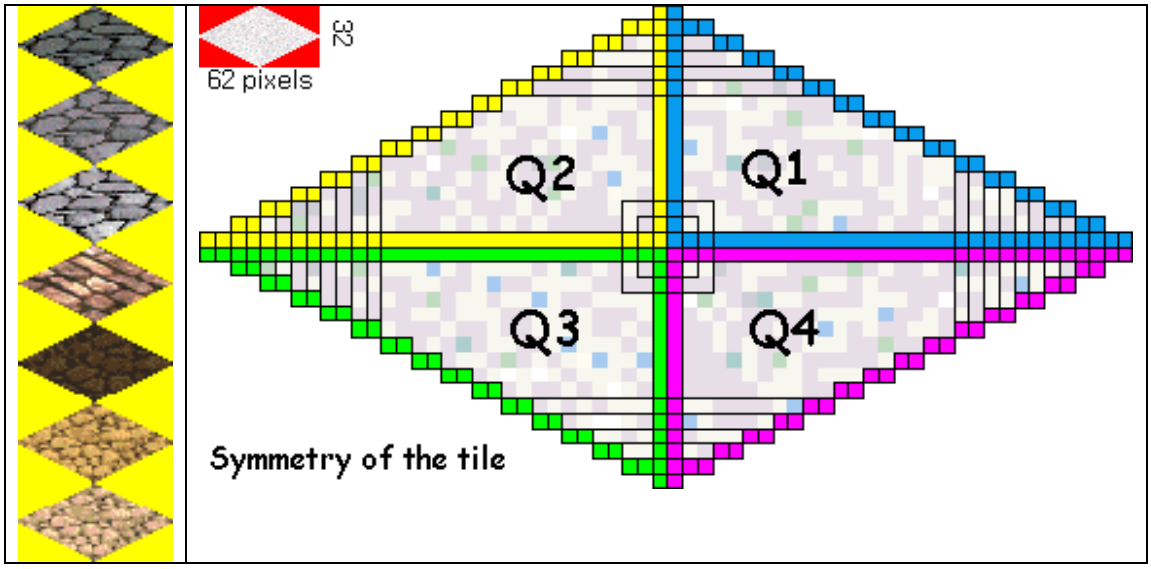
5. Tile Theories

UniChat uses a new kind of data structure for the background drawing. In QuarterView graphics, background image is dynamically composed by a set of tiles. Use of tiles can dramatically reduce in file size for expressing various backgrounds. Owing to this technology, UniChat 2 was able to pack all of the files into 1.2 MB having more than 50 backgrounds.

Tile Attributes

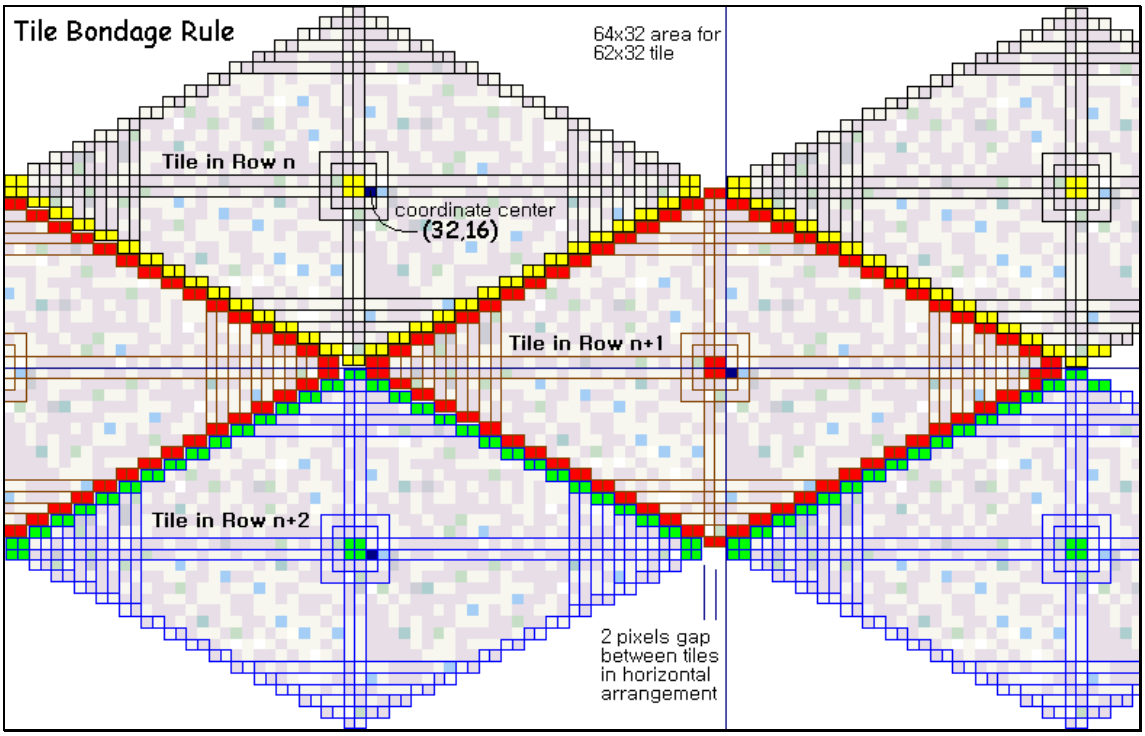
```
TILE
CPhasedSprite* pPS;
int      nEA;
WORD     wDA;
```

In our source code, Tile is defined as a PhasedSprite with two attributes for elevation and direction. CTileMap is a two dimensional array of these tiles. But the sprite for a tile should have the same size and shape as in the following picture:



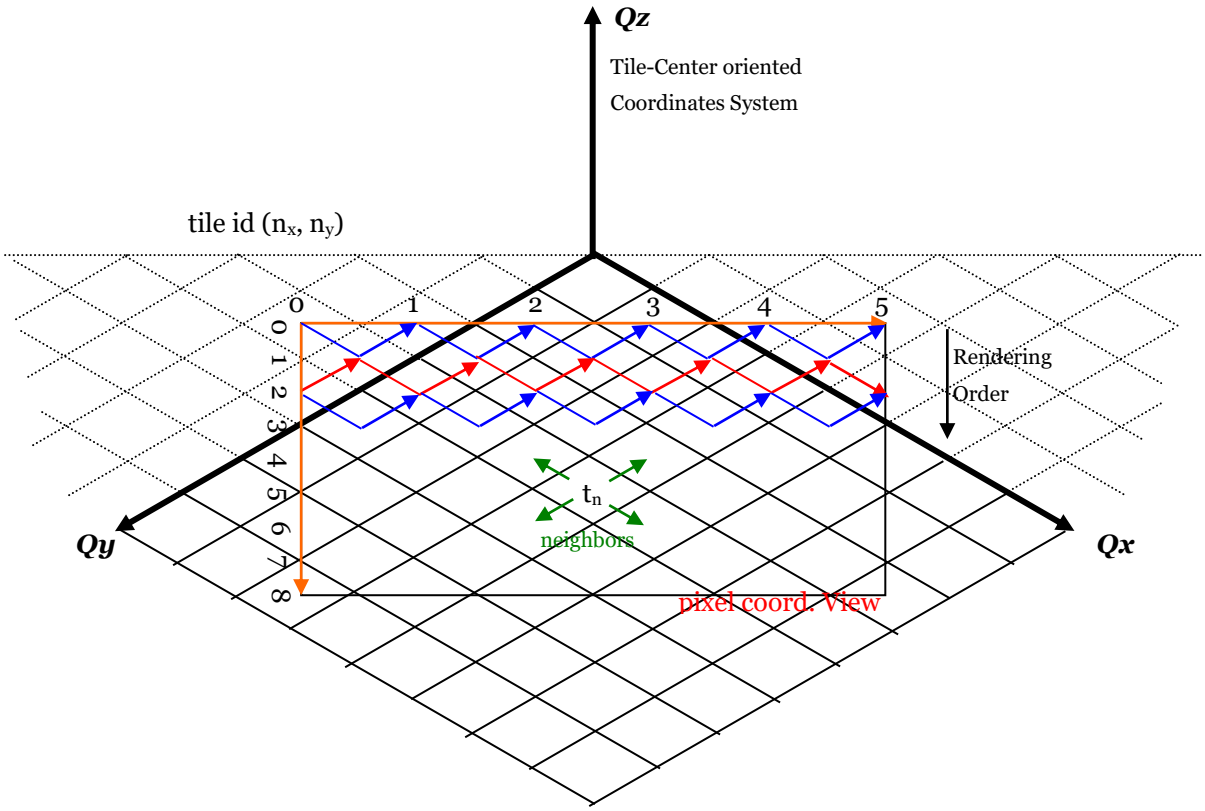
The left image shows many tiles packed into a file. By the horizontal and vertical symmetries of the tile structure, we can flip the image horizontally and vertically forming different images. But it doesn't have a rotational symmetry.

Tile Bondage Rule



Pixels actually engaged in the bondage are shown with different colors grouped by the rows.

Tile Coordinate System



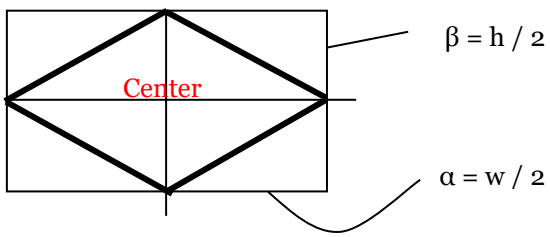
To find the neighboring 4 tiles of tile (n_x, n_y)

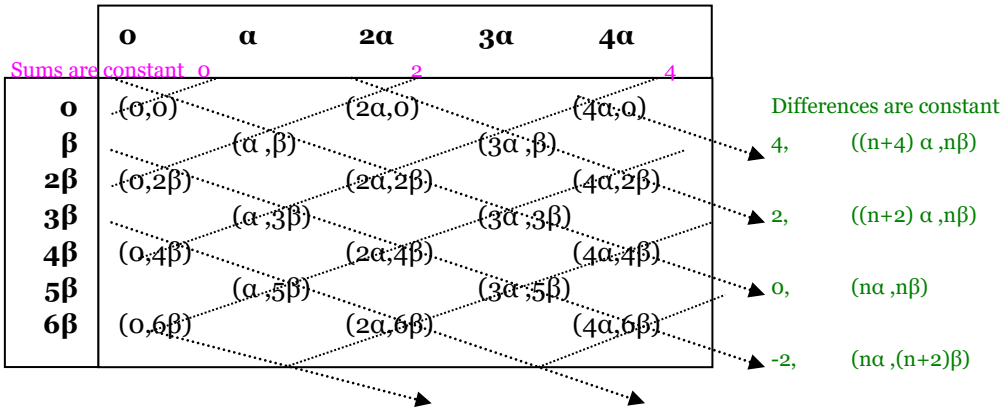
$$
\begin{aligned}
N(n_x, n_y) = \; & \{ (n_x-1, n_y\pm1), (n_x, n_y\pm1)\} \text{ for } n_y = 2n \;\& \\
& \{ (n_x, n_y\pm1), (n_x+1, n_y\pm1)\} \text{ for } n_y = 2n+1 \\
= \; & \{ (n_x-(-1)^{n_y}, n_y\pm1), (n_x, n_y\pm1)\}
\end{aligned}
$$

Tile Hit Test Algorithm

Problem: Find the nearest tile index for a given (x', y') position in pixel coordinates.

According to our tile index system, the center positions of tiles in pixel coordinates are like the following:





Here, α is the half width of the tile and β is the half height of the tile.
In this table of the series of tile center coordinates, we can find that each point is specified by the coordinates of sums and differences of its coefficients. Given sum and difference of the coefficients of x and y, we can find a unique point.

The problem is to find the closest integer M and N that satisfies the following:
Since the coefficient of x coordinate is x / α and that of y coordinate is y / β,
Sum of coef. $= x_c / \alpha + y_c / \beta = 2M$, where M=0,1,2,... - (1)
Difference of coef. $= x_c / \alpha - y_c / \beta = 2N$, where N = ...,-2,-1,0,1,2,... - (2)

Above equations are only applied to the exact center position of each tile. So, for a given point (x', y'), that may not be a center position, we must solve the following inequalities:
$2M-1 < x' / \alpha + y' / \beta <= 2M+1$
$2N-1 < x' / \alpha - y' / \beta <= 2N+1$

Here we define $S= x' / \alpha + y' / \beta$, $D= x' / \alpha - y' / \beta$, and these cannot be integers.
$(S-1)/2 <= M < (S+1)/2$ -(3)
$(D-1)/2 <= N < (D+1)/2$ -(4)

These can be solved for M and N with the condition that they should be integers.

So the problem reduces to finding such an integer that satisfies above inequalities. If we find such integers M and N, we can solve for the center point.
By adding and subtracting equations (1) and (2), we get

$C = (\alpha(M+N), \beta(M-N))$ -(5)

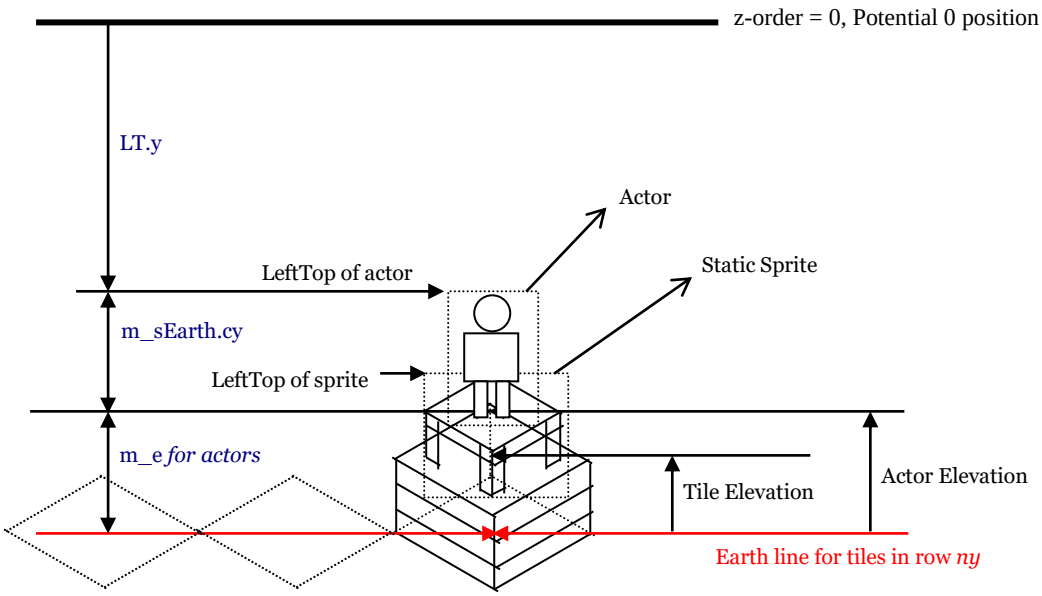
This algorithm has been implemented as a function CTileMap::GetNearestTileCenter(const CPoint& pt).

```
CPoint CTileMap::GetNearestTileCenter(const CPoint& pt) const
{
    float fx = float(pt.x)/m_sTH.cx;   // x/a
    float fy = float(pt.y)/m_sTH.cy;   // y/b
    double fM = (fx + fy + 1.)/2.; // (S-1)/2 <= M < (S+1)/2
```

```
    double fN = (fx - fy + 1.)/2.;  // (D-1)/2 <= N < (D+1)/2 can be negative
    if (fN <= 0.)    // Consider a problem to find an integer that satisfies
        fN--;        // -1.5 <= N < -0.5, But int(-0.5)=0
    int M = int(fM);
    int N = int(fN);
    return CPoint(m_sTH.cx*(M + N), m_sTH.cy*(M - N));
}
```

Attributes for Rendering Order

The program has to know which sprite in the sprite list should be rendered first. This is the rendering order problem. In normal animation technique, every sprite has an attribute called *z-order* to indicate this order. But in quarterview this is not sufficient. Since we are actually in 3D coordinate space in quarterview, we need one more coordinate besides x and y. So I have introduced an attribute named *elevation*.



z = -GetEarthPointY();
= - (LT.y + m_sEarth.cy + m_e)

Z orders are the same for any actors on the tiles in the same row. Actually Z-order is determined by the tile on which the actor stands. This center of the tile is the *earth point*.

6. Some Methods

There are some commonly used methods in UniChat 2 source codes. It is some kind of personal programming style. But to understand the codes easily you have to identify these styles.

Dynamic Array

This is a method for allocating memories for a list of objects. If we can't determine the number

of the objects at compile time, we cannot use static array definition in our code. Normally the number of objects is read from data file and determined by some calculations at runtime, so this method is very useful.

This method is just another usage of C pointer and some kind of naming rule.
For example, to construct a list of string objects like the following in memory:

```
Data File     // SIT files
{
0000csin;     // 0000csin.sit
0001ctrm;
…
0009ctrm;
}

// Sample Source
class CSample
{
…
    CString* m_aSIT;
    int      m_nSITs;
}

CSample::Load()
{
    m_nSITs = parser.CountItems();      // Let's say parser.CountItems() returns the number of items in the data file
    m_aSIT = new CString[m_nSITs];
    for (int i=0; i < m_nSITs; i++)
        m_aSIT[I] = parser.ReadLine();
}

CSample::~CSample()
{
    if (m_aSIT)
        delete [ ] m_aSIT;
}
```

This is the pattern used to handle this kind of dynamic memory.

TYPE*	**m_a**Obj;	// pointer to an array of Objs
int	**m_n**Obj**s**;	// the number of objects

Parser

A text line parser is globally used in UniChat code to interpret our data files and load into the memory structured. CParser, once implemented in DOS environment, is a general class for this purpose modified for MFC classes. CTextFileBuffer is a utility class that enables CParser to use memory loaded files and LZ compressed files.

If you use CParser, it's very easy to count some items before dynamically allocating memory and to add some comment rules. Moreover some basic data types are interpreted according to the variable types with the same name of member function in CParser.

Here is a fragment of codes that show a typical use of this class:

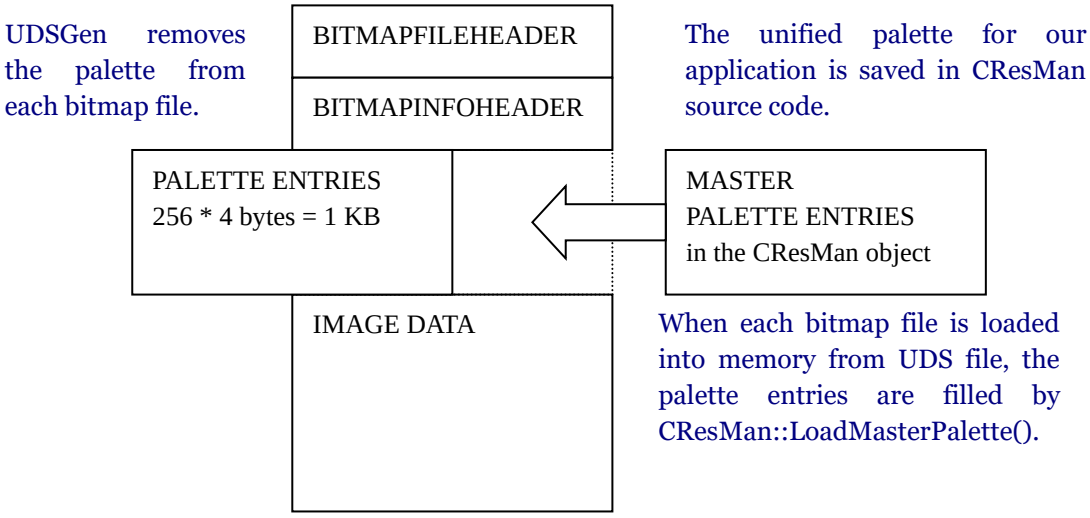
```
extern CParser gParser;

BOOL CTestDoc::OnNewDocument()
{
    ...
    TRY
    {
        CStdioFile f("sample.txt", CFile::modeRead | CFile::typeText);
        char*   szVal = new char[gParser.GetMaxBuffer()];
        int     iVal;
        double fVal;
        CPoint ptVal;
        while(f.ReadString(szBuf, gParser.GetMaxBuffer()))
        {
            gParser.CopyBuffer(szBuf);  // "Toto=1,3.4,(12,345);"
            if (gParser.IsCommentLine())
                continue;

            gParser.GetValueRightToken(szVal, '=');
            gParser.SetLeftToken('=');
            gParser.GetValueRightToken(iVal, ',');
            gParser.GetValueRightToken(fVal, ',');
            gParser.GetValueRightToken(ptVal);
        }
        return TRUE;
        f.Close();
        delete [] szBuf;
    }
    CATCH( CFileException, e )
    {
        #ifdef _DEBUG
            afxDump << "File could not be opened " << e->m_cause << "\n";
        #endif
        delete [] szBuf;
        return FALSE;
    }
    END_CATCH
}
```

256 color Palettes in UniChat 2

Windows 256-color bitmap files have a palette in its header part. But in UniChat 2, we use only

on color table for all the images except for the dialog. So we made a new type of data source for image files without palette. The master palette in UniChat 2 is defined as a static array in the Resource Manager object (CResMan). Then when we load each bitmap into memory, the color table entries are filled with this master palette values by calling CResMan:: LoadMasterPalette(CDIB*).

```
static const PALETTEENTRY apeMASTER[256] =
{
#include "U2Pal.Inc"  // 256 Color Table
};
```



Predefined Palette Indices

In addition to the unified master palette of UniChat, some indices of the palette are assigned as a special purpose. For example, index 240 is registered as an outline color. Here is the rules for our master palette in UniChat.

Index Range	Description
0..9,246..255	Windows System Colors (20)
241..245	Advertising clips (5)
240	Outline Color, rgb=(131,231,131)
239	Black (0,0,0)
238	White (255,255,255)
237	Outline Off, This is the current outline index.
236	Transparent Color
232..235	Hair Color Set #0 (4) – Image contains
228..231	Hair Color Set #1 – program switches
224..227	Hair Color Set #2 – program switches
220..223	Hair Color Set #3 – program switches
216..219	Face color set (4)
212..215	Clothes Color Set #0 (4) – Image contains
208..211	Clothes Color Set #1 – program switches
204..207	Clothes Color Set #2 – program switches
200..203	Clothes Color Set #3 – program switches

196..199	Fixed Color Set (4)
10..195	Background Colors (186)

The reason to use these predefined sets of indices in drawing bitmap graphics is to give some variations in colors of the same image. For example, in a chat room if the same character is selected then the program automatically changes its hair and clothes color sets.

It's important for the graphic designers to keep this rule. They have to draw characters hair using only the colors in the hair color set #0. Other three sets for hair colors are reserved for our program to switch with the original indices.

void CResMan::**RotateActorColorSet**(CDIB* pDIB, const int nColorSet) const;

Every character has a border line or outline drawn with the index of 237. This rgb value is equal to the transparent color so users cannot identify the outline. But once the program determined that the character should show its outline, for the client's own character, when the program loads this image into memory, the rgb value of index 237 is replaced with that of index 240.

void CResMan::**ShowOutline**(CDIB* pDIB) const;

One important thing for our graphics system is that we use identity palette. We us this for better performance in animation which involves repeated rendering and drawing of the same image. In this scheme, while loading the bitmap image into memory, all the bits in the image are replaced with a new index set in the identity palette. So once we load an image from a bitmap file, we cannot say that the bits in the file are intact in memory.

How to make a dialog with 256-color image

There is no support for the 256-color bitmap in MFC. Using our own graphics library, UC2Ani, we can simply make a dialog window with a background image of 256 colors. Or you can tile the background.

Once you know how to use this graphics library, all the processes can be shown as a pattern. Here is the procedure. This code also shows how we can make a 256-color buttons.

Step	Procedure
1	Make a dialog template in resource view of the VC++ workspace panel. Using ClassWizard, make associated code and header file for the CDialog inherited class. Let's say it CTestDlg.
2	In header file add the following codes, #include "UC2Ani/DIB.h" #include "UC2Ani/DIBPal.h" #include "UC2Ani/PSButton.h" class CDIB; #define TESTFILE_BMP "c:\\test.bmp" In the class definition, protected:


```
    CDIB*       m_pDIBBack;     // Background frame image
    CPalette*   m_pPal;         // main palette
    BOOL        m_bPaletteCreated;
```

3 In CMainFrame class, add a utility function to determine current video mode.

```
BOOL CMainFrame::Is256Palette() const
{
    BOOL bResult=TRUE;
    // Get a screen DC to work with.
    HWND hwndActive = ::GetActiveWindow();
    HDC hdcScreen = ::GetDC(hwndActive);
    ASSERT(hdcScreen);

    // Make sure we are on a palettized device.
    if (!(::GetDeviceCaps(hdcScreen, RASTERCAPS) & RC_PALETTE))
    {
        bResult = FALSE;
    }
    else
    {
        // Get the number of system colors and the number of palette
        // entries. Note that on a palletized device the number of
        // colors is the number of guaranteed colors, i.e., the number
        // of reserved system colors.
        int iSysColors = ::GetDeviceCaps(hdcScreen, NUMCOLORS);
        int iPalEntries = ::GetDeviceCaps(hdcScreen, SIZEPALETTE);

        // If there are more than 256 colors we are wasting our time.
        if (iSysColors < 0 || iSysColors > 256)
        {
            bResult = FALSE;
        }
    }
    ::ReleaseDC(hwndActive, hdcScreen);
    return bResult;
}
```

4 In constructor,

```
CTestDlg::CTestDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CTestDlg::IDD, pParent)
{
…
    m_pPal = NULL;  // Set it NULL before loading DIB

    CMainFrame* pMF = (CMainFrame*)AfxGetMainWnd();

    CString strFile(TESTFILE_BMP);
    m_pDIBBack = new CDIB;
    if (!m_pDIBBack->Load(strFile))
    {
        delete m_pDIBBack;
        m_pDIBBack = NULL;
        return;
    }

    if (pMF->Is256Palette()) // Assume that this function is already implemented
    {   // Use mainframe's palette to avoid color flickering
```

```
            m_pPal = pMF->GetPalette();
            m_pDIBBack->MapColorsToPalette(m_pPal);
            m_bPaletteCreated = FALSE;
        }
        else    // Use original palette in the file for TRUE color system
        {       // Create the palette from the DIB.
            CDIBPal* pDIBPal;
            pDIBPal = new CDIBPal;
            ASSERT(pDIBPal);
            if (!pDIBPal->Create(m_pDIBBack))
            {
                AfxMessageBox("Failed to create palette from DIB file");
                delete pDIBPal;
            }
            m_pPal = pDIBPal;    // type casting to parent class
            m_bPaletteCreated = TRUE;
        }
    }
```

5 In destructor,

```
CTestDlg::~CTestDlg()
{
    if (m_pDIBBack)
        delete m_pDIBBack;
    if (m_pPal && m_bPaletteCreated)
        delete m_pPal;
}
```

6 Using ClassWizard add following message handlers:

```
    virtual BOOL OnInitDialog();
    afx_msg void OnPaletteChanged(CWnd* pFocusWnd);
    afx_msg BOOL OnQueryNewPalette();
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
```

7 Modify OnInitDialog()

```
BOOL CTestDlg::OnInitDialog()
{
    if (!m_pDIBBack)
        return FALSE;
    CDialog::OnInitDialog();
    m_btnOK.SubclassDlgItem(IDOK, this);
    m_btnCancel.SubclassDlgItem(IDCANCEL, this);

    CPoint ptLT(349, 238);  // find adequate button position
    m_btnOK.MoveResize(ptLT);
    ptLT.x = 17;
    m_btnCancel.MoveResize(ptLT);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

8 Add palette message handlers.

```
void CTestDlg::OnPaletteChanged(CWnd* pFocusWnd)
{
```



```
        CDialog::OnPaletteChanged(pFocusWnd);

        if (pFocusWnd != this)
            OnQueryNewPalette();
    }

    BOOL CTestDlg::OnQueryNewPalette()
    {
        if (m_pPal)
        {
            CDC* pdc = GetDC();
            CPalette* pPalOld = pdc->SelectPalette(m_pPal, FALSE);          // foreground
            UINT u = pdc->RealizePalette();
            if (pPalOld)
                pdc->SelectPalette(pPalOld, FALSE);
            ReleaseDC(pdc);
//          if (u)
//          {   // Some colors changed so we need to do a repaint.
                Invalidate(); // Repaint the lot.
                return TRUE; // Say we did something.
//          }
        }
        return FALSE; // Say we did nothing.
    }
```

9 Finally add a handler for WM_SIZE to fit for the background image.

```
    void CTestDlg::OnSize(UINT nType, int cx, int cy)
    {
        CDialog::OnSize(nType, cx, cy);

        if (m_pDIBBack)
        {
            SetWindowPos(NULL, 0, 0,
                        m_pDIBBack->GetWidth(), m_pDIBBack->GetHeight(),
                        SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE);
        }
    }
```

MFC SDI Architecture

It's very helpful if you know the sequence of calling MFC functions in your MFC SDI application.

Loaded symbols for 'C:\WINDOWS\SYSTEM\MSVCRTD.DLL'
Loaded symbols for 'C:\WINDOWS\SYSTEM\MFC42D.DLL'
// Loading...
CGenSDIApp::CGenSDIApp() // Application class constructor
CGenSDIApp::InitApplication() // for backward compatibility
CGenSDIApp::**InitInstance**()

CGenSDIDoc::CGenSDIDoc() // Document constructor

CMainFrame::CMainFrame() // Frame Window constructor
CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
CMainFrame::PreCreateWindow(CREATESTRUCT& cs) // why called twice?

CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct) // WM_CREATE
CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)

CGenSDIView::CGenSDIView() // View Window constructor
CGenSDIView::PreCreateWindow(CREATESTRUCT& cs)
CGenSDIView::OnCreate(LPCREATESTRUCT lpCreateStruct)// WM_CREATE

CGenSDIDoc::SetTitle(Untitled)
CGenSDIDoc::**DeleteContents**()
CGenSDIDoc::**OnNewDocument**()

CGenSDIView::**OnInitialUpdate**()
CMainFrame::ActivateFrame(int nCmdShow)
CGenSDIView::OnDraw(CDC* pDC=0x63f970)

// Application Logic will be here.

// Closing...
CMainFrame::**OnClose**() // WM_CLOSE

CGenSDIDoc::CanCloseFrame(CFrameWnd* pFrame=0x750d14)
CGenSDIDoc::OnCloseDocument()

CMainFrame::OnDestroy() // WM_DESTROY

CGenSDIView::OnDestroy() // WM_DESTROY
CGenSDIView::**~CGenSDIView**() // View Window destructor

CMainFrame::**~CMainFrame**() // Frame Window destructor

CGenSDIDoc::**DeleteContents**()
CGenSDIDoc::**~CGenSDIDoc**() // Document destructor

CGenSDIApp::**ExitInstance**()

The program 'D:\MsDev\Projects\kSm\GenSDI\Debug\GenSDI.exe' has exited with code 0 (0x0).

7. Class Hierarchies