



VIEWMAX 2000 API

Application Programming Interface

Development Guide

Preliminary

Table of Contents

INTRODUCTION	5
SYSTEM GUIDE - AUDIENCE	6
INFORMATION SOURCES	6
GENERAL DESCRIPTION	7
GENERAL DESCRIPTION	8
<i>Application Programmer's Interface</i>	8
<i>ViewMax 2000 World Wide Web Interface</i>	8
THE COMPONENTS OF STANDARD VIEWMAX 2000	10
A TYPICAL VIEWMAX2000 SESSION (A PAGE REQUEST)	10
THE COMPONENTS OF VIEWMAX 2000 WITH THE API	12
USING THE VIEWMAX 2000 API	12
API LANGUAGE REFERENCE	15
API REFERENCE	16
Classes	16
Constructor	16
Destructor	16
close() - Close a ViewMax Session	17
collect() - Collect Tag Fields from connected ViewMax 2000 Session	18
connect() - Connect to an existing ViewMax 2000 Session	19
disconnect() - Disconnect from a ViewMax Session	20
fieldCount() - Get the Number of Tag Fields Returned	21
flushFields() - Flush the Session Field Space	22
getErrorMsg() - Get the Error Message	23
getErrorCode() - Get the Error Message	24
getField() - Get the Contents of a Field by Name or Index	25
getLink() - Get the current value of Link from a ViewMax Session	26
getSocket() - Get the File Descriptor of the ViewMax Socket Used	27
getUniqueId() - Get the Unique Id	28
open() - Open a Session to a ViewMax Server	29
putField() - Create a ViewMax Input Field	30
setDebugLevel() - Set the Debug Level	31
setTimeout() - Set the Default Timeout	32
setUniqueId() - Set the Unique Id	33
submit() - Submit Fields to a connected ViewMax 2000 Session	34
API PROGRAM EXAMPLES	35
C++ EXAMPLE	36
<i>Dos.cpp</i>	36
J++ EXAMPLE	45
<i>Dos.java</i>	45

Figures

Figure 1 Overview of ViewMax 2000..... 8
Figure 2 Components of ViewMax 2000..... 10
Figure 3 Components of ViewMax 2000 - Using The API 12

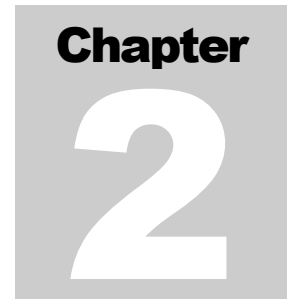


Introduction

System Guide - Audience

The **ViewMax 2000** System Guide is aimed at Developers who need to interface their own applications to **ViewMax 2000**.

Information Sources



General Description

General Description

Application Programmer's Interface

The **ViewMax 2000** API enables developers to replace the standard ViewMax web browser front-end with their own application interface. Using the API, back-end application screens can be driven from a front-end interface, submitting and collecting data as required. The configuration of mapping scripts is performed using the **ViewMax 2000** Java configuration tool.

To use the **ViewMax 2000** API it helps to understand how it is used in the **ViewMax 2000** Word Wide Web Interface.

ViewMax 2000 World Wide Web Interface

The original **ViewMax 2000** design enables applications using a variety of terminal interfaces (VT100, 327x, 5250, etc.) to be accessed by a Web browser.

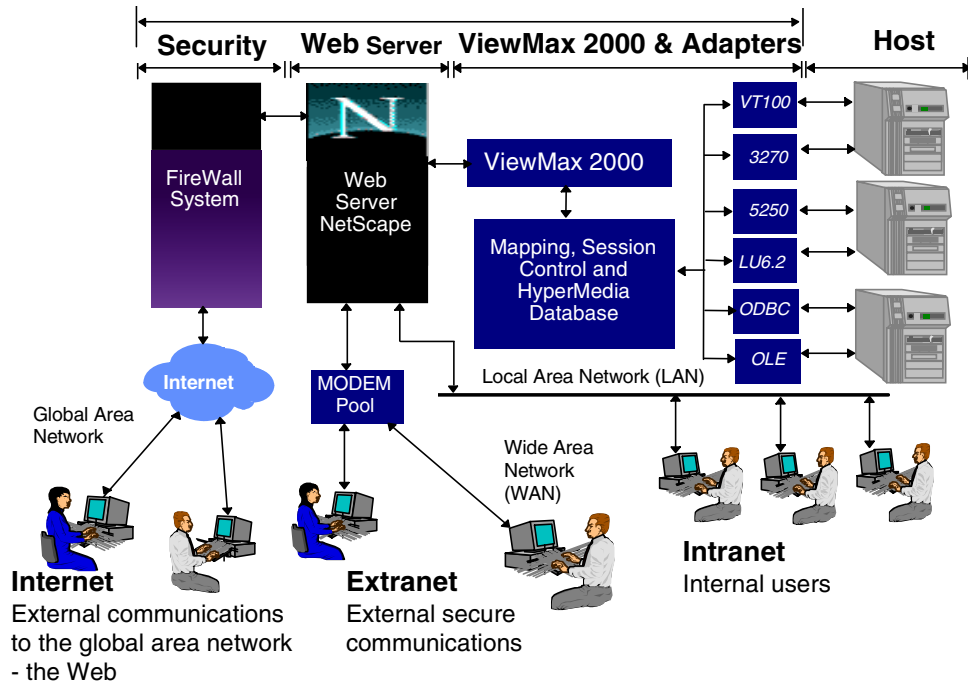


Figure 1 Overview of ViewMax 2000

ViewMax 2000 API
GENERAL DESCRIPTION

ViewMax 2000 offers a friendlier graphical user interface to existing green screen based applications. It provides an alternative link to existing mainframe applications. The interface is virtual-screen based, the application interacts with **ViewMax 2000** as though it were another (vt100, 327x or 5250) terminal. Therefore, no changes need to be made to the application.

The Components of Standard ViewMax 2000

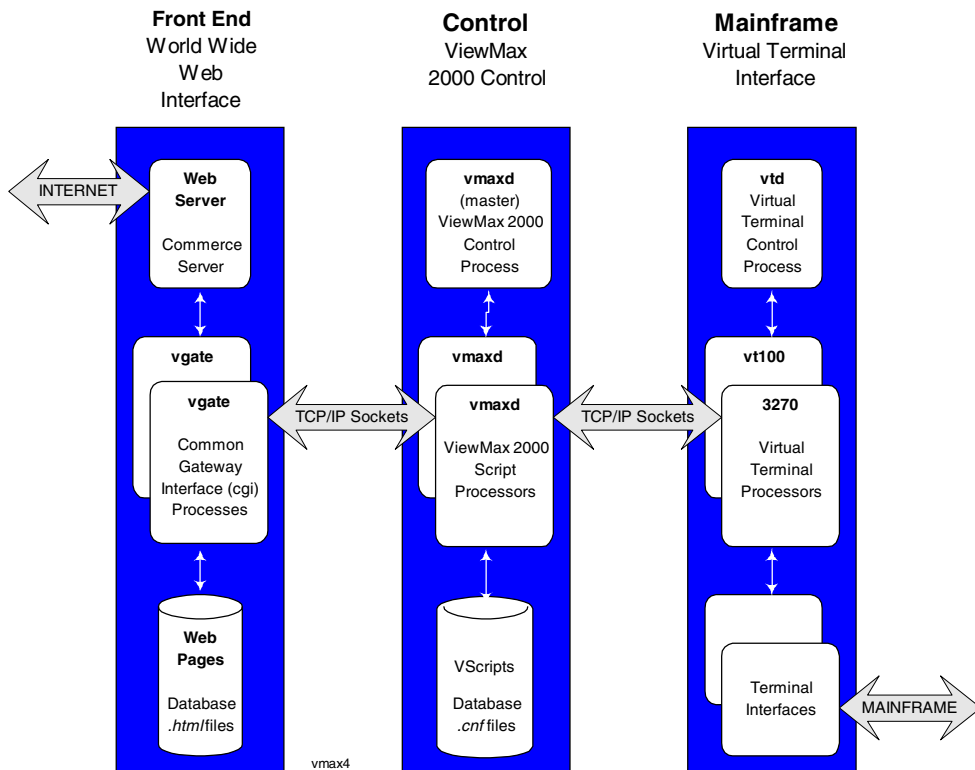


Figure 2 Components of ViewMax 2000

A Typical ViewMax2000 Session (a page request)

A ViewMax 2000 server sitting in a quiescent state has two daemon processes running, **vmaxd** and **vtd**.

All web forms submitted during a ViewMax session activate the ViewMax CGI (Common Gateway Interface) process **vgate**. **vgate** is activated when the web server receives a URL request pointing at it. Once **vgate** starts, it collects any html form field information from the server using the CGI. This information will be a list of name/value pairs which reflect the html form field names and their values. The name of a VScript is also passed to **vgate** within the query string which was appended to the form action property of the html page.

If the URL request is for an initial session connection then **vgate** connects to the daemon **vmaxd** process, requests a new session (forks a child process) and connects to it. If the session is already established, it simply makes a connection to the existing session.

Once the connection is made to **vmaxd**, **vgate** sends it the html field information and the name of the VScript file to run.

vmaxd receives all of the field and script name information and runs the VScript file whose name has been passed to it. The VScript runs and executes whatever commands it has been configured to. This normally entails asking the **vtd** process to start a virtual terminal session. Any data needed to be sent back to the browser is put into ViewMax tag variables by the VScript.

The last command in a VScript sequence is “next_html nextpage”, which triggers **vmaxd** to send back to **vgate** all the tag variables that have been loaded. The “nextpage” argument to “next_html” is also sent back to **vgate** and **vmaxd** then enters its quiescent, listening state.

Once **vgate** receives the tag variables and the name of the next html page from **vmaxd**, it loads the html page from disc. **vgate** then scans the html page for ViewMax tag variables that match the names of those passed from **vmaxd**. If the names match, the tag variables on the html page are replaced by the values received from **vmaxd**. Finally, the completed html page is sent back to the web server where it is returned to the requesting browser. Once **vgate** has returned the html page, it exits.

This sequence of events is repeated until a “last_html lastpage” command is encountered in a VScript, at which point the **vmaxd** session also exits.

The Components of ViewMax 2000 with the API

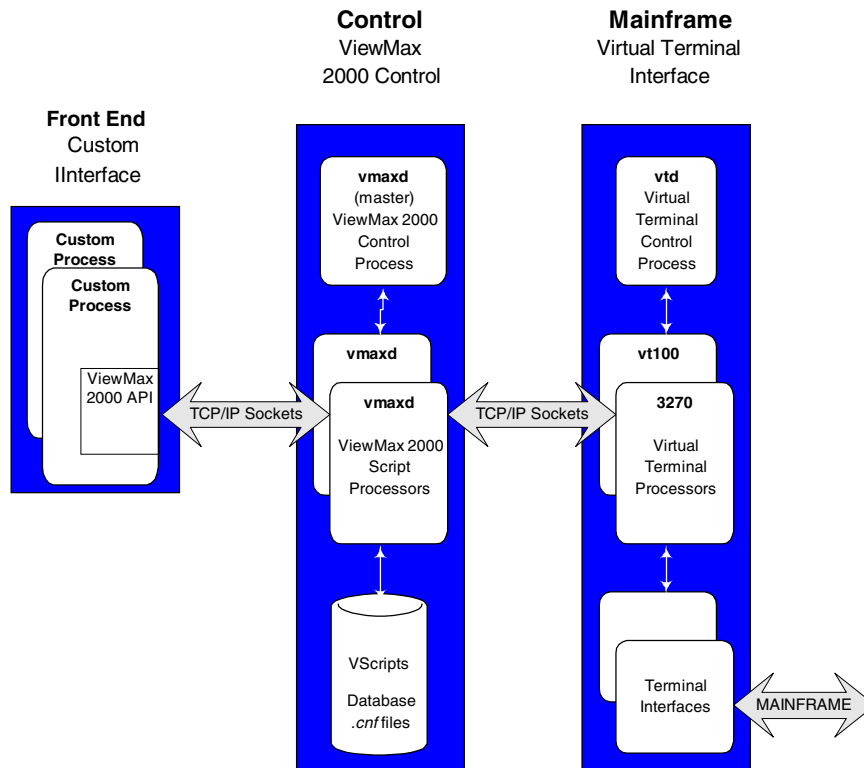


Figure 3 Components of ViewMax 2000 - Using The API

Using the ViewMax 2000 API

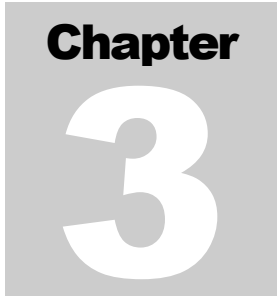
The ViewMax API has been designed to mimic the operation of the standard **vgate** process. The sequence of events required to drive the ViewMax server using the API are:

1. Connect to **vmaxd** and start a session.
2. Send **vmaxd** a list of fields and the name of a VScript to run.
3. Collect the ViewMax tag variables returned from **vmaxd**
4. Repeat steps 2 - 3 as required.
5. Close the session.

It is possible to disconnect from **vmaxd** and reconnect later as long as the session was not closed.

ViewMax 2000 API
GENERAL DESCRIPTION

Custom application programs may (using the ViewMax API) connect directly over a network to the ViewMax server. Alternatively they could reside on the same server, connecting to ViewMax using localhost.



API Language Reference

API Reference

Classes

Session

Constructor

Session()

Destructor

~Session()

close() - Close a ViewMax Session

C++ **int Session.close()**

Java **int Session.close()**

Description:

Close disconnects from and closes the current ViewMax session, such that it is not possible to re-connect to the same session. A close() can only be performed on a currently open and connected session.

Parameters:

None.

Return values:

On successful completion a zero will be returned. A value -1 indicates an error has occurred.

Error codes:

NOT_OPENED

NOT_CONNECTED

CLOSE_ERROR

SEQUENCE_ERROR

WRITE_ERROR

READ_ERROR

TIMED_OUT

collect() - Collect Tag Fields from connected ViewMax 2000 Session

C++ **int Session.collect()**

Java **int Session.collect()**

Description:

Collect is only legitimate if a Session.Submit() call has been made. The collect will wait for the response to the Submit from the ViewMax 2000 server before returning to the user. When the collect() returns the session field space will have been flushed and loaded with any tag field name/value pairs.

Following this call, the link variables become meaningful and can be accessed by the method getLink(). fieldCount() may also be used to find the number of tag fields that were sent from the ViewMax server.

Return values:

On successful completion a zero will be returned. A value -1 indicates an error has occurred.

Error codes:

NOT_CONNECTED
NOT_SUBMITTED
READ_ERROR
SEQUENCE_ERROR
SCRIPT_FAILURE

connect() - Connect to an existing ViewMax 2000 Session

C++ `int Session.connect(char *vmHost, int sessionId)`
 `int Session.connect(char *vmHost, int sessionId, int uniqueId)`

Java `int Session.connect(String vmHost, int sessionId)`
 `int Session.connect(String vmHost, int sessionId, int uniqueId)`

Description:

Connect() attempts to connect to a previously opened session which has been disconnected. The sessionId is the value returned when the session was initially opened. If this is a new session object then the open() session call must have been called to get a uniqueId or a uniqueId must be supplied from a previous call to open() by another application or Browser submission.

Parameters:

vmHost The hostname of the ViewMax 2000 server
sessionId The sessionId number;
uniqueId The uniqueId number;

Return values:

On successful completion a zero will be returned. A value of -1 indicates an error has occurred.

Error codes:

ALREADY_CONNECTED
CONNECT_ERROR
SEQUENCE_ERROR
WRITE_ERROR
READ_ERROR
TIMED_OUT

disconnect() - Disconnect from a ViewMax Session

C++ **int Session.disconnect()**

Java **int Session.disconnect()**

Description:

Disconnect, drops the TCP/IP connection to the currently connected session. However, it leaves the session open for subsequent re-connection.

Parameters:

None.

Return values:

On successful completion a **uniqueId** will be returned, that can be used in subsequent connect() calls. A value of -1 indicates an error has occurred.

Error codes:

NOT_CONNECTED
DISCONNECT_ERROR
SEQUENCE_ERROR
WRITE_ERROR
READ_ERROR
TIMED_OUT

fieldCount() - Get the Number of Tag Fields Returned

C++ **int Session.fieldCount()**

Java **int Session.fieldCount()**

Description:

fieldCount() returns the number of fields that are present in the session's field space. The field count is only valid when a session has been opened.

Return values:

On successful completion the number of fields are returned. A value of -1 indicates an error has occurred.

Error code:

NOT OPENED

flushFields()- Flush the Session Field Space

C++ **int Session.flushFields()**

Java **int Session.flushFields()**

Description:

FlushFields flushes the field space of the current session i.e. causes ALL field variables and their values to be deleted.

The field count is only valid when a session has been opened.

Return values:

On successful completion a zero will be returned. A value of -1 indicates an error has occurred.

Error codes:

NOT_OPENED

getErrorMsg() - Get the Error Message

C++ `const char *Session.getErrorMsg()`

Java `String Session.getErrorMsg()`

Description:

getErrorMsg() gets the last error message that was generated by the session.

Return values:

On completion a pointer to the error message will be returned, if there has not been an error the message will be NULL.

getErrorCode()- Get the Error Message

C++ **int Session.getErrorCode()**

Java **int Session.getErrorCode()**

Description:

getErrorCode() gets the last error code that was generated by the session.

Return values:

On completion an integer Code will be returned, if there has not been an error the value returned will be 0.

getField() - Get the Contents of a Field by Name or Index

C++ **int Session.getField(const char *name, char *&value)**

Java **int Session.getField(String name, StringBuffer value)**

Description:

The getField(by Name) returns into the value argument of the current contents of the field referenced by the name argument. If the field does not exist value will be loaded with a null.

This method is normally used for accessing the tag fields returned from the ViewMax server after a collect(). Tag fields are loaded up by VScript language commands running on the ViewMax server and are returned to the API process for collection by the user. The tag fields can be retrieved by name or index using this or the following command.

Return values:

On completion a zero will be returned.

Error codes:

NOT_OPENED - This error code will be set as a warning but will not cause a bad return.

C++ **int Session.getField(int index, char *&name, char *&value)**

Description:

getField (by Index) returns into the value argument the current contents of the field referenced in the index variable. If the field does not exist, value will be loaded with a null. Additionally, the name of the variable will be loaded into the name argument.

Return values:

On completion a zero will be returned

Error codes:

NOT_OPENED - This error code will be set as a warning but will not cause a bad return.

J++ **Enumeration getField()**

Description:

getField (by Enumeration) returns the Enumeration of all Fields. If there are no fields the Error Code will be set to NOT_OPENED.

Return values:

On completion an Enumeration will be returned or NULL

Error codes:

NOT_OPENED - This error code will be set as a warning but will not cause a bad return.

getLink() - Get the current value of Link from a ViewMax Session

C++ `const char *GetLink()`

Java `String GetLink()`

Description:

The link is only meaningful **following** a successful Session.Submit call. This variable is used to convey the name of the link argument associated with the VScript commands “next_html” or “last_html”.

Return values:

On successful completion a pointer to the session’s current link value will be returned. If the link variable is invalid a null pointer will be returned. A value of -1 indicates an error has occurred.

Errors:

NOT_OPENED

getSocket() - Get the File Descriptor of the ViewMax Socket Used

C++ **int Session.getSocket()**

Java **Socket Session.getSocket()**

Description:

GetSocket gets the file descriptor/Socket of the currently used ViewMax 2000 socket. In C++ the socket's descriptor can be used in a select() call to wait for a reply from a submit(). Once a reply has been received from the ViewMax session the collect() method can be used to pick up the data. However, in theory without knowledge of the underlying ViewMax Socket API this would be a difficult thing to use.

Return values:

On successful completion the socket number associated with the connected Viewmax session will be returned. A value of -1 or in J++ null indicates an error has occurred.

Errors:

NOT_CONNECTED

getUniqueid() - Get the Unique Id

C++ **int Session.getUniqueid()**

Java **int Session.getUniqueid()**

Description:

getUniqueid gets the Unique Id used by ViewMax to identify the ViewMax Server session that this Session Class is communicating with.

Return values:

On completion a zero will be returned

open() - Open a Session to a ViewMax Server

C++ **int Session.open(char *vmHost, char *vmPort, char *vmProject, int timeout)**

Java **int Session.open(String vmHost, String vmPort, String vmProject, int timeout)**

Description:

open() attempts to connect to a ViewMax server running on a host vmHost using TCP/IP port vmPort. If successful, a ViewMax session will be established on the server for the project specified in vmProject.

The timeout parameter sets the default timeout in seconds that an API request to the ViewMax session will wait before exiting with a TIMED_OUT error. Once a session has been opened the default timeout can also be changed dynamically using the setTimeout() method. **Note:** any timeout value specified in a VScript command will override the default timeout value specified for a session.

Parameters:

vmHost The hostname of the ViewMax 2000 server
VmPort The service port the ViewMax 2000 server is listening on
VmProject The name of the project configured on the server
timeout The time in seconds to wait before receiving a timeout error

Return values:

On successful completion the new session's ID will be returned. A value of -1 indicates an error has occurred.

Errors:

CONFIG_ERROR
INVALID_PROJECT
ALREADY_OPEN
CONNECT_ERROR
WRITE_ERROR
READ_ERROR
TIMED_OUT

putField() - Create a ViewMax Input Field

C++ **int Session.putField(const char *name, const char *value)**

Java **int Session.putField(String name, String value)**

Description:

Each ViewMax session that has been opened contains a field space which can contain field name/value pairs. PutField(name, value) creates a ViewMax input field called name and loads it with value.

Subsequent calls to putField() will create further input fields associated with the session. If putField() is called with a name that already exists, the field value will be overwritten by the new value.

The input fields created are analogous to the input fields defined within the form of an HTML page.

Parameters:

name name of the variable to create in the sessions field space.
value the value to load into the variable specified by name.

Return values:

On successful completion a zero will be returned

Errors:

NOT_OPENED - This error code will be set as a warning but will not cause a bad return.

setDebugLevel() - Set the Debug Level

C++ **int Session.setDebuglevel(int level)**

Java **int Session.setDebuglevel(int level)**

Description:

SetDebuglevel() sets the debug trace level of the ViewMax API. The level is initially set to 0 (no trace). Levels 1-5 can be used to get increased levels of debug.

Return values:

On completion a zero will be returned.

setTimeout() - Set the Default Timeout

C++ **int Session.setTimeout(int timeout)**

Java **int Session.setTimeout(int timeout)**

Description:

setTimeout() sets the default timeout in seconds that an API request to the ViewMax session will wait before exiting with a TIMED_OUT error. **Note:** any timeout value specified in a VScript command will override the default timeout value specified for a session.

Return values:

On successful completion a zero will be returned. A value of -1 indicates an error has occurred.

Errors:

NOT_OPENED
INVALID_TIMEOUT

setUniqueId() - Set the Unique Id

C++ **int Session.setUniqueId(int id)**

Java **int Session.setUniqueId(int id)**

Description:

setUniqueId sets the Unique Id used by ViewMax to identify the ViewMax Server session that this Session Class will communicate with. This Unique Id must be a valid Id sent by a ViewMax session following a disconnect() call.

Return values:

On completion a zero will be returned

submit() - Submit Fields to a connected ViewMax 2000 Session

C++ **int Session.submit(char *script)**

Java **int Session.submit(String script)**

Description:

Submit sends any ViewMax input fields previously loaded by the putField method and sends them to a previously opened ViewMax session. Along with the fields, the name of a ViewMax VScript, specified in script, is also passed to the session.

Parameters:

script The name of a project VScript script file.

Return values:

On successful completion a zero will be returned. A value of -1 indicates an error has occurred.

Errors:

NOT_CONNECTED
ALREADY_SUBMITTED
WRITE_ERROR

API Program Examples

C++ Example

Dos.cpp

// This example program uses a ViewMax 2000 project that was originally configured for Web access.
// The project has the ability logon to a remote system with a dos emulation and run a simple get system id, time and date.
// The project has three main scripts which are used in this program; login, menu_1 and logoff.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include "vmapi.h"
#include <string.h>
#include <time.h>
#include <stdarg.h>

BOOL WINAPI ControlHandler ( DWORD dwCtrlType );
extern int WSA_init();

#define MAX_SESSIONS 1
#define DEBÜG_LEVEL 0
#define MAX_LOOPS 1
#define MAX_MENU_LOOPS 1

FILE *debug_file=NULL;
void debug(char *format, ...);

/*****
main starts here
*****/
main(int argc, char **argv)
{
    int *vmSessionId;
    int *uniqueID;
    int *vmSocket;
    int i, j, k;
    int viewmax_sessions = MAX_SESSIONS;
    int maximum_number_of_loops = MAX_LOOPS;
    int menu_loops=MAX_MENU_LOOPS;
    const char *field;
    int error_code;

    char *viewmax_host = "localhost";

    //
    // Initialise the Winsock2 interface.
    //
    int wsa_err;
    if ((wsa_err=WSA_init()) != 0) {
        debug("wsa_init reports error %d, WSAGetLastError() = %d\n", wsa_err,
            WSAGetLastError());
        Sleep(10000);
        return(0);
    }

    //
    // set up the Control Handler.
    //
    SetConsoleCtrlHandler( ControlHandler, TRUE );
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
//
// If we have been run with Arguments use them.
//
if (argc > 1) {
    viewmax_host = argv[1];
}

if (argc > 2) {
    viewmax_sessions = atoi(argv[2]);
}

//
// Dynamically create the Control structures.
//
vmSessionId = new int[viewmax_sessions];
uniqueId = new int[viewmax_sessions];
vmSocket = new int[viewmax_sessions];

//
// Set the debug level
//
Session::setDebuglevel(DEBUG_LEVEL);

//
// Create a number of ViewMax 2000 session object
//
Session **vmSession = new Session*[viewmax_sessions];
for (i = 1; i <= viewmax_sessions; i++) {
    vmSession[i] = new Session();
    uniqueId[i] = 0;
}

// set debug
vmSession[0]->setDebuglevel(DEBUG_LEVEL);

for(j = 0; j < maximum_number_of_loops; j++) {
    debug("Attempt %d to Open %d ViewMax sessions\n", j, viewmax_sessions);
    debug("-----\n");
    for(i = 1; i <= viewmax_sessions; i++) {

        uniqueId[i] = 0;

        // open a ViewMax session
        if((vmSessionId[i] = vmSession[i]->open( viewmax_host , "viewmax" , "dos"
                                                , 120) ) < 0) {
            debug("Failed to open a ViewMax 2000 session %d\n", i);
            debug("ViewMax error message = %s\n", vmSession[i]->getErrorMsg());
            WSACleanup();
            Sleep(36000000);
            exit(1);
        }
        vmSocket[i] = vmSession[i]->getSocket();
        debug("ViewMax session %d established, session id = %d, socket number=
              %d\n", i, vmSessionId[i], vmSocket[i]);
    }

    // now logon all the sessions
    debug("Logon all session\n");
    debug("-----\n");
    for(i = 1; i <= viewmax_sessions; i++) {

        debug("Logging on session %d, script = logon\n", i);
    }
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
if(vmSession[i]->submit("login") < 0) {
    debug("logon failed to submit fields\n");
    WSACleanup();
    Sleep(36000000);
    exit(1);
}

int status;
if((status = vmSession[i]->collect()) < 0) {
    const char *error, *errorMessage;
    vmSession[i]->getField("error", error);
    vmSession[i]->getField("errorMessage", errorMessage);
    if (error == NULL) {
        error = "NULL";
    }
    if (errorMessage == NULL) {
        errorMessage = "NULL";
    }

    printf("Logon failed to collect fields error = %s, errorMessage =
        %s\n", error, errorMessage);
    WSACleanup();
    Sleep(36000000);
    exit(1);
}

//
// get the return values.
//
debug("login returned from the submit, link = %s\n",
    vmSession[i]->getLink());
error_code = vmSession[i]->getField("version", field);
debug("login returned from the submit, version = %s\n", field);
error_code = vmSession[i]->getField("date", field);
debug("login returned from the submit, date = %s\n", field);
error_code = vmSession[i]->getField("time", field);
debug("login returned from the submit, time = %s\n", field);
error_code = vmSession[i]->getField("value1", field);
debug("login returned from the submit, value1 = %s\n", field);

switch(status) {
    case Session::CONTINUE_REQUEST :
        debug("dos: received CONTINUE from vmaxd\n");
        break;

    case Session::CLOSE_REQUEST : {
        debug("dos: received CLOSE_REQUEST from vmaxd\n");
        if(vmSession[i]->close() == -1) {
            debug("logon failed to close the session, error code = %d\n",
                vmSession[i]->getErrorCode());
            debug("logon ViewMax error message = %s\n",
                vmSession[i]->getErrorMsg());
            WSACleanup();
            Sleep(36000000);
            exit(1);
        }
        break;
    }

    default :
        debug("dos: default\n");
        break;
}
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
debug("menu_loops has been set to %d\n", menu_loops);
for(k=0;k<menu_loops;k++) {
    debug("menu_1 comand %d for session %d, script = menu_1\n", k, i);
    if(vmSession[i]->submit("menu_1") < 0) {
        debug("menu_1 submit fields\n");
        WSACleanup();
        Sleep(36000000);
        exit(1);
    }

    int status;
    if((status = vmSession[i]->collect())< 0) {
        const char *error, *errorMessage;
        vmSession[i]->getField("error", error);
        vmSession[i]->getField("errorMessage", errorMessage);
        if (error == NULL) {
            error = "NULL";
        }
        if (errorMessage == NULL) {
            errorMessage = "NULL";
        }
        printf("menu_1 failed to collect fields error = %s, errorMessage =
            %s\n", error, errorMessage);
        WSACleanup();
        Sleep(36000000);
        exit(1);
    }

    //
    // get the return values.
    //
    debug("menu_1 returned from the submit, link = %s\n",
        vmSession[i]->getLink());
    error_code = vmSession[i]->getField("version", field);
    debug("menu_1 returned from the submit, version = %s\n", field);
    error_code = vmSession[i]->getField("date", field);
    debug("menu_1 returned from the submit, date = %s\n", field);
    error_code = vmSession[i]->getField("time", field);
    debug("menu_1 returned from the submit, time = %s\n", field);
    error_code = vmSession[i]->getField("value1", field);
    debug("menu_1 returned from the submit, value1 = %s\n", field);

    switch(status) {
        case Session::CONTINUE_REQUEST :
            debug("dos: received CONTINUE from vmaxd\n");
            break;

        case Session::CLOSE_REQUEST :
            debug("dos: received CLOSE_REQUEST from vmaxd\n");
            if(vmSession[i]->close() == -1) {
                debug("menu_1 failed to close the session, error code = %d\n",
                    vmSession[i]->getErrorCode());
                debug("menu_1 ViewMax error message = %s\n",
                    vmSession[i]->getErrMsg());
                WSACleanup();
                Sleep(36000000);
                exit(1);
            }
            break;

        default :
            debug("dos: default\n");
            break;
    }
}
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
// now disconnect all the sessions
debug("Disconnect all session\n");
debug("-----\n");

for(i = 1; i <= viewmax_sessions; i++) {

    debug("Disconnecting session %d\n", i);

    if((uniqueId[i] = vmSession[i]->disconnect()) < 0) {
        debug("logoff failed to disconnect fields\n");
        WSACleanup();
        Sleep(36000000);
        exit(1);
    }
}

// now connect all the sessions
debug("Connect all session\n");
debug("-----\n");

for(i = 1; i <= viewmax_sessions; i++) {

    debug("Connecting session %d\n", i);

    if(vmSession[i]->connect(viewmax_host, vmSessionId[i],uniqueId[i]) < 0) {
        debug("logoff failed to disconnect fields\n");
        WSACleanup();
        Sleep(36000000);
        exit(1);
    }
}

// now submit menu_1 all the sessions
debug("submit menu_1 on all session\n");
debug("-----\n");

for(i = 1; i <= viewmax_sessions; i++) {
    debug("menu_loops has been set to %d\n", menu_loops);
    for(k=0;k<menu_loops;k++) {
        debug("menu_1 comand %d for session %d, script = menu_1\n", k, i);
        if(vmSession[i]->submit("menu_1") < 0) {
            debug("menu_1 submit fields\n");
            WSACleanup();
            Sleep(36000000);
            exit(1);
        }
    }

    int status;
    if((status = vmSession[i]->collect())< 0) {
        const char *error, *errorMessage;
        vmSession[i]->getField("error", error);
        vmSession[i]->getField("errorMessage", errorMessage);
        if (error == NULL) {
            error = "NULL";
        }
        if (errorMessage == NULL) {
            errorMessage = "NULL";
        }
        printf("menu_1 failed to collect fields error = %s, errorMessage =
            %s\n", error, errorMessage);
        WSACleanup();
        Sleep(36000000);
        exit(1);
    }
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
//
// get the return values.
//
debug("menu_1 returned from the submit, link = %s\n",
      vmSession[i]->getLink());
error_code = vmSession[i]->getField("version", field);
debug("menu_1 returned from the submit, version = %s\n", field);
error_code = vmSession[i]->getField("date", field);
debug("menu_1 returned from the submit, date = %s\n", field);
error_code = vmSession[i]->getField("time", field);
debug("menu_1 returned from the submit, time = %s\n", field);
error_code = vmSession[i]->getField("value1", field);
debug("menu_1 returned from the submit, value1 = %s\n", field);

switch(status) {
  case Session::CONTINUE_REQUEST :
    debug("dos: received CONTINUE from vmamd\n");
    break;

  case Session::CLOSE_REQUEST :
    debug("dos: received CLOSE_REQUEST from vmamd\n");
    if(vmSession[i]->close() == -1) {
      debug("menu_1 failed to close the session, error code = %d\n",
            vmSession[i]->getErrorCode());
      debug("menu_1 ViewMax error message = %s\n",
            vmSession[i]->getErrMsg());
      WSACleanup();
      Sleep(36000000);
      exit(1);
    }
    break;

  default :
    debug("dos: default\n");
    break;
}
}
}

// now logoff all the sessions
debug("Logoff all session\n");
debug("-----\n");

for(i = 1; i <= viewmax_sessions; i++) {

  debug("Logging off session %d, script = logoff\n", i);

  if(vmSession[i]->submit("logoff") < 0) {
    debug("logoff failed to submit fields\n");
    WSACleanup();
    Sleep(36000000);
    exit(1);
  }

  int status;
  if((status = vmSession[i]->collect()) < 0) {
    const char *error, *errorMessage;
    vmSession[i]->getField("error", error);
    vmSession[i]->getField("errorMessage", errorMessage);
    if (error == NULL) {
      error = "NULL";
    }
    if (errorMessage == NULL) {
      errorMessage = "NULL";
    }
    printf("logoff failed to collect fields error = %s, errorMessage =
           %s\n", error, errorMessage);
  }
}
```

**ViewMax 2000 API
API PROGRAM EXAMPLES**

```

        WSACleanup();
        Sleep(36000000);
        exit(1);
    }

    debug("returned from the submit, link = %s\n", vmSession[i]->getLink());

    switch(status) {
        case Session::CONTINUE_REQUEST :
            debug("dos: received CONTINUE from vmaxd\n");
            break;

        case Session::CLOSE_REQUEST :
            debug("dos: received CLOSE_REQUEST from vmaxd\n");
            if(vmSession[i]->close() == -1) {
                debug("failed to close the session, error code = %d\n",
                    vmSession[i]->getErrorCode());
                debug("ViewMax error message = %s\n",
                    vmSession[i]->getErrorMsg());
                WSACleanup();
                Sleep(36000000);
                exit(1);
            }
            break;

        default :
            debug("dos: default\n");
            break;
    }
}
}
WSACleanup();
debug("Finished all my loops\n");
Sleep(10000);
return(0);
}

void debug(char *format, ...)
{
    if (!debug_file && DEBUG_LEVEL) {
        debug_file = fopen("c:\\vtest.dbg", "a");
    }

    time_t timeval;
    struct tm *newtime;
    char buf[2048];
    char dbuf[2048];
    char thread_id[20];
    char time_date[30];

    va_list ap;
    va_start(ap, format);

    timeval = time(NULL);
    newtime = localtime(&timeval); /* Convert time to struct */
    int len = _snprintf(buf, 30, "%s", asctime(newtime));
    buf[len-1] = '\0';
    (void)_snprintf(thread_id, 20, "<%d>", GetCurrentThreadId());
    thread_id[19] = '\0';
    (void)_snprintf(time_date, 30, "<%s> ", buf);
    time_date[29] = '\0';
    (void)_vsnprintf(buf, 2048, format, ap);
    buf[2047] = '\0'; // just in case string bigger than 2K
    _snprintf(dbuf, 2048, "%s %s %s", thread_id, time_date, buf);

```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
if (debug_file && DEBUG_LEVEL) {
    (void)fprintf(debug_file, "%s", dbuf);
    (void)fflush(debug_file);
}
printf("%s", dbuf);
fflush(stdout);

va_end(ap);
return;
}
//
// FUNCTION: ControlHandler ( DWORD dwCtrlType )
//
// PURPOSE: Handled console control events
//
// PARAMETERS:
//     dwCtrlType - type of control event
//
// RETURN VALUE:
//     True - handled
//     False - unhandled
//
// COMMENTS:
//
//
BOOL WINAPI ControlHandler ( DWORD dwCtrlType )
{
    char *ThreadName = "vtest:ControlHandler";
    BOOL KnownEvent = TRUE;

    debug("ControlHandler started\n");

    switch( dwCtrlType )
    {
        case CTRL_C_EVENT: // SERVICE_CONTROL_STOP in debug mode
            printf("Console Event CTRL_C_EVENT received. Shutting down\n");
            debug("Console Event CTRL_C_EVENT received. Shutting down\n");
            break;
        case CTRL_BREAK_EVENT: // use Ctrl+C or Ctrl+Break to simulate
            printf("Console Event CTRL_BREAK_EVENT received. Shutting down\n");
            debug("Console Event CTRL_BREAK_EVENT received. Shutting down\n");
            break;
        case CTRL_CLOSE_EVENT:
            printf("Console Event CTRL_CLOSE_EVENT received. Shutting down\n");
            debug("Console Event CTRL_CLOSE_EVENT received. Shutting down\n");
            break;
        case CTRL_LOGOFF_EVENT:
            printf("Console Event CTRL_LOGOFF_EVENT received. Not Shutting
                down\n");
            debug("Console Event CTRL_LOGOFF_EVENT received. Not Shutting
                down\n");
            KnownEvent = FALSE;
            break;
        case CTRL_SHUTDOWN_EVENT:
            printf("Console Event CTRL_SHUTDOWN_EVENT received. Shutting
                down\n");
            debug("Console Event CTRL_SHUTDOWN_EVENT received. Shutting down\n");
            break;
        default:
            printf("Console Control Event %x not handled\n", dwCtrlType);
            KnownEvent = FALSE;
            break;
    }
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
}  
WSACleanup();  
printf("Cleanup complete\n");  
exit(0);  
return FALSE;  
}
```

J++ Example

Dos.java

// This example program uses a ViewMax 2000 project that was originally configured for Web access.
// The project has the ability logon to a remote system with a dos emulation and run a simple get system id, time and date.
// The project has three main scripts which are used in this program; login, menu_1 and logoff.

```
import java.*;
import java.net.*;
import viewmaxAPI.*;

public class dos {
    static int vmaxdId;
    static int uniqueId;
    static int status;
    static int error_code;
    static int level=0;
    static Socket vmSocket;
    static StringBuffer field;
    static Session vmaxd;
    static StringBuffer error;
    static StringBuffer errorMessage;

    static String viewmax_host = new String("localhost");

    public static void main(String args[]) {
        vmaxd = new Session();
        // Set the debug level
        //
        vmaxd.setDebuglevel(level);

        uniqueId = 0;

        // open a ViewMax session
        if((vmaxdId = vmaxd.open( viewmax_host , "4000" , "dos" , 120) ) < 0) {
            System.out.print("Failed to open a ViewMax 2000 session \n");
            System.out.print("ViewMax error message = "+vmaxd.getErrorMsg()+"\n");
        }
        vmSocket = vmaxd.getSocket();
        System.out.print("ViewMax session established, session id = "+vmaxdId+"
            socket number= "+vmSocket+"\n");

        // now logon all the sessions
        System.out.print("Logon all session\n");
        System.out.print("-----\n");

        if(vmaxd.submit("login") < 0) {
            System.out.print("logon failed to submit fields\n");
        }

        if((status = vmaxd.collect())< 0) {
            error = new StringBuffer();
            errorMessage = new StringBuffer();
            vmaxd.getField("error", error);
            vmaxd.getField("errorMessage", errorMessage);
            if (error == null) {
                error = new StringBuffer("null");
            }
            if (errorMessage == null) {
                errorMessage = new StringBuffer("null");
            }
        }
        System.out.print("Logon failed to collect fields error =
            "+error.toString()+", errorMessage = "+errorMessage.toString()+"\n");
    }
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
}

//
// get the return values.
//

System.out.print("login returned from the submit, link =
    "+vmaxd.getLink()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("version", field);
System.out.print("login returned from the submit, version =
    "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("date", field);
System.out.print("login returned from the submit, date =
    "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("time", field);
System.out.print("login returned from the submit, time =
    "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("value1", field);
System.out.print("login returned from the submit, value1 =
    "+field.toString()+"\n");

switch(status) {
    case vmaxd.CONTINUE_REQUEST :
        System.out.print("dos: received CONTINUE from vmaxd\n");
        break;

    case vmaxd.CLOSE_REQUEST : {
        System.out.print("dos: received CLOSE_REQUEST from vmaxd\n");
        if(vmaxd.close() == -1) {
            System.out.print("logon failed to close the session, error code =
                "+vmaxd.getErrorCode()+"\n");
            System.out.print("logon ViewMax error message =
                "+vmaxd.getErrorMsg()+"\n");
        }
        break;
    }

    default :
        System.out.print("dos: default\n");
        break;
}

if(vmaxd.submit("menu_1") < 0) {
    System.out.print("menu_1 submit fields\n");
}

if((status = vmaxd.collect()) < 0) {
    error = new StringBuffer();
    errorMessage = new StringBuffer();
    vmaxd.getField("error", error);
    vmaxd.getField("errorMessage", errorMessage);
    if (error == null) {
        error = new StringBuffer("null");
    }
    if (errorMessage == null) {
        errorMessage = new StringBuffer("null");
    }
    System.out.print("menu_1 failed to collect fields error =
        "+error.toString()+", errorMessage =
        "+errorMessage.toString()+"\n");
}
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
//
// get the return values.
//
System.out.print("menu_1 returned from the submit, link =
                "+vmaxd.getLink()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("version", field);
System.out.print("menu_1 returned from the submit, version =
                "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("date", field);
System.out.print("menu_1 returned from the submit, date =
                "+field.toString()+"\n");
field = new StringBuffer();

error_code = vmaxd.getField("time", field);
System.out.print("menu_1 returned from the submit, time =
                "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("value1", field);
System.out.print("menu_1 returned from the submit, value1 =
                "+field.toString()+"\n");

switch(status) {
  case vmaxd.CONTINUE_REQUEST :
    System.out.print("dos: received CONTINUE from vmaxd\n");
    break;

  case vmaxd.CLOSE_REQUEST :
    System.out.print("dos: received CLOSE_REQUEST from vmaxd\n");
    if(vmaxd.close() == -1) {
      System.out.print("menu_1 failed to close the session, error code =
                      "+vmaxd.getErrorCode()+"\n");
      System.out.print("menu_1 ViewMax error message =
                      "+vmaxd.getErrorMsg()+"\n");
    }
    break;

  default :
    System.out.print("dos: default\n");
    break;
}

// now disconnect all the sessions
System.out.print("Disconnect all session\n");
System.out.print("-----\n");

if((uniqueId = vmaxd.disconnect()) < 0) {
  System.out.print("logoff failed to disconnect fields\n");
}

// now connect all the sessions
System.out.print("Connect all session\n");
System.out.print("-----\n");

if(vmaxd.connect(viewmax_host, vmaxdId, uniqueId) < 0) {
  System.out.print("logoff failed to disconnect fields\n");
}

// now submit menu_1 all the sessions
System.out.print("submit menu_1 on all session\n");
System.out.print("-----\n");

if(vmaxd.submit("menu_1") < 0) {
  System.out.print("menu_1 submit fields\n");
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
}

if((status = vmaxd.collect()) < 0) {
    error = new StringBuffer();
    errorMessage = new StringBuffer();
    vmaxd.getField("error", error);
    vmaxd.getField("errorMessage", errorMessage);
    if (error == null) {
        error = new StringBuffer("null");
    }
    if (errorMessage == null) {
        errorMessage = new StringBuffer("null");
    }
    System.out.print("menu_1 failed to collect fields error =
                    "+error.toString()+", errorMessage =
                    "+errorMessage.toString()+"\n");
}

//
// get the return values.
//
System.out.print("menu_1 returned from the submit, link =
                "+vmaxd.getLink()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("version", field);
System.out.print("menu_1 returned from the submit, version =
                "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("date", field);
System.out.print("menu_1 returned from the submit, date =
                "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("time", field);
System.out.print("menu_1 returned from the submit, time =
                "+field.toString()+"\n");
field = new StringBuffer();
error_code = vmaxd.getField("value1", field);
System.out.print("menu_1 returned from the submit, value1 =
                "+field.toString()+"\n");

switch(status) {
    case vmaxd.CONTINUE_REQUEST :
        System.out.print("dos: received CONTINUE from vmaxd\n");
        break;

    case vmaxd.CLOSE_REQUEST :
        System.out.print("dos: received CLOSE_REQUEST from vmaxd\n");
        if(vmaxd.close() == -1) {
            System.out.print("menu_1 failed to close the session, error code =
                            "+vmaxd.getErrorCode()+"\n");
            System.out.print("menu_1 ViewMax error message =
                            "+vmaxd.getErrorMsg()+"\n");
        }
        break;

    default :
        System.out.print("dos: default\n");
        break;
}

// now logoff all the sessions
System.out.print("Logoff all session\n");
System.out.print("-----\n");

if(vmaxd.submit("logoff") < 0) {
    System.out.print("logoff failed to submit fields\n");
}
}
```

ViewMax 2000 API
API PROGRAM EXAMPLES

```
if((status = vmaxd.collect())< 0) {
    error = new StringBuffer();
    errorMessage = new StringBuffer();
    vmaxd.getField("error", error);
    vmaxd.getField("errorMessage", errorMessage);
    if (error == null) {
        error = new StringBuffer("null");
    }
    if (errorMessage == null) {
        errorMessage = new StringBuffer("null");
    }
    System.out.print("logout failed to collect fields error =
                    "+error.toString()+", errorMessage =
                    "+errorMessage.toString()+"\n");
}

System.out.print("returned from the submit, link = "+vmaxd.getLink()+"\n");

switch(status) {
    case vmaxd.CONTINUE_REQUEST :
        System.out.print("dos: received CONTINUE from vmaxd\n");
        break;

    case vmaxd.CLOSE_REQUEST :
        System.out.print("dos: received CLOSE_REQUEST from vmaxd\n");
        if(vmaxd.close() == -1) {
            System.out.print("failed to close the session, error code =
                            "+vmaxd.getErrorCode()+"\n");
            System.out.print("ViewMax error message =
                            "+vmaxd.getErrorMsg()+"\n");
        }
        break;

    default :
        System.out.print("dos: default\n");
        break;
}
}
```