MUSE CV® Information System API

Programmer's Guide

Software Version 005C & higher

2002783-021 Revision B

GE Medical Systems

Information Technologies

gemedical.com

NOTE: The information in this manual only applies to MUSE CV API software version 005C and higher. It does not apply to earlier software versions. Due to continuing product innovation, specifications in this manual are subject to change without notice.

Listed below are GE Medical Systems *Information Technologies* trademarks. All other trademarks contained herein are the property of their respective owners.

900 SC, ACCUSKETCH, AccuVision, APEX, AQUA-KNOT, ARCHIVIST, Autoseq, BABY MAC, C Qwik Connect, CardioServ, CardioSmart, CardioSys, CardioWindow, CASE, CD TELEMETRY, CENTRA, CHART GUARD, CINE 35, CORO, COROLAN, COROMETRICS, Corometrics Sensor Tip, CRG PLUS, DASH, Digistore, Digital DATAQ, E for M, EAGLE, Event-Link, FMS 101B, FMS 111, HELLIGE, IMAGE STORE, INTELLIMOTION, IQA, LASER SXP, MAC, MAC-LAB, MACTRODE, MANAGED USE, MARQUETTE, MARQUETTE MAC, MARQUETTE MEDICAL SYSTEMS, MARQUETTE UNITY NETWORK, MARS, MAX, MEDITEL, MEI, MEI in the circle logo, MEMOPORT, MEMOPORT C, MINISTORE, MINNOWS, Monarch 8000, MULTI-LINK, MULTISCRIPTOR, MUSE, MUSE CV, Neo-Trak, NEUROSCRIPT, OnlineABG, OXYMONITOR, Pres-R-Cuff, PRESSURE-SCRIBE, QMI, QS, Quantitative Medicine, Quantitative Sentinel, RAC RAMS, RSVP, SAM, SEER, SILVERTRACE, SOLAR, SOLARVIEW, Spectra 400, Spectra-Overview, Spectra-Tel, ST GUARD, TRAM, TRAM-NET, TRAM-RAC, TRAMSCOPE, TRIM KNOB, Trimline, UNION STATION, UNITY logo, UNITY NETWORK, Vari-X, Vari-X Cardiomatic, VariCath, VARIDEX, VAS, and Vision Care Filter are trademarks of GE Medical Systems *Information Technologies* registered in the United States Patent and Trademark Office.

12SL, 15SL, Access, AccuSpeak, ADVANTAGE, BAM, BODYTRODE, Cardiomatic, CardioSpeak, CD TELEMETRY®-LAN, CENTRALSCOPE, Corolation, EDIC, EK-Pro, Event-Link Cirrus, Event-Link Cumulus, Event-Link Nimbus, HI-RES, ICMMS, IMAGE VAULT, IMPACT.wf, INTER-LEAD, IQA, LIFEWATCH, Managed Use, MARQUETTE PRISM, MARQUETTE® RESPONDER, MENTOR, MicroSmart, MMS, MRT, MUSE CardioWindow, NST PRO, NAUTILUS, O_2SENSOR, Octanet, OMRS, PHi-Res, Premium, Prism, QUIK CONNECT V, QUICK CONNECT, QT Guard, SMART-PAC, SMARTLOOK, Spiral Lok, Sweetheart, UNITY, Universal, Waterfall, and Walkmom are trademarks of GE Medical Systems *Information Technologies*.

© GE Medical Systems *Information Technologies*, 2001. All rights reserved.

Contents

For your notes

Manual Information

Revision History

Each page of the document has the document part number followed by a revision letter at the bottom of the page. This letter identifies the document's update level. The latest letter of the alphabet corresponds to the most current revision of the document. The revision history of this document is summarized in the table below.

Table 1. Revision History PN 2002783-021		
Revision	**Date**	**Comment**
A	31 May 2000	Initial release of manual.
B	7 December 2001	Extended to software version 005C and higher.

Introduction

The first version of the MUSE API supports connecting and disconnecting from a MUSE database, translating a patient name into a PID, extracting a list of tests for a given patient, and generating an output report for a given patient and test. Future versions will enhance these base functions as well as add more API entry points.

Installation Requirements

- MUSE database: The MUSE API assumes an operational 005C (or greater) MUSE system database and database structure.
- Win32: The MUSE API is a set of 32-bit DLLs and assumes a true Win32 operating platform (WinNT, preferred, or Windows 95); Win32s is not supported.
- MUSE API DLL: The 32-bit API entry point DLL must be in the current path: MUSEAPI2.DLL.
- Microsoft runtime DLLs: The Microsoft runtime libraries, msvc*.dll, must be in the current path.
- MACCRAPS.DLL and MACCRA.TLB must be installed on the client using the standard windows program REGSVR32.DLL as: REGSVR32.EXE MACCRAPS.DLL.

API

Basic Data Types

The header file MUSEAPI.H defines data types used with the MUSE API entry points. Note: To preserve the structure alignment, C/C++ sources must be compiled with alignment on 1-byte boundaries. Structures defined in MUSEAPI.H include:

MUSEAPI_HANDLE:	A 32-bit value used by the MUSE API for internal variable storage.
MUSE_PNAME:	Patient name structure.
MUSE_PID:	Patient ID structure.
MUSE_DATE:	Date structure.
MUSE_TIME:	Time structure.
MUSE_DEMOGRAPHIC:	Patient information as stored in patients.btr.
MUSE_FILENAME:	Filename structure.
MUSE_TEST_INFORMATION:	Information about a given test (e.g., test date and time).

Entry Points

The return value of all MUSE API functions is the completion status of the function. If the function was successful, this value will be 0 (MUSE_SUCCESS). If the return value is anything other than 0, an error occurred in processing. Note: All MUSEAPI functions use the C function calling mechanism (aka cdecl). If you are using a programming language besides C/C++ make sure you declare the functions appropriately.

MEI_OPENSERVER()

Establishes a connection with the MUSE database on a specific server. This assumes MACCRA.EXE has been installed and properly configured on the MUSE server.

Arguments:

MUSEAPI_HANDLE *phMUSE — the address of the MUSE handle. Note that the address of the handle is passed, not the handle itself.

BYTE site — the site into the data base to which you are connecting. This value will be used to determine site in all API calls.

CHAR*pszServerName — the name of the MUSE server.

mei_OpenMUSE()

Establishes the connection with the MUSE database.

NOTE

This function is for legacy support, and should be replaced with MEI_OPENSERVER.

Arguments:

MUSEAPI_HANDLE *phMUSE — the address of the MUSE handle. Note that the address of the handle is passed, not the handle itself.

HWND hWndCaller — the caller's window handle. This may be NULL if a window is not used by the caller.

WORD wNodeID — the MUSE node that the task is running on. If the calling task is not running on a MUSE node, this should be a unique number as it is used in logging errors to the MUSE error log.

WORD wTaskID — task ID of the calling task. Again, this should be a unique number such that errors can be recognized when registered in the MUSE error log.

BOOL bBinding — for now, always set this to TRUE

BYTE site — the site into the data base to which you are connecting. This value will be used to determine site in all API calls.

Return Value:

MUSE_SUCCESS, if successful; error status from MUSE connection routines or MUSE_MEMORY_ALLOCATION_ERROR, if not.

Example Call:

```
MUSEAPI_HANDLE hMUSE ;
WORD status ;
status = mei_OpenMUSE(&hMUSE, NULL, 999, 9999, TRUE, 1) ;
if (status != MUSE_SUCCESS)
{
  wsprintf (errorBuffer, “Error Status %d”, status) ;
  MessageBox ((HWND)0, errorBuffer, “Error Connect To MUSE”, MB_OK) ;
  return 0 ;
}
```

mei_CloseMUSE()

Drops the connection to the MUSE database and frees any memory allocated by the MUSE API DLL.

Arguments:

MUSEAPI_HANDLE *phMUSE — the address of the MUSE handle. Note that the address is passed, not the handle itself.

BOOL bUnmapPrimaryDrive — a boolean stating whether to unmap the primary network drive. If true, the primary network drive will be released; if false, the primary drive will not be released.

Return Value:

MUSE_SUCCESS, if successful;
MUSE_APIHANDLE_NOT_INITIALIZED, if not.

Example Call:

```
mei_CloseMUSE(&hMUSE, FALSE) ;
```

mei_PatientIDFromName()

Translates a patient name (as stored in a MUSE_PNAME structure) into a patient ID (in a MUSE_PID structure). In order to provide a means for the calling application to resolve duplicate names, an array of patient demographic structures is returned. That is, the caller supplies an array of MUSE_DEMOGRAPHIC structures and the API will fill the buffer with matching demographic information based on the passed in patient name. The buffer will be filled with the *pNumberOfEntries that most closely match the name.

Arguments:

MUSEAPI_HANDLE hMUSE — The MUSE handle.

MUSE_PNAME *pPatientName — the patient name. A complete name must be provided. See table on page 6 for usage.

MUSE_DEMOGRAPHIC *pDemoBuffer — an array of MUSE_DEMOGRAPHICS. This is a pointer to the caller's buffer.

unsigned short *pNumberOfEntries — number of entries in pDemoBuffer. On input, this represents the size of pDemoBuffer in entries; on output, the MUSE API sets this to be the number of entries filled in by the MUSE API.

MUSE_PID *pPID – the patient ID. (See table on page 6 for usage.)

Return Value:

MUSE_SUCCESS, if successful; Btrieve error status or
MUSE_APIHANDLE_NOT_VALID, if not.

Example Call:

```
#define MAX_DEMOGRAPHICS 10
MUSE_DEMOGRAPHIC demographicBuffer[MAX_DEMOGRAPHICS] ;
  strcpy (aPName.last, "PREGO") ;
  strcpy (aPName.first, “Maria”) ;
  numEntries = MAX_DEMOGRAPHICS ;
  status = mei_PatientIDFromName(hMUSE, &aPName, demographicBuffer,
                                 &numEntries, &aPID) ;
  if (status != MUSE_SUCCESS || numEntries==0)
  {
    wsprintf (errorBuffer, "Error Status %d: entries %d", status,
              numEntries) ;
    MessageBox ((HWND)0, errorBuffer, "Error getting patient", MB_OK) ;
    mei_CloseMUSE(&hMUSE) ;
    return 0 ;
  }
  else if (numEntries>1)
    doSomePruningToFindID(demographicBuffer, numEntries) ;
    /* If name is not found, call mei_PatientIDFromName with
       demographicBuffer[numEntries-1].name to get the next numEntries
     */
```

Input / Output Matrix

Patient Last Name	Patient FirstName	Patient ID	Respond with:
“”	“”	“”	1. First “n” patients listed in last name alphabetical order.
Defined	“”	“”	2. First “n” patients which partially/ completely match last name in last name alphabetical order.
Defined	Defined	“”	3. Same as 2 above except starts with: <lastname><firstname>
Defined	“”	Defined	4. Same as 2.
Defined	Defined	Defined	5. Same as 3 except starts with: <lastname><firstname><PID>
“”	Defined	“”	6. Same as 1.
“”	Defined	Defined	7. Same as 1.
“”	“”	Defined	8. Same as 1.

mei_PatientNameFromID()

Given a valid MUSE_PID, returns the appropriate MUSE_PNAME.

Arguments:

MUSEAPI_HANDLE hMUSE — The MUSE handle.

MUSE_PNAME *pPatientName — the patient name is returned in this buffer

MUSE_PID *pPatientPID — the patient ID.

Return Value:

MUSE_SUCCESS, if successful; Btrieve error status if key is not found (ENOTKEY or EEOF).

Example Call:

```
MUSE_PNAME aPName ;
MUSE_PID aPID ;
strcpy (aPID.id, “007”) ;
status = mei_PatientNameFromID(hMUSE, &aPName, &aPID) ;
if (status != MUSE_SUCCESS || stricmp (aPName.last, “Bond”)!=0)
  MessageBox ((HWND)0, “007 is missing”, “Attention”, MB_OK) ;
```

mei_TestsForPatient()

Given a valid MUSE_PID, fills an array of MUSE_TEST_INFORMATION entries with tests for the patient. The caller can request tests only for a single data type (e.g., resting ECG vs

Holter) or all tests for the patient. In addition, the caller may specify the most recent tests or only those tests before a given date and time.

NOTE

The stucture, MUSE_TEST_INFORMATION, was modified between software versions 005B and 005C.

Arguments:

MUSEAPI_HANDLE hMUSE — The MUSE handle.

MUSE_PID *pPID — the patient ID.

BYTE cseType — (see MUSEAPI.H for numerical #define-itions). The signal record type for the requested type. For example, resting ECG, Holter, Exercise testing, etc.

MUSE_DATE *pFromDate — date from which to begin accumulation of tests. If all fields in pFromDate are set to 0xFF, the most recent tests will be obtained. (See table on page 8 for usage.)

MUSE_TIME *pFromTime — time from which to begin accumulation of tests. Set to 0xFF if pFromDate is set to 0xFF. (See table on page 8 for usage.)

MUSE_TEST_INFORMATION *pTestBuffer — an array of MUSE_TEST_INFORMATION. This is a pointer to the caller's buffer.

unsigned short *pNumberOfEntries — number of entries in pTestBuffer. On input, this represents the maximum number of possible entries in pTestBuffer; on output, the MUSE API sets this to be the number of entries actually filled in.

Return Value:

MUSE_SUCCESS, if successful; Btrieve error status or MUSE_APIHANDLE_NOT_VALID, if not.

Example Call:

```
#define MAX_TESTINFOS    10
TEST_INFORMATION testInfo[MAX_TESTINFOS] ;

// Get the 10 most recent tests for our patient.
numEntries = MAX_TESTINFOS ;
memcpy (&aPID, &demographicBuffer[0].patient, sizeof(MUSE_PID)) ;
memset (&aDate, 0xFF, sizeof(MUSE_DATE)) ;
memset (&aTime, 0xFF, sizeof(MUSE_TIME)) ;
status = mei_TestsForPatient(hMUSE, &aPID, CSE_ALLTESTS, &aDate, &aTime,
                             testInfo, &numEntries) ;
if (status != MUSE_SUCCESS || numEntries==0)
{
  wsprintf (errorBuffer, "Error Status %d: entries %d", status, numEntries) ;
  MessageBox ((HWND)0, errorBuffer, "Error getting tests", MB_OK) ;
  mei_CloseMUSE(&hMUSE) ;
  return 0 ;
}
else if (numEntries>1)
  doSomePruningToFindTest(testInfo, numEntries) ;
```

Input / Output Matrix

Start Date	Start Time	Respond with:
ALL	ALL	1. the “n” most recent tests.
Defined	ALL	2. Same as 1, but only tests from the designated date are considered.
Defined	Defined	3. “n” tests from the designated day, starting with (and including) the specified time.
ALL	Defined	4. Undefined. DO NOT USE!

mei_CreateOutputForTestInfo()

Generate an output report for a given MUSE_TEST_INFORMATION. Currently supported output types include OT_POSTSCRIPT, an Adobe Level 2 PostScript file; OT_WMF, a set of Windows Metafile (not an enhanced metafile) stored in an MEI bitmap directory (see external documentation of MEI bitmap directory); OT_FAX, a set of Group 3 Compliant Fax images stored in an MEI bitmap directory; OT_JPEG, a set of JPEG compressed images stored in an MEI bitmap directory.

Arguments:

MUSEAPI_HANDLE hMUSE — The MUSE handle.

MUSE_PID *pPID — the patient ID.

MUSE_TEST_INFORMATION *pTestBuffer — information about the test to format

BYTE outputType — type of output (see MUSEAPI.H for #define-initions)

char *pOutputFileName — destination file name for formatted report.

Return value:

MUSE_SUCCESS, if successful; Btrieve error or MUSE_APIHANDLE_NOT_VALID, if not

Example Call:

```
status = mei_CreateOutputForTestInfo(hMUSE, &aPID, testInfo, OT_WMF,
                                                    "C:\\output.wmf") ;
if (status != MUSE_SUCCESS)
{
  wsprintf (errorBuffer, "Error Status %d", status) ;
  MessageBox ((HWND)0, errorBuffer, "Error formatting report", MB_OK) ;
  mei_CloseMUSE(&hMUSE) ;
  return 0 ;
}
```

mei_CreateOutputForID()

A direct mechanism for generating an output report. The same output types are supported as for mei_CreateOutputForTestInfo().

Arguments:

MUSEAPI_HANDLE hMUSE — The MUSE handle.

MUSE_PID *pPID — the patient ID.

MUSE_DATE *pDate — test date. If all fields in pDate are set to 0xFF, the current report will be formatted.

MUSE_TIME *pTime — test time. If all fields in pTime are set to 0xFF, the current report will be formatted.

BYTE cseType — (see MUSEAPI.H for numerical #define-ition). The signal record type for the requested type. For example, resting ECG, Holter, Exercise testing, etc.

BYTE outputType — type of output (see MUSEAPI.H for #define-initions)

char *pOutputFileName — destination file name for formatted report.

Return Value:

MUSE_SUCCESS, if successful; Btrieve error calls, or MUSE_APIHANDLE_NOT_VALID, if not.

Example Call:

```
status = mei_CreateOutputForID(hMUSE, &aPID, &aDate, &aTime, CSE_RESTING,
                               OT_POSTSCRIPT, "C:\\output.ps") ;
if (status != MUSE_SUCCESS)
{
  wsprintf (errorBuffer, "Error Status %d", status) ;
  MessageBox ((HWND)0, errorBuffer, "Error formatting report", MB_OK) ;
  mei_CloseMUSE(&hMUSE) ;
  return 0 ;
}
```

Output File Formats

The output types fall into three classes: Vector, Raster, and Text. Supported output types for the initial release of the MUSE CV-API include Adobe Level 2 PostScript, Group 3 Fax, JPEG, TIFF, HP PCL 5, Windows Enhanced Metafiles, Windows Bitmap's (.BMP's), ASCII Text, HTML, and Adobe Portable Document Format (PDF).

All image outputs except for Adobe PostScript and Portable Document Format and TIFF use the MMS Bitmap Directory to store the output images, one image for each page of the output. The MMS Bitmap Directory contains an 8K header at the beginning of the file (see definition at the end of this document) that indicates the number of pages included in the file, as well as information about each page (e.g., offset of the start of the image in the file, size of the image in the file, whether the image is portrait or landscape mode, the dimensions of the image, etc.). The MMS Bitmap Directory is required on the various output types because they do not inherently support multiple page documents. Each image pointed to by the directory header is stored according to the standards for the given type, i.e. as a programmer you can extract the information for a given page and pass it directly to an "off-the-shelf" viewing program without any additional processing.

The two text formats- ASCII and HTML- do not utilize the bitmap header since they do not include any graphical information.

Vector Formats

PostScript

The MUSE CV-API generates a single output file, suitable for transmission to an Adobe Level 2 PostScript printer. With a minor exception regarding the documentation of fonts used in the document, the output file strictly follows Adobe Document Structuring conventions. As such, public domain utilities such as psnup can be used with the generated files and PostScript viewers such as GhostView should properly display the image.

Portable Document Format

The MUSE CV-API generates a single output file, suitable for use with the Adobe Acrobat Reader.

This format is a Vector representation of the report and produces excellent results when viewed on screen or printed, since the Acrobat Reader is capable of using the maximal resolution of the rendering device. The Adobe Acrobat Reader is available for free from Adobe Systems, Inc., at http://www.adobe.com.

Windows Enhanced Metafile

The MUSE CV-API generates Windows Enhanced Metafiles and stores them in an MMS Bitmap Directory. The metafiles generated are scaled to fit on a display page of 27940 pixels (Horizontal) by 21950 pixels (Vertical), landscape mode, and 21950(H)x27940(V), portrait mode. To display the metafiles in a Windows application, the coordinates of the window must be resized to the logical coordinates through the calls:

```
SetMapMode(MM_ANISOTROPIC) ;
SetWindowExt(27940, 21950) ; // if landscape mode
SetViewportExt(windowX, windowY) ;
```

An HMETAFILE can be obtained through the Windows API call, SetMetaFileBitsEx. Once the HMETAFILE has been obtained and the logical coordinates for the window set, it is a simple process to call PlayMetaFile on the target window.

Raster Formats

Group 3 Fax

As with Windows metafiles, fax bitmaps are stored in an MMS Bitmap Directory. MUSE uses Group 3 Compliant Fax images with 1 dimensional encoding. All images are stored in the file with 1728 dots across the 8 1/2 " page edge and 2168 dots in the other. There are 10 bytes of MMS header information at the beginning of each image.

JPEG Images

JPEG images stored in the MMS Bitmap Directory should be decompressable by standard JPEG techniques. The images are 992(H)x768(V) pixels in landscape mode and 768(H)x992(V) pixels in portrait mode. It should be noted that due to the computationally complex nature of JPEG compression, this output type will take the longest of any output requests to render.

Windows Bitmaps (.BMP)

Windows Bitmap images are stored in the MMS Bitmap Directory. The images are 992(H)x768(V) pixels in landscape mode and 768(H)x992(V) pixels in portrait mode. Output files in this format tend to be very large.

TIFF Version 4

Because TIFF images can accommodate multiple images per file, TIFF files are not stored in the MMS Bitmap Directory. The images are 992(H)x768(V) pixels in landscape mode and 768(H)x992(V) pixels in portrait mode. Currently, only black and white rendering is supported for TIFF.

PCL 5

The MUSE CV-API generates a single output file, suitable for transmission to a HP PCL5 printer. The image uses HPGL encoding whereever possible to make best use of printer resolution; however, on some older printers these images may take longer to render (1/2-1 minute).

Text Formats

ASCII

The forms which are utilized with MUSE CV can be rendered into a plain ASCII file. The file consists of each of the forms specified in the FMTOPTS file appended into the file.

HTML

The forms which are utilized with MUSE CV can be rendered into a HTML file. The file consists of each of the forms specified in the FMTOPTS file appended into the file. The forms are preceded in the file with a set of hypertext links to each form page.

MMS Bitmap Directory Header

The MMS Bitmap Directory Header has the following structure:

```
typedef struct
{
  unsigned long  lOffset; /*  4 offset to page                                  */
  unsigned long  lSize;   /*  4 byte count of page                              */
  MUSE_PID  Pid;          /* 17 patient id                                      */
  BYTE SignalRecType;     /*  1 Signal Rec Type                                 */
  MUSE_DATE AcqDate;      /*  4 Acq Date                                        */
  MUSE_TIME AcqTime;      /*  4 Acq Time                                        */
  WORD nPage;             /*  2 Page n of mPage                                 */
  WORD mPage;             /*  2 Page m in mPage                                 */
  BYTE ImageType;         /*  1 OT_FAX, OT_POSTSCRIPT,....                      */
  BYTE Site;              /*  1 Site Number                                     */
  MUSE_PNAME Patient;     /* 28 Patient Name                                    */
  WORD hPixels;           /*  2 Horizontal Pixels                               */
  WORD vPixels;           /*  2 Vertical Pixels                                 */
#define R4B_ORIENT_LANDSCAPE 0
#define R4B_ORIENT_PORTRAIT  1
  BYTE PageOrientation;   /*  1 Landscape (0) or Portrait (1)                   */
  BYTE WhichOne;          /*  1 Current, 1st Prev, 2nd Prev Oldest              */
  BYTE Retrieval;         /*  1 0=from plot distribution, 1 = manually requested */
  BYTE Pad;               /*  1 Even it out                        */
#define R4B_TITLE_SIZE       32
  BYTE szPageTitle[R4B_TITLE_SIZE] ; /* title for the page */
} IMGPAGEDIR;             /* 108 */

typedef struct
{
#define R4B_IDENTIFIER  "IMGDIR"
#define R4B_IDENTIFIER_SIZE 6
  BYTE        identifier[R4B_IDENTIFIER_SIZE] ;   /* always 'IMGDIR' without null */
  WORD        wPages;                             /* number of pages              */
  FAXPAGEDIR  stimagePage[1];                     /* page info-one for each page  */
} IMGDIR;

#endif
```

GE Medical Systems
Information Technologies

gemedical.com