Manage your IBM i Domino servers using APIs

Bin Yang
Shuang Hong Wang

August 08, 2011

This article describes a set of APIs which enable programmatic access and management of Domino servers on IBM i. The article includes code examples to demonstrate the usage of APIs to perform these tasks.

Introduction

Many customers run their Domino servers on IBM i. Domino on IBM i differentiates itself from on other platform in that you can install multiple versions of Domino on one single IBM i partition and configure multiple Domino partitions(instances) of each version of Domino, while on the other platforms on each server usually you can only install and configure one Domino server instance. This enables IBM i to be the natural platform for the enterprises who need to deploy multiple Domino servers.

Lotus Domino Administrator is the primary tool provided by IBM to manage and operate the Domino servers on different kind of platforms including IBM i. Given the specific characteristic of Domino on IBM i, IBM provides a couple of Domino CL(Control Language) commands for you. By using the Domino CL commands you can view all the Domino servers configured on an IBM i partition, start or stop Domino server(s), monitor Domino server status, view and mange all the jobs of a Domino server, view and edit Domino server Notes.ini file, view Domino server console log, and so on. Besides the manual steps to manage Domino servers using the Domino CL commands, some IT departments have a need to manage Domino servers programmatically. For example, IT may require that all the active Domino servers be shutdown automatically each Saturday at 5PM, so that backups can be performed and all of the NSF databases compacted to release disk storage. Domino on IBM i provides a set of APIs to address these type of requirements. This IBM i API set allows you to list all the Domino servers, retrieve information from a server, get or set Domino server's Notes.ini, and so on. By using the Domino for IBM i APIs together with some of the Domino CL commands, you can easily fulfill the above requirements of managing Domino servers automatically. These Domino for IBM i unique APIs are different from the Lotus C APIs for Notes/Domino, which are used to create or delete NSF databases, read or edit documents in the NSF database, create and use database index/views, and so on.

In this article, we give some brief introduction to the Domino for IBM i unique APIs, some examples on how to use them, and some tips on how to use them. These APIs are specific for IBM i and included in the base option of Lotus Domino on IBM i License Program.

The examples in this article assume a basic knowledge of Lotus Domino and are based on Lotus Domino 8.5.1 running on an IBM i system with Version 5 Release 4 (V5R4).

Table 1 lists the Domino for IBM i unique APIs.

Table 1. API function list

API Function Name	Description
QnninListDominoServers	Get the list of Domino servers on the system.
QnninRtvDominoServerI	Get specific information of Domino server.
QnninRtvDominoServerAttr	Retrieve specific information of Domino server. This API program can retrieve all of the information provided by QnninRtvDominoServerI plus additional information based on the format name of the parameter.
QnninSetDominoEnv	Set the current job's working environment into a state to allow the NotesInitExtended API to be called for a specific Domino server. This removes the burden from the caller from having to know this specific information about each Domino server when initializing the Notes API environment.
QnninGetDominoEnv	Get the information about the current job's Domino server environment.
QnninListDominoRlsI	Get all of the installed Domino release information on the current system.
QnninGetIniValue and QnninGetIniValuez	Get a value of ini variable from the Domino server's notes.ini file.
QnninSetIniValue and QnninSetIniValuez	Set a value of ini variable in the Domino server's notes.ini file.
QnninGetServerDocItem and QnninGetServerDocItemz	Get an item value from the Domino server's server document found in the Domino directory.
QnninSetServerDocItem and QnninSetServerDocItemz	Change or set an item value in the Domino server's server document found in the Domino directory.

How to use Domino for IBM i unique APIs

This section describes some of the Domino for IBM i unique APIs, gives some C program examples on how to use these APIs. These APIs are from service program QNNINLDS.SRVPGM under the QNOTES library. Most of these APIs have no similar commands except QnninSetDominoEnv API which has a similar CL command called SETDOMENV.

To compile the programs in this article and get the PGM objects, you can refer to below commands in listing 1.

Listing 1. General compile commands

```
ADDLIBLE QNOTES
CRTCMOD lib_name/mod_name lib_name/QCSRC output(*PRINT)
CRTPGM  lib_name/mod_name
```

List Domino Servers (QnninListDominoServers)

To manage a Domino server, it is critical to have the name of the Domino server. The QnninListDominoServers API provides an easy way to retrieve all of the Domino server's names configured on a system. With the list of the names, we can manage or access the Domino servers through other APIs.

QnninListDominoServers parameters:

a. Data buffer for Domino servers
   Output;Char(*)This buffer stores the list of Domino servers. And the server names are returned in the CCSID of the current job. The format of the output buffer is defined by the 3rd parameter(Format name).
b. Data buffer length
   Input; Binary(4)The length of the data buffer. If the length is big enough, the list of configured Domino servers will be placed in the data buffer area. Otherwise it will result in errors or incomplete data being returned.
c. Format name
   Input;Char(8)The name of the format used to retrieve all of the configured Domino servers. Currently, the only supported format name is DSRV0100. Please see Table 2.
d. Error code
   I/O;Char(*)Return error information. If this parameter is omitted, diagnostic and escape messages will be issued to the application.

Table 2. “DSRV0100” format

Offset	Type	Field	Field description
0x00	Binary(4)	Bytes returned	The length of data that could be returned.
0x04	Binary(4)	Bytes available	The length of data returned in this structure.
0x08	Binary(4)	Offset to server entry	The offset to the start of the first server entry.
0x0C	Binary(4)	Number of server entries returned	The number of server entries in the returned "Server entries" field.
0x10	Binary(4)	Length of server entry	The length of each server entry returned. It should be 255.
0x14	Char(*)	The start of the server entries returned.	

QnninListDominoData_t is a pre-defined data structure in the header file H/QNNINLDS under the QNOTES library that can be used when calling the QnninListDominoServers API.

Listing 2. Structure QnninListDominoData

```
typedef struct QnninListDominoData {
     int      BytesReturned;         /* Bytes returned             */
     int      BytesAvailable;        /* Bytes available to return  */
     int      OffsetToServers;       /* Offset to server entries   */
     int      NumberOfServers;       /* Number of server entries   */
     int      LengthOfEntry;         /* Length of each server entry*/
}  QnninListDominoData_t;
```

The sample code below gives an example to list all of the Domino server's names on the current system. First, we defined a structure my_sturct_t which contains a QnninListDominoData_t and a string array to hold Domino server names. The variable svr_list is defined of type my_struct_t and it's the first parameter of QnninListDominoServers API. After we called this API, the server names

could be returned into the svr_list. Please note that the length of returned server name is 255 and it's not NULL terminated. If it's less than 255 it will be padded with blanks. The "for" statements in this sample is used to list all of the returned Domino server names without the blanks.

Listing 3. Sample code for QnninListDominoServers

```
#include "QNNINLDS.H"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>

typedef  struct my_struc_t {
  QnninListDominoData_t   hdr;
  char                    server_names[255*100];
}  my_struct_t;

int  main(int argc, char * argv[])
{
  my_struct_t  svr_list;
  Qus_EC_t     error_code;
  int          lcl_NumberOfServers, lcl_LengthOfEntry, lcl_OffsetToServers;
  char         * svr_ptr, * svr_entry;
  int          i,j;

  memset((char *)&error_code, 0, sizeof(error_code));
  error_code.Bytes_Provided  = sizeof(error_code);
  QnninListDominoServers((void *)&svr_list,  sizeof(svr_list), "DSRV0100", &error_code);
  if (error_code.Bytes_Available)  /* Check the error_code. */
  {
     printf("Error retrieving the list of Domino servers.\n");
     return(-1);
  }

  lcl_NumberOfServers = svr_list.hdr.NumberOfServers;
  lcl_LengthOfEntry = svr_list.hdr.LengthOfEntry;
  lcl_OffsetToServers = svr_list.hdr.OffsetToServers;

  for (i=1; i <= lcl_NumberOfServers; i++)  /* List all of the domino servers. */
  {
     svr_ptr = ((char *)&svr_list) + lcl_OffsetToServers + lcl_LengthOfEntry*(i-1) + 254;
     svr_entry = svr_ptr-254;
     j=254;
     while(*svr_ptr == ' '&& j>=0)  /* Remove the blank from the returned server name. */
     {
         *svr_ptr = '\0';
         j--;
         svr_ptr--;
     }
     printf("Server names: %s\n", svr_entry);

  }
  return(0);
}
```

Retrieve Domino Server Attributes (QnninRtvDominoServerAttr)

When we have the Domino server's name, we can use the QnninRtvDominoServerAttr API to retrieve more attributes from this server such as server status, data directory path,executable directory path, Domino version and so on.

QnninRtvDominoServerAttr parameters:

a. Data buffer for Domino server attributes
   Output;Char(*)This buffer stores the Domino server's attributes. The format of the output buffer is defined by the 5th parameter(Format name).
b. Data buffer length
   Input;Binary(4)The length of the data buffer. Please make sure it's big enough.
c. Server name
   Input;Char(255)The name of Domino server which could be returned by QnninListDominoServers API to retrieve information. This field is blank padded and is not null terminated.
d. Server name length
   Input;Binary(4)The length of server name. It should always be 255. if the real server name length is less than 255, it will be padded with blanks.
e. Format name
   Input;Char(8)The name of the format to retrieve different Domino server attributes. DATR0100, DATR0200. See Table 3 for DATR0100 format.
f. Error code
   I/O;Char(*)Return error information for the caller.

Table 3. “DATR0100” Attribute Buffer

Offset	Type	Field
0x00	Binary(4)	Bytes returned
0x04	Binary(4)	Bytes available
0x08	Binary(4)	Primary type of domino server
0x0C	Binary(4)	Secondary type of Domino server
0x10	Binary(4)	Number of active jobs in the subsystem
0x14	Binary(4)	Offset to data directory path
0x18	Binary(4)	Length of data directory path
0x1C	Binary(4)	Offset to executable directory path
0x20	Binary(4)	Length of executable directory path
0x24	Char(20)	Subsystem description
0x38	Char(10)	Library name
0x42	Binary(1)	Server status
0x43	Binary(1)	Auto-Start with TCP/IP Servers
0x44	Binary(4)	Partition Number
0x48	Binary(4)	Offset to generic information
0x4C	Binary(4)	Length of generic information
0x50	Char(16)	The version of Domino Server(VRM)
*	Char(*)	Variable data

For the detail information about DATR0100 and DATR0200 formats, you can refer to the Lotus Domino for i5/OS Application Development Guide in the Resources section.

QnninRtvDominoData_t is a pre-defined data structure in the header file H/QNNINLDS under the QNOTES library that can be used when calling the QnninRtvDominoserverAttr API. QnninRtvDominoData_t contains the data with DATR0100 format.

Listing 4. Structure QnninRtvDominoData

```
/*DATR0100 format*/
typedef struct QnninRtvDominoData {
     int      BytesReturned;         /* Bytes returned             */
     int      BytesAvailable;        /* Bytes available to return  */
     int      PrimaryType;           /* Type of Domino server      */
     int      SecondaryType;         /* Secondary type of server   */
     int      ActiveJobsInSbs;       /* Number of active jobs in   */
                                     /* subsystem.                 */
     int      OffsetToPath;          /* Offset to data dir. path   */
     int      LengthOfPath;          /* Length of data dir. path   */
     int      OffsetToRunPath;       /* Offset to run path         */
     int      LengthOfRunPath;       /* Length of run path         */
     char     SubsystemName[20];     /* Subsystem description name */
     char     RunTimeLibrary[10];    /* Library where run time     */
                                     /* routines of the server     */
                                     /* are found.                 */
     char     Status;                /* Status of server.          */
     char     TCPAutoStart;          /* Auto start server on STRTCP*/
     int      PartitionNumber;       /* Partition number of svr    */
     int      OffsetToInfo;          /* Offset to generic info     */
     int      LengthOfInfo;          /* Length of generic info     */
     char     ServerVRM[16];         /* VRM of Domino Server       */
}  QnninRtvDominoData_t;
```

The sample code below gives an example to retrieve some information of the Domino server. In this sample, we called QnninRtvDominoServerAttr API with "DATR0100" format name and extracted the server's subsystem name, runtime library, server version and server status.

Listing 5. Sample code for QnninRtvDominoServerAttr

```
#include "QNNINLDS.H"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>

int  main(int argc, char * argv[])
{
     QnninRtvDominoData_t   svr_attr;
     Qus_EC_t     error_code;
     char         server[]="D851";
     char *       status;
     char         sbsname[21], runtimelib[11],svrm[17];
     int          i;


     memset((char *)&error_code, 0, sizeof(error_code));
     error_code.Bytes_Provided  = sizeof(error_code);
     QnninRtvDominoServerAttr((void *)&svr_attr, sizeof(svr_attr), server, sizeof(server),
     "DATR0100",&error_code);

     if (error_code.Bytes_Available)  /* Check the error_code. */
     {
        printf("Error retrieving the attributes of Domino server. \n");
        return(-1);
     }
```

```
    sbsname[20]='\0';
    memcpy(sbsname, svr_attr.SubsystemName,sizeof(svr_attr.SubsystemName));
    runtimelib[10]='\0';
    memcpy(runtimelib, svr_attr.RunTimeLibrary,sizeof(svr_attr.RunTimeLibrary));
    svrm[16]='\0';
    memcpy(svrm, svr_attr.ServerVRM,sizeof(svr_attr.ServerVRM));

    switch(svr_attr.Status){
    case 1: { status = "Server ended"; break; }
    case 2: { status = "Server started"; break; }
    case 3: { status = "Server starting"; break; }
    case 4: { status = "Server ending"; break; }
    case 5: { status = "Server in standby mode"; break; }
    case 99:{ status = "Server in unknown status"; break; }

    default:
          printf("Error status\n");
    }
    printf("Server name: %s;\nServer status: %s;\nSubsystemName: %s;\nRunTimeLibrary:
  %s;\nServer version: %s\n", server, status, sbsname, runtimelib, svrm);

    return(0);
}
```

List Domino Release Information (QnninListDominoRlsI)

This API can list all of installed Domino releases on the current system. When you develop an application to support different Domino releases, this API is very helpful and flexible to retrieve Domino release information from the system. This information can be accessed manually from the Work with Licensed Programs menu (Go LICPGM), but this API retrieves just the Domino licensed program information which eliminates the work of searching through the entire list installed licensed programs.

QnninListDominoRlsI parameters:

a. Data buffer for Domino releases
   Output;Char(*)This buffer contains the returned list of installed Domino releases information with the defined format shown in Table 4.
b. Data buffer length
   Input;Binary(4)The length of the data buffer. The length must be at least big enough to hold the Bytes returned field and Bytes available field. Failure to provide enough room for the data will result in errors or incomplete data being returned.
c. Format name
   Input;Char(8)The name of the format for Domino release information. You can use "DRLS0100", Please see Table 5.
d. Error code
   I/O;Char(*)Return error information for the caller.

Table 4. Data buffer's structure

Offset	Type	Field
0x00	Binary(4)	Bytes returned
0x04	Binary(4)	Bytes available

0x08	Binary(4)	Offset to release entries
0x0C	Binary(4)	Number of release entries
0x10	Binary(4)	Length of each release entry

Table 5. “DRLS0100” format

Offset	Type	Field
0x00	Binary(4)	Length of run path
0x04	Char(256)	Run path
0x104	Char(10)	Run time library
0x10E	Char(16)	Release
0x1E	Char(7)	Product ID
0x125	Char(5)	Product Option
0x12A	Char(16)	Reserved

Below are the pre-defined data structures in the header file H/QNNINLDS under the QNOTES library that can be used when calling the QnninListDominoRlsI API. QnninListDominoRls defines the lists of release information and DominoRls100 defines the release information with DRLS0100 format.

Listing 6. Structures used in QnninListDominoRlsI

```
/* DRLS0100 format Rls Info List structure */
typedef struct QnninListDominoRls {
     int      BytesReturned;         /* Bytes returned             */
     int      BytesAvailable;        /* Bytes available to return  */
     int      OffsetToRls;           /* Offset to release entries  */
     int      NumberOfRls;           /* Number of release entries  */
     int      LengthOfEntry;         /* Length of each release entry*/
}  QnninListDominoRls_t;

/* DRLS0100 format Rls Info structure */
typedef struct DominoRls100
{
     int  LengthOfRunPath;       /* Length of run path         */
     char RunPath[256];          /* Run path                   */
     char RunTimeLibrary[10];
     char Release[16];           /* Domino Release (n.n.n format) */
     char ProdID[7];
     char Option[5];
     char Reserved[16];
} DominoRls100_t;
```

The sample code below gives an example to list all of the installed Domino releases on the system. The release information includes runtime library, run path, release, product ID and product option.

Listing 7. Sample code for QnninListDominoRlsI

```
#include "QNNINLDS.H"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <qusec.h>

int  main(int argc, char * argv[])
{
   char buffer[8192];
   QnninListDominoRls_t * hdr;
   DominoRls100_t   *dom_rls;
   Qus_EC_t     error_code;
   int  lcl_NumberOfRls,lcl_OffsetToRls, lcl_LengthOfEntry;
   int  i;
   int  LengthOfRunPath;       /* Length of run path         */
   char RunPath[256];          /* Run path                   */
   char RunTimeLibrary[11];
   char Release[17];         /* Domino Release (n.n.n format) */
   char ProdID[8];
   char Option[6];

   memset((char *)&error_code, 0, sizeof(error_code));
   error_code.Bytes_Provided  = sizeof(error_code);
   QnninListDominoRlsI(buffer,  sizeof(buffer), "DRLS0100", &error_code);

   if (error_code.Bytes_Available)  /* Check the error_code. */
   {
      printf("Error retrieving the release information of Domino server. \n");
      return(-1);
   }

   hdr = (QnninListDominoRls_t *)buffer;
   lcl_NumberOfRls = hdr->NumberOfRls;
   lcl_OffsetToRls = hdr->OffsetToRls;
   lcl_LengthOfEntry = hdr->LengthOfEntry;

   for(i=0; i<lcl_NumberOfRls; i++)
   {

      dom_rls = (DominoRls100_t *)(buffer + lcl_OffsetToRls + (i*lcl_LengthOfEntry));

      LengthOfRunPath = dom_rls->LengthOfRunPath;
      RunPath[LengthOfRunPath] = '\0';
      RunTimeLibrary[10] = '\0';
      Release[16] = '\0';
      ProdID[7] = '\0';
      Option[5] = '\0';

      memcpy(RunPath, dom_rls->RunPath, LengthOfRunPath);
      memcpy(RunTimeLibrary, dom_rls->RunTimeLibrary, sizeof(dom_rls->RunTimeLibrary));
      memcpy(Release, dom_rls->Release, sizeof(dom_rls->Release));
      memcpy(ProdID, dom_rls->ProdID, sizeof(dom_rls->ProdID));
      memcpy(Option, dom_rls->Option, sizeof(dom_rls->Option));

      printf("########################################\n");
      printf("RunPath %s\n", RunPath);
      printf("RunTimeLibrary %s\n", RunTimeLibrary);
      printf("Release %s, ProdID %s, Option %s\n", Release, ProdID, Option);

   }

   return(0);
}
```

Get and Set Notes.ini Value (QnninGetIniValue and QnninSetIniValue)

These two APIs are used to get or set a value in Domino server's notes.ini file. It provides an easy way to programatically change settings in notes.ini file. For example, if a customer wants to enable semaphore debug on all of their Domino servers; they can use this API to check each server to

see if the setting is already turned on. If not, they can programatically enable semaphore debug. By doing this, the customer avoids having to manually check or set it on each server using CL commands.

QnninSetIniValue parameters:

a. Server name
   Input;Char(255)The name of the Domino server which you want to set the notes.ini value.
b. Server name length
   Input:Binary(4)The length of the server name.
c. Notes.ini variable name
   Input;Char(*)The name of the variable which you want to set a value.
d. Notes.ini variable name length
   Input:Binary(4)The length of the Notes.ini variable name.
e. The value for notes.ini variable
   Input:Char(*)The new value for this variable.
f. Flag
   Input;Binary(4);REPLACE_LIST, APPEND_TO_LIST and REMOVE_FROM_LIST.
g. Error code
   I/O; Char(*)Returned error information.

This sample code below gives an example to get the value and then set another value to the notes.ini variable. In this sample, we checked the Domino server's notes.ini setting "CONSOLE_LOG_ENABLED" and set it to "1". By using these two APIs, you can operate with any other notes.ini settings.

Listing 8. Sample code for QnninGetIniValue and QnninSetIniValue

```
#include "QNNINLDS.H"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>

int  main(int argc, char * argv[])
{
   char server[]="D851";
   char ini_name[] = "CONSOLE_LOG_ENABLED";
   char ini_value[] = "1";
   char buffer[8192];
   int  buffer_length, buffer_return;
   Qus_EC_t     error_code;

   memset((char *)&error_code, 0, sizeof(error_code));
   error_code.Bytes_Provided  = sizeof(error_code);

   buffer_length = sizeof(buffer);
   QnninGetIniValue(server, sizeof(server), ini_name,  sizeof(ini_name)
           , buffer, buffer_length, &buffer_return, &error_code);

   if (error_code.Bytes_Available)  /* Check the error_code. */
   {
      printf("Error retrieving the ini value of Domino server. \n");
      return(-1);
   }
```

```
   if(8192>buffer_return)
   {
      buffer[buffer_return] = '\0';
      printf("CONSOLE_LOG_ENABLED value %s\n",buffer);

   }

   memset((char *)&error_code, 0, sizeof(error_code));
   error_code.Bytes_Provided  = sizeof(error_code);

   QnninSetIniValue(server,sizeof(server),ini_name, sizeof(ini_name),
   ini_value, sizeof(ini_value), REPLACE_LIST ,&error_code);

   if (error_code.Bytes_Available)  /* Check the error_code. */
   {
     printf("Error setting the ini value of Domino server. \n");
     return(-1);
   }

   return(0);
}
```

QnninGetIniValuez is the same as QnninGetIniValue API except that the input string parameters are entered as null terminated strings. The same as QnninSetIniValuez and QnninSetIniValue.

Get and set Server Document Item

We have 4 related APIs to get or set a value of server document item.

- QnninGetServerDocItem
- QnninGetServerDocItemz
- QnninSetServerDocItem
- QnninSetServerDocItemz

The server document item means an item from a Domino server's server document in the Domino directory database (names.nsf). QnninGetServerDocItemz API is identical to the QnninGetServerDocItem API except that the input string parameters are entered as null terminated strings. It's the same as QnninSetServerDocItemz and QnninSetServerDocItem.

Here we give an example on how to use QnninGetServerDocItemz.

QnninGetServerDocItemz parameters:

a. Server name
   Input:Char(255)The name of the Domino server that you want to get information from.
b. Server document item name
   Input:Char(*)The item name for the entry in the server document.
c. Return buffer
   Output:Char(*)The buffer used to store item's value.
d. Return buffer length
   Input:Binary(4)The length of return buffer.
e. Return bytes available
   Output:Binary(*)The number of bytes that returned in the return buffer.

f. Return buffer data type
   Input:Binary(4)There are three different data types for the item value. TEXT_DATA_TYPE is used for the text data. FLOAT_DATA_TYPE is used for the float data and INTEGER_DATA_TYPE is for the integer data.
g. Error code
   I/O:Char(*)Returned error information.

In this example, the program retrieves the server title from the server document in names.nsf and sets the title to "Domino server for API test".

Listing 9. Sample code for QnninGetServerDocItem and QnninSetServerDocItem

```
#include "QNNINLDS.H"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>
#include <qusrjobi.h>

int  main(int argc, char * argv[])
{
     char server[]= "D851";
     char item_name[] = "ServerTitle";
     char item_new_value[] = "Domino server for API test";
     char buffer[8192];
     int  buffer_length, buffer_return;
     Qus_EC_t     error_code;

     memset((char *)&error_code, 0, sizeof(error_code));
     error_code.Bytes_Provided  = sizeof(error_code);

     buffer_length = sizeof(buffer);
     QnninGetServerDocItem(server, sizeof(server), item_name,  sizeof(item_name)
               , buffer, buffer_length, &buffer_return, TEXT_DATA_TYPE ,&error_code);

     if (error_code.Bytes_Available)  /* Check the error_code. */
     {
        printf("Error retrieving the item's value of Domino server. \n");
        return(-1);
     }

     if(8192>buffer_return)
     {
        buffer[buffer_return] = '\0';
        printf("ServerTitle %s\n",buffer);
     }

     memset((char *)&error_code, 0, sizeof(error_code));
     error_code.Bytes_Provided  = sizeof(error_code);

     QnninSetServerDocItem(server,sizeof(server),item_name,sizeof(item_name),
     item_new_value, sizeof(item_new_value), REPLACE_LIST, TEXT_DATA_TYPE ,&error_code);

     if (error_code.Bytes_Available)  /* Check the error_code. */
     {
        printf("Error setting the item's value of Domino server. \n");
        return(-1);
     }

     return(0);
 }
```

Tips when using these APIs

1. When compiling your C program, you may meet below error messages:
   #include file "QNNINLDS.H" not found. You should make sure that QNOTES is in your library list. And then try it again.
2. A header file H/QNNINLDS under the QNOTES library contains the Macros list, Structures list and Function Prototype list for the APIs.
3. When you change notes.ini settings or server document settings, some of them may not take effective before Domino server restarts.
4. When you deal with server document item, you should use the correct item name.

Conclusion

This article introduces some basic knowledge of the Domino for IBM i unique APIs together with some sample code. These APIs provide an easy way to develop programs to manage and operate Domino servers on IBM i, especially for customers who have many Domino servers. We hope that this article serves as a first step for Domino on IBM i developers to explore and exploit these APIs in their applications.

Resources

- Domino for i5/OS server APIs : Read this main web page for these APIs.
- Chapter 4, Domino for i5/OS APIs : Read this chapter of Application Development Guide.
- IBM Lotus Domino and Notes Information Center: More information for Lotus Domino
- IBM i Information Center: More information for IBM i.