Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

First Edition (November 1996)

This edition applies to Version 3, Release 7, Modification Level 0, of IBM Application System/400 ILE C/400 (Program 5716-CX2) and Visual Age for C++ for AS/400 (Program 5716-CX5) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory,
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada. M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See “Communicating Your Comments to IBM” for a description of the methods. This page immediately precedes the Readers’ Comment Form at the back of this publication.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Notices ........................................... vii
Programming Interface Information ...................... vii
Trademarks and Service Marks ................................ vii

About This Book ................................... ix
Who Should Use This Book ............................ ix
A Note About Examples ............................... ix

Machine Interface Library Functions ..................... 1
Access to the Machine Interface ......................... 2
Omitted Operands .................................... 2
Exception Handling .................................... 3
Using the Templates declared in the MI Header Files .......... 4
Header Files ........................................... 6
  Computation and Branching Instructions ................. 6
  Date/Time/Timestamp Instructions .................... 7
  Pointer/Name Resolution Addressing Instructions .......... 7
  Space Object Addressing Instructions ................ 7
  Space Management Instructions ....................... 7
  Program Management Instructions .................... 8
  Program Execution Instructions ...................... 8
  Independent Index Instructions ..................... 8
  Queue Management Instructions .................... 8
  Object Lock Management Instructions ................. 8
  Authorization Management Instructions ............... 9
  Process Management Instructions .................... 9
  Resource Management Instructions .................. 9
  Machine Observation Instructions ................... 9
  Machine Interface Support Instructions ............... 9
  Mutex Instructions ................................ 10
  Job Information Instructions ....................... 10
Original Program Model Description ..................... 10
Independent Index Instructions ........................ 11
Queue Management Instructions ........................ 13
Establishing an Invocation Exit Program (ATIEXIT) ........... 15
Compute Date Duration (CDD) .......................... 17
Clear Bit in String (CLRBTS) .......................... 21
Compare Pointer for Space Addressability (CMPPSPAD) .......... 23
Compare Pointer for Object Addressability (CMPPTRA) .............. 25
Compare Pointer Type (CMPPTRT) ...................... 28
Compress Data (CPRDATA) ............................ 31
Compute Time Duration (CTD) .......................... 33
Compute Timestamp Duration (CTSD) .................... 36
Convert Binary Synchronous Communications to Character (CVTBC) .... 40
Convert Character to Binary Synchronous Communications (CVTCB) .... 42
Convert Eight Bit Character to Hex Nibbles (CVTCH) ............. 44
Convert a Character String to Multi-Leaving Remote Job Entry (CVTCM) ...... 46
Convert a Character String to System Network Architecture (CVTCS) ....... 48
Convert Date (CVTD) .................................. 50
Convert External Form to Numeric Value (CVTEFN) ............. 54

© Copyright IBM Corp. 1996
Convert Hex to Character (CVTHC) ........................................ 56
Convert Multi-Leaving Remote Job Entry to Character (CVTMC) .... 58
Convert System Network Architecture to Character (CVTSC) .......... 60
Convert Time (CVTT) ................................................ 62
Convert Timestamp (CVTTS) ........................................... 65
Copy Bytes Left-Adjusted (CPYBLA) .................................... 69
Copy Bytes Left-Adjusted with Pad (CPYBLAP) ....................... 71
Copy Hex Digit Numeric to Numeric (CPYHEXNN) .................... 73
Copy Hex Digit Numeric to Zone (CPYHEXNZ) ....................... 75
Copy Hex Digit Zone to Numeric (CPYHEXZN) ....................... 77
Copy Hex Digit Zone to Zone (CPYHEXZZ) ........................... 79
Copy Numeric Value (CPYNV) ......................................... 81
Decompress Data (DCPDATA) ......................................... 83
Decrement Date (DECD) ............................................... 85
Decrement Time (DECT) ............................................... 89
Decrement Timestamp (DECTS) ....................................... 92
Dequeue (DEQ) ................................................................ 96
Dequeue Message with Indicator (DEQI) ............................... 98
Edit (EDIT) .................................................................... 100
Edit Packed Decimal (EDIT_PACKED) ................................ 102
Enqueue (ENQ) ................................................................ 104
Ensure Object (ENSOBJ) ............................................... 106
Extract Exponent (EXTREXP) .......................................... 108
Find Independent Index Entry (FNDINXEN) ......................... 110
Find Relative Invocation Number (FNDRINVN) ..................... 113
Increment Date (INCD) ................................................ 116
Increment Time (INCT) ................................................ 120
Increment Timestamp (INCTS) ......................................... 123
Insert Independent Index Entry (INSINXEN) ....................... 127
Lock Object (LOCK) ...................................................... 130
Lock Space Location (LOCKSL) ...................................... 133
Lock Space Location with Time-Out (LOCKSL2) ................... 136
Materialize Activation Attribute (MATACTAT) .................... 139
Materialize Access Group Attributes (MATAGAT) ................ 141
Materialize Activation Group Attributes (MATAGPAT) .......... 145
Materialize Allocated Object Locks (MATAOL) .................... 148
Materialize Invocation Attributes (MATINVAT) ................... 150
Materialize Independent Index Entries (MATINXAT) ............. 153
Materialize Machine Attributes (MATMATTR) .................... 155
Materialize Machine Data (MATMDATA) ............................ 157
Materialize Object Locks (MATOBJLK) .............................. 159
Materialize Program (MATPG) ....................................... 161
Materialize Pointer (MATPCTR) ....................................... 163
Materialize Pointer Locations (MATPTRL) .......................... 165
Materialize Process Activation Groups (MATPRAGP) ............ 168
Materialize Process Attributes (MATPRATR) ..................... 171
Materialize Process Locks (MATPRLOCK) .......................... 173
Materialize Queue Attributes (MATQAT) ............................ 176
Materialize Queue Messages (MATQMSG) .......................... 178
Materialize Resource Management Data (MATRMD) .............. 182
Materialize Space Attributes (MATS) .................................. 184
Materialize Selected Locks (MATESELLK) ......................... 186
Materialize System Object (MATSOBJ) .............................. 188
Materialize Time-of-Day Clock (MATTOD) .......................... 190
Machine Interface Time (MITIME) .......................... 192
Modify Automatic Storage Allocation (MODASA) ................. 195
Modify Independent Index (MODINX) .......................... 197
Modify Space Attributes (MODS) ................................ 200
Modify Space Attributes - Long Form with Template (MODS2) ...... 203
Resolve System Pointer (RSLVSP) ............................ 206
Retrieve Computational Attributes (RETCA) ..................... 208
Remove Independent Index Entry (RMVINXEN) ..................... 211
Scan with Control (SCANWC) ..................................... 214
Set Access State (SETACST) ...................................... 216
Set Bit in String (SETBTS) ....................................... 219
Set Computational Attributes (SETCA) .......................... 221
Set System Pointer from Pointer (SETSPFP) ..................... 225
Set Space Pointer from Pointer (SETSPFP) ..................... 228
Set Space Pointer Offset (SETSPPO) ............................. 231
Store Space Pointer Offset (STSPPO) ............................ 233
Test Authority (TESTAU) .......................................... 234
Trim Length (TRIML) .............................................. 236
Test Bit in String (TSTBTS) ...................................... 238
Unlock Object (UNLOCK) ......................................... 240
Unlock Space Location (UNLOCKSL) ............................ 243
Wait on Time (WAITTIME) ........................................ 246
Transfer Object Lock (XFRLOCK) ................................ 248
Translate with Table (XLATEWT) .................................. 252

Appendix: Reference Summary .................................. 255

Bibliography ......................................................... 273

Index ................................................................. 275
Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user’s responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This book is intended to help you write Integrated Language Environment (ILE) C/400 programs using the ILE C/400 compiler or ILE C++ programs using the Visual Age for C++ for AS/400 compiler. It primarily documents general-use programming interfaces and associated guidance information provided by the compilers.

Trademarks and Service Marks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

- AS/400
- OS/400
- 400
- Integrated Language Environment
- IBMLink
- Application System/400
- C/400
- PROFS
- IBM

Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.
About This Book

This book contains reference information on Machine Interface (MI) instructions.

Use this book as a reference when you write Integrated Language Environment (ILE) C and C++ applications. This book is intended to be used in conjunction with the Machine Interface Functional Reference, SC41-4810.

This book does not describe how to program in the C programming language, nor does it explain the concepts of ILE.

For information about other AS/400 publications, see either of the following:

- The AS/400 Information Directory, a unique, multimedia interface to a searchable database containing descriptions of titles available from IBM or from selected other publishers. The AS/400 Information Directory is shipped with your system at no charge.

For a list of related publications, see “Bibliography” on page 273.

Who Should Use This Book

This book is intended for programmers who are familiar with the ILE C/400 programming language and who plan to use the MI function interface in their ILE C or C++ applications. You also need knowledge of ILE as explained in ILE Concepts, SC41-4606

A Note About Examples

The examples in this book are written in a simple style. These examples do not demonstrate all of the possible uses of C language constructs. Some examples are only code fragments and do not compile without additional code.

For ILE C users, all complete runnable examples for the machine interface instructions can be found in library QCLE, in source file QACSRC. The example names are the same as the function name or instruction name. For example, the source code for the example illustrating the use of the cpyla function in this book, can be found in library QCLE, file QACSRC, member CPYBLA.

The QSYSINC library must be installed.
Machine Interface Library Functions

This book describes a set of Machine Interface (MI) library functions that provide system-level programming capabilities. The syntax, parameter descriptions, notes on usage, return values and exceptions are presented where applicable for each function. The information presented here is intended to provide you with an indication of what each function can do and highlight any differences in behavior between the function and the MI instruction. You will need to refer to the Machine Interface Functional Reference, SC41-4810 for a complete description of each MI instruction.

The Machine Interface (MI) is the machine instruction set that allows you to access low level machine procedures.

Most of the MI instructions can be accessed through two interfaces: the builtin interface and the function interface. Some of the functions in the MI library do not have a builtin interface, and some of the builtins do not have a function interface.

The builtin interface is a builtin routine that directly accesses the low level machine procedure. You cannot take the address of a routine through its builtin interface, and there is no stack frame associated with a call to a routine through its builtin interface. Performance may be improved if you use the builtin interface but the machine procedures do not use a consistent parameter passing mechanism nor do they make use of return values and null terminated strings like C library functions do.

Although builtin versions are mentioned in the discussion of the function, the syntax is not provided. You will need to refer to the Machine Interface Functional Reference where the syntax of all the MI builtins are provided.

The function interface provides an easier, more consistent way for passing parameters and using the function calling conventions. If the parameter lists of the function interface and the builtin interface to a machine procedure are identical, the function interface is available as a macro that maps directly to the builtin. If a parameter for an MI function is a string which specifies the name of an AS/400 object, the string must not have any preceding blanks.

Although builtin versions are mentioned in the discussion of the function, the syntax is not provided. You will need to refer to the Machine Interface Functional Reference where the syntax of all the MI builtins are provided.

See Table 1 on page 255 for a summary of all the ILE C/C++ MI function prototypes. For each function, the prototype for the associated builtin is also provided if it is identical to that of the function. The table also contains the prototypes for builtins which do not have a function interface.

The MI header files allow you to access declarations for both the builtin and function interface to an MI instruction.

Not all the MI instructions described in the Machine Interface Functional Reference can be accessed through the functions declared in the MI header files. Many MI instructions are simply not accessible in ILE since the operation they perform can be done just as easily using standard C language constructs. For some of these instructions an equivalent C function is supplied to perform the same operation as the MI instruction. These cases are noted in the function description.
Access to the Machine Interface

You use the AS/400 pointer types with the type definitions and function declarations that make up the MI library to access the MI. The header file <pointer.h> contains the type definitions (typedefs) of the AS/400 pointer types.

Examples

This example shows you how to use the MI builtin _MATS to materialize the attributes of a user space object.

```c
#include <pointer.h>
#include <QSYSINC/MIH/RLVSP>
#include <QSYSINC/MIH/MATS>
#include <QSYSINC/H/QUSCRUTUS>
#define CREATION_SIZE 65536

int main(void)
{
    _SPC_Template_T space_t;
    _SYSPTR ptr_to_space;
    int error_code = 0;
    QUSCRUTUS("MYSPACE  QTEMP  ",
           "MYSPACE  ",
           CREATION_SIZE,
           ",  
           "*ALL  ",
           "MYSPACE example for Programmer's Reference  ",
           "*YES  ",
           &error_code);

    ptr_to_space = rlsvsp(_Usrspc, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);

    space_t.TmpSize = sizeof(_SPC_Template_T);
    _MATS(&space_t, &ptr_to_space);
}
```

In this example, the space is created using the Create User Space (QUSCRUTUS) API. On the call to _MATS, the argument space_t contains the address of the structure into which the space attributes are materialized. This structure is defined in the <QSYSINC/MIH/MATS> header file. The argument ptr_to_space contains the address of the system pointer to the space object whose attributes are materialized. This pointer was obtained by using the rlsvsp function to resolve to the space object. The type definition for _SYSPTR is included in the <pointer.h> header file.

Omitted Operands

If you want to omit an operand (referred to as a "null operand" in the Machine Interface Functional Reference) when you use the function interface to access the MI, you must supply a NULL pointer. This only applies to omitted pointer operands. This allows you to use a single interface to access all variations of an MI instruction.
Examples
This example shows how the process control space pointer (the second parameter) may be omitted from the matpratr (Materialize Process Attributes) function interface by passing a NULL pointer on the call to the function.

```c
void matpratr (_MPRT_Template_T *mprt_t, _SYSPTR ctrl_space, char options);
matpratr (&mptr_t, NULL, options);
```

This example shows you the builtin interface. There are 2 builtins that provide access to the MATPRATR machine instruction; one in which the process control space pointer is omitted and the other in which it is not. If a NULL is passed for the second argument on the call to matpratr, the semantics are the same as the _MATPRATR1 builtin.

```c
void _MATPRATR1 (_MPRT_Template_T *mprt_t, char *options);
void _MATPRATR2 (_MPRT_Template_T *mprt_t, _SYSPTR *ctrl_space, char *options);
```

---

Exception Handling

Exceptions in the Machine Interface Functional Reference are listed as 2-byte hexadecimal numbers as follows:

```
22 Object Access     <---- Group
  01 Object not found <---- Subtype
  02 Object destroyed
  03 Object suspended
```

To monitor for exceptions around calls to MI library functions and builtins you can follow steps 1 and 2 or steps 1, 3 and 4:

1. Convert hex to decimal to determine the correct MCH message. For example, hexadecimal 2201 becomes MCH3401.
2. Use the #pragma exception_handler directive
3. Find out what C signal the exception maps to. It could be one of SIGPFE, SIGILL or SIGSEGV.
4. Choose to:
   - Ignore the signal. The exception is handled and placed in the joblog.
   - Accept the default actions. Using the initial state (SIG_DFL), the exception is not handled and percolates.
   - Call a signal handler. The exception is handled and is placed in the joblog.

Example
This example shows you how exceptions from MI library functions can be monitored and handled using a signal handling function. The signal handler my_signal_handler is registered before the rslvsp function signals a 0x2201 exception. When a SIGSEGV signal is raised, the signal handler is called. If an 0x2201 exception occurred, the signal handler calls the QUSRCRTS API to create a space.
Using the Templates declared in the MI Header Files

Some MI instructions require templates that are 16-byte aligned in memory. If a template contains a pointer (of any type), a structure of this type will always be 16-byte aligned. To ensure 16-byte alignment of a template that does not contain a pointer, you can do one of two things:

1. Use malloc to allocate the storage for the template. Storage allocated by malloc is always 16-byte aligned.

2. Declare the template as part of a union whose other member is a pointer. This will force 16-byte alignment since the alignment of a union is always that of its strictest member.

Many templates contain variant portions that require the template user to allocate the correct amount of storage based on information that is usually not known until run time. Consider, for example, the template used for the matobjlk (Materialize Object Locks) function. The typedef for _MOBJL_Template_T is:
The number of lock state descriptors is variable depending upon the number of locks currently allocated (or waited for) on the object specified. One way to handle the variant portion is to first materialize the information to obtain the number of lock state descriptors. This is given by the Num_Descripts field. It is then known exactly how much storage you need to allocate in order to materialize all the lock state descriptors. If the number of lock state descriptors happens to be three, then the following section of code would allocate the storage needed to materialize all lock state descriptors.

```c
typedef _Packed struct _MOBJL_Template_T {
    int Template_Size;  /* size for materialization*/
    int Bytes_USED;     /* size of data available */
    char Lock_Alloc;    /* Lock states allocated */
    char Lock_Synch;    /* Lock states Synch Wait */
    char Lock_Asynch;   /* Lock states Asynch Wait */
    char reserved1;
    short Num_Descripts; /* # of Lock Descriptors */
    char reserved2[2];
    _LOCK_Descript_T Locks[1];  /* Lock state descriptor(s)*/
    /* Lock state descriptor is repeated for each lock currently */
    /* allocated or waited for. This number is given in the */
    /* Num_Descripts field. */
} _MOBJL_Template_T;

/* Allocate storage for template - disregard variant portion */
MOBJL = (_MOBJL_Template_T *)malloc(sizeof(_MOBJL_Template_T));

/* Set all fields in the template to zero. */
memset(MOBJL, '\0', sizeof(_MOBJL_Template_T));

MOBJL->Template_Size = sizeof(_MOBJL_Template_T);

/* Materialize to obtain number of lock state descriptors */
matobjlk(MOBJL, sys[0]);

/* Calculate size required to hold all lock state descriptors */
    size = sizeof(_MOBJL_Template_T) +
          (MOBJL->Num_Descripts - 1)*sizeof(_LOCK_Descript_T);

/* Allocate the storage through realloc() */
MOBJL = (_MOBJL_Template_T *)realloc(MOBJL, size);

/* Set all fields in the template to zero. */
memset(MOBJL, '\0', sizeof(_MOBJL_Template_T));

MOBJL->Template_Size = size;

/* Materialize again with storage allocated for all descriptors */
matobjlk(MOBJL, sys[0]);
```
Header Files

Each MI function and builtin is prototyped in a header file provided in library QSYSINC, file MIH. If you wish to use the MI instructions and header files, you must have the library QSYSINC installed on your system.

Most of the MI functions and builtins are prototyped in a header file that corresponds to the MI instruction they reference. For example, the clrblts function is declared in library QSYSINC, file MIH, and member CLRBLTS. Following is an illustration of how to include the MI header file that prototypes the clrblts function, from the QSYSINC library, in your program:

```c
#include <QSYSINC/MIH/CLRBLTS>
```

For ILE C/400, in releases prior to V3R6, the MI functions and builtins were declared in header files according to their use (MI group header files). For example, all computation functions were in the <micomput.h> header file in library QCLE. In V3R6 and following releases, the MI header files reside in the library QSYSINC. The H file in QSYSINC contain the MI header files which were previously in QCLE (the MI group header files), and these files, in turn, include the individual MI header files. If you have hard coded QCLE/H when including group MI header files it is necessary for you to duplicate the MI header files from QSYSINC/H to QCLE/H or change your source to remove the QCLE qualification. You may be able to reduce compile time by including the header files for individual MI instructions rather than the group MI header files.

Computation and Branching Instructions

The following lists the MI header files for computation and branching instructions:

- QSYSINC/MIH/CLRBLTS
- QSYSINC/MIH/CPYBYTES
- QSYSINC/MIH/CPRDATA
- QSYSINC/MIH/CPYNV
- QSYSINC/MIH/CVTCB
- QSYSINC/MIH/CVTBC
- QSYSINC/MIH/CVTCM
- QSYSINC/MIH/CVTCS
- QSYSINC/MIH/CVTEFN
- QSYSINC/MIH/CVTMC
- QSYSINC/MIH/RETCA
- QSYSINC/MIH/CVTSC
- QSYSINC/MIH/DCPDATA
- QSYSINC/MIH/EDIT
- QSYSINC/MIH/EDIT_PACKED
- QSYSINC/MIH/LBCPYNV
- QSYSINC/MIH/SCANX
- QSYSINC/MIH/SETBTS
- QSYSINC/MIH/SETCA
- QSYSINC/MIH/TSTBTS
- QSYSINC/MIH/XLATEWT
- QSYSINC/MIH/CVTCH
- QSYSINC/MIH/CVTHC
- QSYSINC/MIH/CPYBLA
- QSYSINC/MIH/CPYBLAP
QSYSINC/MIH/CPYHEXNN
QSYSINC/MIH/CPYHEXNZ
QSYSINC/MIH/CPYHEXZZ
QSYSINC/MIH/CPYHEXZN
QSYSINC/MIH/EXTREXP
QSYSINC/MIH/SCANWC
QSYSINC/MIH/TRIML
QSYSINC/MIH/EXTREXP
QSYSINC/MIH/TRIML
QSYSINC/MIH/XLATEB

*Note:* The above header files were previously combined in the header file <micomput.h>.

**Date/Time/Timestamp Instructions**

The MI header file for date, time and timestamp instructions is:

QSYSINC/MIH/MIDTTM

This header file prototypes all the MI functions and builtins that provide access to the Date/Time/Timestamp instructions.

*Note:* The declarations in <QSYSINC/MIH/MIDTTM> were previously found in <mitime.h>.

**Pointer/Name Resolution Addressing Instructions**

The MI header files for pointer/name resolution addressing instructions are:

QSYSINC/MIH/CMPPTRA
QSYSINC/MIH/CMPPTRT
QSYSINC/MIH/RSLVSP

*Note:* The above header files were previously combined in the header file <miptrnam.h>.

**Space Object Addressing Instructions**

The MI header files for space object addressing instructions are:

QSYSINC/MIH/CMPPSPAD
QSYSINC/MIH/LSPCO
QSYSINC/MIH/SETSPFP
QSYSINC/MIH/SETSPPPFP
QSYSINC/MIH/SETSPPO
QSYSINC/MIH/STSPPO

*Note:* The above header files were previously combined in the header file <mispcobj.h>.

**Space Management Instructions**

The MI header files for space management instructions are:

QSYSINC/MIH/MATS
QSYSINC/MIH/MODS

*Note:* The above header files were previously combined in the header file <mispace.h>.
Program Management Instructions
The MI header file for project management instructions is:

QSYSINC/MIH/MATPG

Note: The declarations is <QSYSINC/MIH/MATPG> were previously found in <mipgmgmt.h>.

Program Execution Instructions
The MI header files for program execution instructions are:

QSYSINC/MIH/MATACTAT
QSYSINC/MIH/MATAGPAT
QSYSINC/MIH/MODASA

Note: The above header files were previously combined in the header file <mipgexec.h>.

Independent Index Instructions
The MI header files for independent index instructions are:

QSYSINC/MIH/FNDINXEN
QSYSINC/MIH/INSINXEN
QSYSINC/MIH/MATINXAT
QSYSINC/MIH/RMVINXEN
QSYSINC/MIH/MODINX

Note: The above header files were previously combined in the header file <miindex.h>.

Queue Management Instructions
The MI header files for queue management instructions are:

QSYSINC/MIH/ENQ
QSYSINC/MIH/DEQ
QSYSINC/MIH/DEQWAIT
QSYSINC/MATQMSG
QSYSINC/MIH/MATQAT

Note: The above header files were previously combined in the header file <miqueue.h>.

Object Lock Management Instructions
The MI header files for object lock management instructions are:

QSYSINC/MIH/LOCK
QSYSINC/MIH/LOCKSL
QSYSINC/MIH/MATAOL
QSYSINC/MIH/MATOBJLK
QSYSINC/MIH/MATPRLK
QSYSINC/MIH/MATSELLK
QSYSINC/MIH/UNLOCK
QSYSINC/MIH/UNLOCKSL
QSYSINC/MIH/XFRLOCK

Note: The above header files were previously combined in the header file <milock.h>.
Authorization Management Instructions
The MI header file for authorization management instructions are:

QSYSINC/MIH/TESTAU

Note: The declarations in <QSYSINC/MIH/TESTAU> were previously found in <miauth.h>.

Process Management Instructions
The MI header files for process management instructions are:

QSYSINC/MIH/MATPRAGP
QSYSINC/MIH/MATPRATR
QSYSINC/MIH/WAITTIME

Note: The above header files were previously combined in the header file <miproces.h>.

Resource Management Instructions
The MI header files for resource management instructions are:

QSYSINC/MIH/ENSOBJ
QSYSINC/MIH/MATAGAT
QSYSINC/MIH/MATRMD
QSYSINC/MIH/SETACST

Note: The above header files were previously combined in the header file <mirsc.h>.

Machine Observation Instructions
The MI header files for machine observation instructions are:

QSYSINC/MIH/FNDRINVN
QSYSINC/MIH/MATINVAT
QSYSINC/MIH/MATPTR
QSYSINC/MIH/MATPTRL
QSYSINC/MIH/MATSOBJ

Note: The above header files were previously combined in the header file <mimchobs.h>.

Machine Interface Support Instructions
The MI header files for machine interface support instructions are:

QSYSINC/MIH/MATMATR
QSYSINC/MIH/MATMDATA
QSYSINC/MIH/MATTOD

Note: The above header files were previously combined in the header file <mimchint.h>.
Mutex Instructions

The MI header files for mutex instructions are:

- QSYSINC/MIH/CRTMTX
- QSYSINC/MIH/DESMTX
- QSYSINC/MIH/LOCKMTX
- QSYSINC/MIH/MATMTX
- QSYSINC/MIH/MATPRMTX
- QSYSINC/MIH/UNLKMTX

Note: The above header files were previously combined in the header file <mimtx.h>.

Job Information Instructions

The header file contains the declarations for the job information instructions that were previously found in the header file <milib.h>:

- QSYSINC/MIH/MICOMMON

Original Program Model Description

The programming environment provided when the AS/400 system was first introduced is called the original program model (OPM). Application developers on the AS/400 enter source code into a source file and compile that source. If the compilation is a success, a program object is created. The set of functions, processes, and rules provided by the OS/400 to create and run a program is known as the OPM.

As an OPM compiler generates the program object, it generates additional code. The additional code initializes program variables and provides any necessary code for special processing that is needed by the particular language. The special processing could include processing any input parameters expected by this program. When a program is to start running, the additional compiler-generated code becomes the starting point (entry point) for the program.

Please refer to the ILE Concepts, SC41-4606 for more information on OPM.
Independent Index Instructions

This program illustrates how to call the system API, QUSCRTUI, to create a user index. This program must be run before any of the user index examples (insinxen, fninxen, matinxat, modinx and rmvinxen).

Example

```c
/*--------------------------------------------------------------------------------*/
/* Example Group: User Indexes <miindex.h> */
/**/
/* Function: none (just an example that can be used with */
/* the other User Index examples) */
/**/
/* Description: This program illustrates how to call the system */
/* API, QUSCRTUI, to create a user index. This */
/* program should be run before any of the user */
/* index examples: insinxen, fninxen, matinxat, */
/* modinx, rmvinxen) */
/* */
/*--------------------------------------------------------------------------------*/

#include <QSYSINC/H/QUSCRTUI>

/* Declare and initialize variables for the call to the QUSCRTUI API. */

/* Index name and library. */
char name_lib[20] = "MYUSRIDX MYLIB ";

char type[] = "F"; /* Fixed Length Entries */
char key_type[] = "I"; /* Insert using "key" */
char immediate_update[] = "0"; /* No immediate update of index */
char optimize[] = "0"; /* Optimize for random reference */
char attribute[10] = "EXAMPLE "; /* Arbitrary attribute name. */
char authority[10] = "*ALL "; /* Authority for the Index. */
char replace[10] = "*NO "; /* Do not replace the index */
/* if it already exists. */
int entry_length = 110, /* Length of each entry. */
key_length = 10; /* Length of the key. */
char description[50] = "User Index being used for keeping a sorted list";
```

/* Define the "error code parameter" structure as defined in the */
/* System Programmer's Interface Reference manual. A typedef of a */
/* structure in this format is provided in the <QSYSINC/H/QUSEC> */
/* header file. The structure typedef is called Qus_EC_t and has */
/* different member names than the structure used in this example, */
/* although the example could be changed to use that typedef. */
/*--------------------------------------------------------------------------------*/
struct {
    int bytes_available;
    int bytes_used;
    char exception_id[7];
    char reserved;
    char exception_data[1];
} error_code;

int main(void) {

    /*---------------------------------------------------------------*/
    /* Call the 'QUSCRTUI' System API to create the user queue with    */
    /* the specified name and library and the above characteristics. */
    /*---------------------------------------------------------------*/

    error_code.bytes_available = 0;  /* Let any exception generated */
    error_code.bytes_used = 0;       /* be sent to this program and */
                                     /* appear in the joblog.     */

    QUSCRTUI( name_lib, attribute, type, entry_length, key_type,
                key_length, immediate_update, optimize, authority,
                description, replace, &error_code );
}
Queue Management Instructions

This program illustrates how to call the system API, QUSCRTUQ, to create a user queue. This program must be run before any of the user queue examples (enq, deq, deqi, matqat and matqmsg).

Example

```c
/* -----------------------------------------------------------------------------*/
/* * Example Group: User Queues <mqueue.h> */
/* */
/* Function: none (just an example that can be used with */
/* the other User Queue examples) */
/* */
/* Description: This program illustrates how to call the system */
/* API, QUSCRTUQ, to create a user queue. This */
/* program should be run before any of the user */
/* queue examples: enq, deq, deqi, matqat, */
/* matqmsg. */
/* */
/* Since User Queues are often used for inter-job */
/* or inter-program communications, the entries on */
/* the queue are referred to as "messages". */
/* */
/* -----------------------------------------------------------------------------*/
#include <QSYSINC/H/QUUSCRTUQ>                 /* This header file contains */
/* the prototype for the */
/* QUUSCRTUQ System API */

/* Declare and initialize variables for the call to the QUUSCRTUQ API. */

int key_length = 0,                   /* Since this is not a keyed */
                 /* user queue, set this to zero */
maximum_message_size = 75,           /* Maximum message size */
initial_messages = 10,               /* Initial number of messages */
additional_messages = 50;            /* Number of additional messages */
                                    /* that can be accommodated. */
char type[] = "F";                    /* First-In-First-Out (FIFO)*/
char name_lib[] = "MYUSRQ MYLIB ";
char attribute[10] = "EXAMPLE ";     /* Arbitrary attribute name.*/
char authority[10] = "*ALL ";        /* Authority for the Index. */
char replace[10] = "*NO ";           /* Do not replace the queue */
                                    /* if it already exists. */
char description[50] = "User Queue used for inter-job communication";
```
mqueue.h

/*-----------------------------------------------*/
/* Define the "error code parameter" structure as defined in the */
/* System Programmer's Interface Reference manual. A typedef of a */
/* structure in this format is provided in the <QSYSINC/H/QUSEC> */
/* header file. The structure typedef is called Qus_EC_t and has */
/* different member names than the structure used in this example, */
/* although the example could be changed to use that typedef. */
/*-----------------------------------------------*/

struct {
    int bytes_available;
    int bytes_used;
    char exception_id[7];
    char reserved;
    char exception_data[1];
} error_code;

int main(void)
{

/*-----------------------------------------------*/
/* Call the 'QUSCRTUQ' System API to create the user queue with */
/* the specified name and library and the above characteristics */
/*-----------------------------------------------*/

    /* Let any exception generated */
    /* be sent to this program and */
    /* appear in the joblog. */

    QUSCRTUQ( name_lib, attribute, type, key_length,
            maximum_message_size, initial_messages,
            additional_messages, authority, description,
            replace, &error_code );
}

Establishing an Invocation Exit Program (ATIEXIT)

Format
#include <mipexec.h>

int atiexit (_OS_func_t *exit_handler, 
     void *parm)

Description
The atiexit function will establish an invocation exit handler at the oldest control boundary in an activation group. If the control boundary invocation is abnormally terminated, the registered exit handler will be invoked. The atiexit function is provided for OPM compatibility.

Invocations cancelled as a result of abort(), CEEMRCR API, RCLACTGRP option(*ABNORMAL), process termination, unhandled function check, and QMHSNDPM/QMHRSNEW APIs are considered abnormal termination.

Invocations cancelled as a result of a longjmp(), exit(), RCLACTGRP option(*NORMAL) are all considered normal termination.

When the atiexit function is called, the exit_handler parameter is copied into an internal buffer. Any changes made to these parameters after the function is called are not reflected. The exit_handler parameter is checked to ensure that it is a valid system pointer or NULL (this does not ensure that the system pointer is a valid pointer to a program object) If this check fails, the atiexit function returns a non-zero value and the previous value of the atiexit function does not change. If it succeeds, the exit_handler parameter is copied to an internal buffer, replacing its contents. Specifying a NULL pointer for exit_handler is equivalent to turning atiexit off.

Parameters
exit_handler(input)
A pointer to an OS_linkage exit handler program. This pointer is copied into an internal buffer when the atiexit function is called. Subsequent changes to this pointer will not be reflected.

parm (input)
A pointer to user data that will be passed to the exit handler. This pointer is copied into an internal buffer when the atiexit function is called. Subsequent changes to this pointer will not be reflected.

Notes on Usage
• The atiexit function cannot be called from within an atiexit-registered handler. This will result in a non-zero return value.
• The atiexit function cannot be called from the OPM default activation group. This will result in a non-zero return value.
• The caller of the exit handler acts like a control boundary with regards to exception percolation. If an unhandled exception is percolated out of an exit handler, then the following is done:
  1. The exception is handled, but the message remains in the joblog.
  2. The *LEFAIL condition is signalled to the caller of the control boundary.
Example
This example illustrates the use of the atiexit function.

```c
#include <mipgexec.h>

#pragma linkage(EXIT, OS)
void EXIT(void *);

#pragma linkage(REPORT, OS)
void REPORT(void);

int rc = 99;
_OS_func_t *exit_ptr = EXIT;

main() {
    /* establish the exit handler */
    atiexit(exit_ptr, &rc);

    /* The rest of the program goes here... */

    /* Program not found...MCH3401 is generated */
    REPORT();
}
```

#include <stdlib.h>

int main(int argc, char *argv[]) {
    /* Do some cleanup... */
    exit(*(int *)&argv[1]); /* Exit using the return code passed */
}
Compute Date Duration (CDD)

Format
#include <QSYSINC/MIH/MIDTM>

void cdd (_SPCPTCN date1,
         _SPCPTCN date2,
         _INST_Template_T2 *inst_t);

Description
The cdd function computes the date duration. The date specified by date2 is subtracted from the date specified by date1 and the value of the result is placed in date_duration. A negative value will be returned when date1 is less than date2.

Parameters
date_duration (input/output)
   Pointer to the location to receive the packed decimal duration.

date1 (input)
   Pointer to the first date.

date2 (input)
   Pointer to the second date.

inst_t (input)
   Pointer to the instruction template which defines the data definitional attributes for date_duration, date1, and date2.

Notes on Usage
- The DDATs for date1 and date2 must be identical.
- The builtin function _CDD also provides access to the MI instruction.
- A macro version is available.
**Example**
This example illustrates the use of the cdd function.

```c
/* televised */
/* Example Group: Date/Time/Timestamp <QSYSINC/MIH/MIDITM> */
/* */
/* Function: cdd (Compute Date Duration) */
/* */
/* Description: This example uses 'cdd' to compute the date */
duration of date1 and date2.
/* */
/* */
#include <QSYSINC/MIH/MIDITM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <decimal.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
define GREGORIAN_TIMELINE_END 5373485

int main(void) {
_INST_Template_T2 *inst_t2;
_DDAT_T *ddat_t1, *ddat_t2;
_Era_Table_T *era_t2;
_Calendar_Table_T *cal_t2;

char buffer[50];

int DDAT_Length, Calendar_Offset;
int DDAT_Size, Template_Size;
int DDAT_Offset1, DDAT_Offset2;

char source2_d[] = "1992-05-21";
char source1_d[] = "1993-01-30";
decimal(8,0) duration;

DDAT_Length = sizeof(_DDAT_T) +
 2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
  sizeof(_Era_Table_T) - 1;

Calendar_Offset = offsetof(_DDAT_T, Tables) +
  sizeof(_Era_Table_T);

DDAT_Size = 2*DDAT_Length +
  2*(sizeof(int)) + /* DDA Offset */
  10 + /* reserved */
  sizeof(short) + /* Num_DDATs */
  sizeof(int); /* DDA Size */

Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T2, DDA) +
```

18  MI Library Reference
```c
sizeof(int); /* ddat offset */

DDAT_Offset1 = (offsetof(_INST_Template_T2, DDAT) -
offsetof(_INST_Template_T2, DDAT_Size)) +
offsetof(int); /* DDAT offset */

DDAT_Offset2 = (offsetof(_INST_Template_T2, DDAT) -
offsetof(_INST_Template_T2, DDAT_Size)) +
DDAT_Length + offsetof(int); /* DDAT2 offset */

inst_t2 = (_INST_Template_T2 *)malloc(Template_Size);

/* Fill in Instruction Template */

memset(inst_t2, '\0', offsetof(_INST_Template_T2));

inst_t2->Template_Size = Template_Size;
inst_t2->DDAT_1 = 1;
inst_t2->DDAT_2 = 2;
inst_t2->DDAT_3 = 2;
inst_t2->Length_2 = 10;
inst_t2->Length_3 = 10;
inst_t2->DDAT_Size = DDAT_Size;
inst_t2->Num_DDATs = 2;
inst_t2->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t2->DDAT_Offset + 1) = DDAT_Offset2;

/* Set table pointers within instruction template */

ddat_t1 = (_DDAT_T *)&(inst_t2->DDAT_Offset + 2);

ddat_t2 = (_DDAT_T *)((char *)ddat_t1 + DDAT_Length);

era_t2 = (_Era_Table_T *)&(ddat_t2->Tables);

cal_t2 = (_Calendar_Table_T *)((char *)ddat_t2 + Calendar_Offset);

/* Fill in DDAT 1 */

ddat_t1->Format_Code = __DATE_DUR;

ddat_t1->DDAT_Length = DDAT_Length;

ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in DDAT2 */

ddat_t2->DDAT_Length = DDAT_Length;

ddat_t2->Format_Code = __ISO_DATE;

ddat_t2->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT2 */

era_t2->Num_Elems = 1;

era_t2->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;

era_t2->Element[0].Era_Name[0] = 'A';

era_t2->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT2 */

cal_t2->Num_Elems = 2;

cal_t2->Element->Effect_Date = GREGORIAN_TIMELINE_START;

cal_t2->Element->Type = 0x0001;
```

Machine Interface Library Functions 19
```c
memset(cal_t2->Element->reserved, '\0', 10);
(cal_t2->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t2->Element+1)->Type = 0;
memset((cal_t2->Element+1)->reserved, '\0', 10);

cdd(&duration, source1_d, source2_d, inst_t2);

printf("The date duration (YYYYMMDD) is %08d(0)\n", duration);
```

**Output**
The date duration (YYYYMMDD) is 00000809
Clear Bit in String (CLRBTS)

**Format**

```c
#include <QSYSINC/MIH/CLRBTS>

void clrbts (_SPCPTR bit_string, 
        unsigned int bit_offset);
```

**Description**
The clrbts function clears the bit in `bit_string` as indicated by `bit_offset`.

**Parameters**

- **bit_string (input)**
  A pointer to a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

- **bit_offset (input)**
  Indicates which bit of `bit_string` is to be set, with an offset of zero indicating the leftmost bit of the leftmost byte of `bit_string`. This value must be less than 64k.

**Notes on Usage**

- If the selected bit is beyond the end of the string, or the value of `bit_offset` is greater than or equal to 64k, the result of the operation is undefined.
- A macro version is available.
- The built-in function `_CLRBTS` also provides access to the MI instruction.

**Exceptions**

In some circumstances, if the selected bit is beyond the end of allocated storage, an MCH3203 exception may be signalled.
Example
This example illustrates the use of the clrbts function.

```c
#include <QSYSINC/MIH/CLRBTS>
#include <QSYSINC/MIH/SETBTS>
#include <QSYSINC/MIH/TSTBTS>
#include <stdio.h>

#define FALSE 0
#define TRUE 1

int main(void) {
    unsigned bit_string = 0;
    unsigned offset = 17;
    unsigned flag = FALSE;
    setbts(&bit_string, offset);
    flag = tstbts(&bit_string, offset);
    if (flag)
        printf("The %u'th bit has been set\n", offset);
    clrbts(&bit_string, offset);
    flag = tstbts(&bit_string, offset);
    if (!flag)
        printf("The %u'th bit has been cleared\n", offset);
}
```

Output
The 17'th bit has been set
The 17'th bit has been cleared
Compare Pointer for Space Addressability (CMPPSPAD)

**Format**
```
#include <QSYSINC/MIH/CMPPSPAD>

int cmppspad (_SPCPT space1,
              _SPCPT space2);
```

**Description**
The cmppspad function compares space addressability. The space addressability contained in the pointer specified by `space1` is compared with the space addressability defined by `space2`. If the two data objects pointed to by `space1` and `space2` are contained in the same space object, a value of 1 is returned, otherwise, a value of 0 is returned.

**Parameters**
- `space1` (input) and `space2` (input)
  - The two space pointers to be compared.

**Notes on Usage**
- `cmppspad` is a compatibility function. Use the `cmppspad` function for compatibility only, otherwise use the `cmpptra` function. The `cmpptra` function allows any type of pointer as arguments.
- Use C comparison operations to determine, for objects in the same space, if the offsets are greater than, less than, or equal.
- An open pointer that points to any type of pointer is a valid argument.

**Exceptions**
If an invalid space address is given for arguments 1 or 2, an MCH3601 is signalled.
Example
This example illustrates the use of the cmppspad function.

```c
#include <stdio.h>
#include <QSYSINC/MIH/CMPPSPAD>

int main(void) {
    int i_array[50], rc;
    _SPCPTT sp1 = i_array, sp2 = i_array + 25;

    rc = cmppspad(sp1, sp2);

    if (rc)
        printf("Objects are in the same space.\n");
    else
        printf("Objects are not in the same space.\n");
}

Output
Objects are in the same space.
Compare Pointer for Object Addressability (CMPPTRA)

Format
#include <QSYSINC/MIH/CMPPTRA>

int cmpptra (_ANYPTR pointer1,
             _ANYPTR pointer2);

Description
The cmpptra function compares the object addressability of 2 pointers. The object
addressed by pointer1 is compared with the object addressed by pointer2 to deter-
mine if both pointers are addressing the same object. A 1 (TRUE) is returned if the
pointers are addressing the same object, otherwise a 0 (FALSE) is returned.

For space pointers, TRUE is returned if they are addressing the same space. The
space offset portion of the pointer is ignored in the comparison.

For system pointers, TRUE is returned if the system pointer is compared with a
space pointer that addresses a byte in a space associated with the object that is
addressed by the system pointer.

Parameters
pointer1 (input)
    First pointer for compare operation. Must be a space pointer or system pointer.

pointer2 (input)
    Second pointer for compare operation. Must be a space pointer or system
    pointer.

Notes on Usage
• A macro version is available.
• The builtin function _CMPPTRA also provides access to the MI instruction.

Exceptions
If a pointer does not exist in argument 1 or 2, an MCH3601 will be signalled. If a
pointer type other than a system pointer or space pointer is passed, an MCH3602
will be signalled.
Example

This example illustrates the use of the cmpptra function.

```c
#include <QSYSINC/MIH/CMPPTRA>
#include <QSYSINC/MIH/RLSVSP>
#include <QSYSINC/MIH/SETSPFP>
#include <stdio.h>

#include <QSYSINC/H/QUSCRUS>

#define CREATION_SIZE 65536

int main(void) {
    __SPCPTR sp;
    __SYSPTR sysp;
    int error_code = 0, result;

    QUSCRUS("MYPACE QTEMP ", /* Create user space */
             "MYPACE ",
             CREATION_SIZE,
             "0",
             "ALL",
             "MYPACE example for Programmer's Reference",
             "YES",
             &error_code);

    /* Resolve to created space object */
    sysp = rsivsp(_Usrspc, "MYPACE", "QTEMP", __AUTH_OBJ_MGMT);
    sp = setppfp(sysp); /* Obtain space pointer into space */
    result = cmpptra(sp, sysp);

    if (result)
        printf("The pointers are addressing the same object\n");
    else
        printf("Error: the pointers are NOT addressing the same object\n");
}
```
Output
The pointers are addressing the same object
Compare Pointer Type (CMPPTRT)

Format
#include <QSYSINC/MIH/CMPPTRT>

int cmpptrt (_ANYPTR pointer,
             char type);

Description
The cmpptrt function compares the pointer type with the character scalar. If pointer
is of the same type as indicated by the character value type, then 1 (TRUE) is
returned, otherwise 0 (FALSE) is returned.

Parameters
pointer (input)
  May be any one of the valid AS/400 pointer types.

type (input)
  The name of the pointer type to compare to. All valid values are supplied
  through macros in the <milib.h> header file.

Notes on Usage
• The builtin function _CMPPTRT also provides access to the MI instruction.
• A macro version is available.

Exceptions
If the scalar value supplied for type is not valid, an MCH5003 is signalled.

Example
This example illustrates the use of the cmpptrt function.
/*---------------------------------------------*/
/*
/* Example Group: Pointer/Name Resolution Addressing */
/*
#include <QSYSINC/MIH_CMPPTRT>
/*
/* Function: cmpptrt (Compare Pointer Type) */
/*
/* Description: This example uses 'cmpptrt' to determine the */
/* pointer's type. */
/*
/*---------------------------------------------*/

#include <QSYSINC/MIH_CMPPTRT>
#include <stdio.h>

int main(int arg, char **argv) {
    int result = 0;

    result = cmpptrt(argv[1], _PTR_T_NULL);

    if (result)
        printf("The pointer is not set\n");
    else {
        result = cmpptrt(argv[1], _PTR_T_SYS);
        if (result)
            printf("The pointer is a system pointer\n");
        else {
            result = cmpptrt(argv[1], _PTR_T_SPC);
            if (result)
                printf("The pointer is a space pointer\n");
            else {
                result = cmpptrt(argv[1], _PTR_T_INV);
                if (result)
                    printf("The pointer is an invocation pointer\n");
                else {
                    result = cmpptrt(argv[1], _PTR_T_PROC);
                    if (result)
                        printf("The pointer is a function pointer\n");
                    else {
                        result = cmpptrt(argv[1], _PTR_T_LBL);
                        if (result)
                            printf("The pointer is a label pointer\n");
                        else {
                            result = cmpptrt(argv[1], _PTR_T_SUSP);
                            if (result)
                                printf("The pointer is a suspend pointer\n");
                            else
                                printf("Unexpected error\n");
                        }
                    }
                }
            }
        }
    }
}

Machine Interface Library Functions  29
Output
Output if the program is passed the argument NULL:
The pointer is not set
Compress Data (CPRDATA)

Format
```c
#include <QSYSINC/MIH/CPRDATA>

int cprdata (_SPCPRTR result,
             int result_length,
             _SPCPRTRCN source,
             int src_length);
```

Description
The `cprdata` function compresses user data of a specified length. If the compressed result is longer than the result area (as specified by `result_length`), the compression is stopped and only `result_length` bytes are stored.

The `cprdata` function returns the number of bytes in the compressed result. This value is always set to the full length of the result, which may be larger than `result_length`.

Parameters
- `result` (input/output)
  - Pointer to the result area to receive the compressed data.
- `result_length` (input)
  - The result area length.
- `source` (input)
  - The data to be compressed.
- `src_length` (input)
  - The length of the source.

Notes on Usage
- Only non-pointer data can be compressed, so any pointers in the data to be compressed are destroyed in the output of the Decompress Data instruction.
- A simple terse algorithm is used.
- The builtin function `_CPRDATA` also provides access to the MI instruction.
Example
This example illustrates the use of the cprdata function.

```c
#include <QSYSINC/MIH/CPRDATA>
#include <QSYSINC/MIH/DCPDATA>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {
    char cpr_data[SIZE];
    char dpr_data[SIZE] = "Good Day";
    int clen, dlen;

    clen = cprdata(cpr_data, SIZE, dpr_data, strlen(dpr_data));
    printf("The compressed length = %d\n", clen);

    dlen = dcpdata(dpr_data, SIZE, cpr_data);
    printf("The decompressed length = %d\n", dlen);

    if (memcmp(dpr_data, "Good Day", strlen(dpr_data)) != 0)
        printf("Error in decompression\n");
    else
        printf("Decompression successful\n");
}
```

Output
The compressed length = 26
The decompressed length = 8
Decompression successful
Compute Time Duration (CTD)

**Format**

```c
#include <QSYSINC/MIH/MIDTTM>

void ctd (_SPCPTTR time_duration,
         _SPCPTRCN time1,
         _SPCPTRCN time2,
         _INST_Template_T2 *inst_t);
```

**Description**

The ctd function determines the time duration. The time specified by `time2` is subtracted from the time specified by `time1` and the value of the result is placed in `time_duration`. A negative value will be returned when `time1` is less than `time2`.

**Parameters**

- **time_duration (input/output)**
  - Pointer to the location to receive the packed decimal duration.

- **time1 (input)**
  - Pointer to the first time.

- **time2 (input)**
  - Pointer to the second time.

- **inst_t (input)**
  - Pointer to the instruction template which defines the data definitional attributes for `time_duration`, `time1` and `time2`.

**Notes on Usage**

- The DDATs for `time1` and `time2` must be identical.
- The builtin function `_CTD` also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the ctd function.

```
#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <decimal.h>

int main(void) {

_INST_Template_T2  *inst_t2;
_DDAT_T       =dat_t1, *ddat_t2;

int     DDAT_Length, Calendar_Offset;
int     DDAT_Size, Template_Size;
int     DDAT_Offset1, DDAT_Offset2;

char source2_d[] = "11.05.30";
char source1_d[] = "14.20.46";
decimal(6,0) duration;

DDAT_Length = sizeof(_DDAT_T) +
       2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
       sizeof(_Era_Table_T) - 1;

Calendar_Offset = sizeof(_DDAT_T, Tables) +
       sizeof(_Era_Table_T);

DDAT_Size = 2*DDAT_Length +
       2*(sizeof(int)) +  /* DDAT_Offset */
       10 +  /* reserved4 */
       sizeof(short) +  /* Num_DDATs */
       sizeof(int);  /* DDAT_Size */

Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T2, DDAT) +
       sizeof(int);  /* ddat offset */

DDAT_Offset1 = (offsetof(_INST_Template_T2, DDAT) -
       offsetof(_INST_Template_T2, DDAT_Size)) +
       sizeof(int);  /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T2, DDAT) -
       offsetof(_INST_Template_T2, DDAT_Size)) +
       DDAT_Length + sizeof(int);  /* DDAT2 offset */
```

```
inst_t2 = (_INST_Template_T2 *)malloc(Template_Size);

/* Fill in Instruction Template */
memset(inst_t2, '\0', sizeof(_INST_Template_T2));

inst_t2->Template_Size = Template_Size;
inst_t2->DDAT_1 = 1;
inst_t2->DDAT_2 = 2;
inst_t2->DDAT_3 = 2;
inst_t2->Length_2 = 8;
inst_t2->Length_3 = 8;
inst_t2->DDAT_Size = DDAT_Size;
inst_t2->Num_DDATS = 2;
inst_t2->DDAT_Offset[0] = DDAT_Offset1;
*inst_t2->DDAT_Offset + 1) = DDAT_Offset2;

ddat_t1 = (_DDAT_T *)&(inst_t2->DDAT_Offset + 2);
/ddat_t2 = (_DDAT_T *)((char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT 1 */

ddat_t1->DDAT_Length = DDAT_Length;
/ddat_t1->Format_Code = _TIME_DUR;
/ddat_t1->Hour_Zone = 0;
/ddat_t1->Min_Zone = 0;
/ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in DDAT2 */

/ddat_t2->DDAT_Length = DDAT_Length;
/ddat_t2->Format_Code = _ISO_TIME;
/ddat_t2->Hour_Zone = 24;
/ddat_t2->Min_Zone = 60;
/ddat_t2->Calendar_Offset = Calendar_Offset;

ctd(&duration, source1_d, source2_d, inst_t2);
printf("The time duration (hhmmss) is %06d(6,0)\n", duration);
}

Output
The time duration (hhmmss) is 031516
```
# Compute Timestamp Duration (CTSD)

### Format

```c
#include <QYSINC/MIH/MIDTMM>

void ctsd (_SCPTR timestamp_duration,
           _SCPTRCN timestamp1,
           _SCPTRCN timestamp2,
           _INST_Template_T2 *inst_t);
```

### Description

The `ctsd` function determines the timestamp duration. The timestamp specified by `timestamp2` is subtracted from the timestamp specified by `timestamp1` and the value of the result is placed in `timestamp_duration`. A negative timestamp will be returned when `timestamp1` is less than `timestamp2`.

### Parameters

- **timestamp_duration** (input/output)
  - Pointer to the location to receive the packed decimal duration.

- **timestamp1** (input)
  - Pointer to the first timestamp.

- **timestamp2** (input)
  - Pointer to the second timestamp.

- **inst_t** (input)
  - Pointer to the instruction template which defines the data definitional attributes for `timestamp_duration`, `timestamp1`, and `timestamp2`.

### Notes on Usage

- The DDATs for `timestamp1` and `timestamp2` must be identical.
- The builtin function `_CTSD` also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the ctsd function.

```c
#include <QSYSINC/MIH/MIDTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <decimal.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {

    _INST_Template_T2 *inst_t2;
    _DDAT_T *ddat_t1, *ddat_t2;
    _Era_Table_T *era_t2;
    _Calendar_Table_T *cal_t2;

    int DDATE_Lengt, Calendar_Offset;
    int DDATE_Size, Template_Size;
    int DDATE_Offset1, DDATE_Offset2;

    char source2_d[] = "1992-02-21-09.15.02.000000";
    char source1_d[] = "1993-11-30-11.22.31.000000";
    decimal(20,6) duration;

    DDATE_Lengt = sizeof(_DDAT_T) +
                 2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
                 sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
                      sizeof(_Era_Table_T);

    DDATE_Size = 2*DDATE_Lengt +
                 2*(sizeof(int)) + /* DDATE_Offset */
                 10 + /* reserved4 */
                 sizeof(short) + /* Num_DDATs */
                 sizeof(int); /* DDATE_Size */

    Template_Size = 2*DDATE_Lengt + offsetof(_INST_Template_T2, DDAT) +
                    sizeof(int); /* ddat offset */
```
DDAT_Offset1 = (offsetof(_INST_Template_T2, DDAT) -
offsetof(_INST_Template_T2, DDAT_Size)) +
sizeof(int); /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T2, DDAT) -
offsetof(_INST_Template_T2, DDAT_Size)) +
DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t2 = (_INST_Template_T2 *)malloc(Template_Size);
/* Fill in Instruction Template */
memset(inst_t2, '\0', sizeof(_INST_Template_T2));

inst_t2->Template_Size = Template_Size;
inst_t2->DDAT_1 = 1;
inst_t2->DDAT_2 = 2;
inst_t2->DDAT_3 = 2;
inst_t2->Length_2 = 26;
inst_t2->Length_3 = 26;
inst_t2->DDAT_Size = DDAT_Size;
inst_t2->Num_DDATs = 2;
inst_t2->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t2->DDAT_Offset + 1) = DDAT_Offset2;

ddat_t1 = (_DDAT_T *)&(inst_t2->DDAT_Offset + 2);
ddat_t2 = (_DDAT_T *)&(char *)ddat_t1 + DDAT_Length;
era_t2 = (_Era_Table_T *)&(ddat_t2->Tables);
cal_t2 = (_Calendar_Table_T *)&(char *)ddat_t2 + Calendar_Offset;
/* Fill in DDAT 1 */

ddat_t1->DDAT_Length = DDAT_Length;
ddat_t1->Format_Code = _TIMESTAMP_DUR;
ddat_t1->Hour_Zone = 0;
ddat_t1->Min_Zone = 0;
ddat_t1->Calendar_Offset = Calendar_Offset;
/* Fill in DDAT2 */

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = _SAA_TIMESTAMP;
ddat_t2->Hour_Zone = 24;
ddat_t2->Min_Zone = 60;
ddat_t2->Calendar_Offset = Calendar_Offset;
/* Fill in Era Table for DDAT2 */

era_t2->Num_Elems = 1;
era_t2->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t2->Element[0].Era_Name[0] = 'A';
era_t2->Element[0].Era_Name[1] = 'D';
/* Fill in Calendar Table for DDAT2 */

cal_t2->Num_Elems = 2;
cal_t2->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t2->Element->Type = 0x0000;
memset(cal_t2->Element->reserved, '\0', 10);
(cal_t2->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t2->Element+1)->Type = 0;
memset((cal_t2->Element+1)->reserved, '\0', 10);

ctsd(&duration, source1_d, source2_d, inst_t2);

printf("The timestamp duration (YYYYMMDDhhmmssuuuuuuu) is %020D(20,6)\n", duration);
}

Output
The timestamp duration (YYYYMMDDhhmmssuuuuuuu) is 0010909020729.000000
Convert Binary Synchronous Communications to Character (CVTBC)

Format
#include <QSYSINC/MIH/CVTBC>

int cvtbc (_SPCPTTR receiver, 
        unsigned int rcvr_length, 
        _CVTBC_Control_T *controls, 
        _SPCPTRCN source, 
        unsigned int src_length);

Description
The cvtbc function converts a string value from the BSC (binary synchronous communications) compressed format to a character string. The function converts source using the information contained in controls and places the result into the location specified by receiver.

Parameters
receiver (input/output)
    Pointer to the location to contain the result of the conversion.

rcvr_length (input)
    The length of the receiver. The length must be between 1 and 32767.

controls (input/output)
    Pointer to the controls template containing additional information for the conversion.

source (input)
    Pointer to the location containing the source in BSC compressed format.

src_length (input)
    The length of the source. The length must be between 1 and 32767.

Notes on Usage
- The builtin function _CVTBC also provides access to the MI instruction.

Return Code
Values returned by the function cvtbc are set as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Completed Record</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td>1</td>
<td>Truncated Record</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the cvtbc function.

```c
#define SIZE 50

int main(void) {
    _CVTBC_Control_T cvtcb_t = {0, 0x01};
    _CVTBC_Control_T cvtbc_t = {0, 0x01};
    char cb[SIZE] = "This is the input string";
    char bc[SIZE];
    int bc_rc, cb_rc;

    cb_rc = cvtcb(bc, SIZE, &cvtcb_t, cb, SIZE);
    printf("The return code from cvtcb is \%d\n", cb_rc);

    bc_rc = cvtbc(cb, SIZE, &cvtbc_t, bc, SIZE);
    printf("The return code from cvtbc is \%d\n", bc_rc);

    if (memcmp(cb, "This is the input string", strlen(cb)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}
```

Output
The return code from cvtcb is 0
The return code from cvtbc is 0
Conversion successful
Convert Character to Binary Synchronous Communications (CVTCB)

Format

```c
#include <QSYSINC/MIH/CVTCB>

int cvtcb (_SPCPTTR receiver,
           unsigned int rcvr_length,
           _CVTCB_Control_T *controls,
           _SPCPTRCN source,
           unsigned int src_length);
```

Description

The cvtcb function converts a string value from character to BSC (binary synchronous communications) compressed format. The function converts `source` using the information contained in `controls` and places the result into the location specified by `receiver`.

Parameters

- **receiver (input/output)**
  - Pointer to the location to contain the result of the conversion.

- **rcvr_length (input)**
  - Length of the receiver. Length must be between 1 and 32767.

- **controls (input/output)**
  - Pointer to the controls template containing additional information for the conversion.

- **source (input)**
  - Pointer to the location containing the source string to be converted.

- **src_length (input)**
  - The length of the source. The length must be between 1 and 32767.

Notes on Usage

- The builtin function _CVTCB also provides access to the MI instruction.

Return Code

Values returned by the function cvtcb are set as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the cvtcb function.

```c
#include <QSYSINC/MIH/CVTCB>
#include <QSYSINC/MIH/CVTBC>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {

    _CVTCB_Control_T cvtcb_t = {0, 0x01};
    _CVTBC_Control_T cvtbc_t = {0, 0x01};
    char cb[SIZE] = "This is the input string";
    char bc[SIZE];
    int bc_rc, cb_rc;

    cb_rc = cvtcb(bc, SIZE, &cvtcb_t, cb, SIZE);
    printf("The return code from cvtcb is %d\n", cb_rc);

    bc_rc = cvtbc(cb, SIZE, &cvtbc_t, bc, SIZE);
    printf("The return code from cvtbc is %d\n", bc_rc);

    if (memcmp(cb, "This is the input string", strlen(cb)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}
```

Output
The return code from cvtcb is 0
The return code from cvtbc is 0
Conversion successful
Convert Eight Bit Character to Hex Nibbles (CVTCH)

**Format**

```c
#include <QSYSINC/MIH/CVTCH>

void cvtch (_SPCPTTR receiver,
            _SPCPTRCN source,
            int size);
```

**Description**
The cvtch function takes each character (8-bit value) of source and converts it to a hexadecimal digit (4-bit value) and places into receiver.

**Parameters**

- **receiver (input/output)**
  Pointer to a 4-bit hexadecimal receiver.
- **source (input)**
  Pointer to an 8-bit character source.
- **size (input)**
  The length in bytes of the source.

**Notes on Usage**
- The characters in source must relate to valid hexadecimal digits or an exception is signaled.
  - **Characters** | **Hex digits**
    - Hex F0-Hex F9 = Hex 0-Hex 9
    - Hex C1-Hex C6 = Hex A-Hex F
- This is a compatibility function to provide the same semantics as the CVTCH MI instruction.
- If 0 is specified as size, no action is taken.

**Exceptions**
If any of the input characters are specified incorrectly, an MCH3601 or MCH0601 will be signalled.

Other exceptions possible from this function are:
- MCH0602 - Boundary alignment
- MCH0801 - Parameter Reference Violation
- MCH3402 - Object Destroyed
- MCH3602 - Pointer Type Invalid
- MCH6801 - Object Domain Violation
Example
This example illustrates the use of the cvtch function.

```c
/* Example Group: Computation and Branching <QSYSINC/MIH/CVTCH> */
/* Function: cvtch (Convert Character to Hexadecimal) */
/* Description: This example uses 'cvtch' and 'cvthc' to 
  convert the input string from character to hexadecimal then back to character. */
/* */
*/--------------------------------------------------------------------------*/

#include <QSYSINC/MIH/CVTCH>
#include <QSYSINC/MIH/CVTHC>
#include <string.h>
#include <stdio.h>

int main(void) {
  char c_array[9] = "A93B1FECO";
  char h_array[6] = {0xA9, 0x3B, 0x1F, 0xEC, 0x0D, 0x00};

  char char_array[9];
  char hex_array[6];

  memcpy(char_array, c_array, sizeof(char_array));
  cvtch(hex_array, char_array, sizeof(char_array));
  if (memcmp(hex_array, h_array, sizeof(h_array)) != 0)
    printf("Error in conversion\n");
  cvthc(char_array, hex_array, sizeof(hex_array)*2);
  if (memcmp(char_array, c_array, sizeof(char_array)) != 0)
    printf("Error in conversion\n");
}

Output
** no screen output **
Convert a Character String to Multi-Leaving Remote Job Entry (CVTCM)

Format
#include <QSYSINC/MIH/CVTCM>

int cvtcm (_SPCPTT receiver,  
             unsigned int rcvr_length,  
             _CVTCM_Control_T *controls,  
             _SPCPTCRN source,  
             unsigned int src_length);

Description
The cvtcm function converts a string of characters to Multi-Leaving Remote Job Entry (MRJE) compressed format. The information supplied in the controls is used to guide the conversion.

Parameters
receiver (input/output)
    Pointer to the storage location to receive the results of the conversion.
rcvr_length (input)
    Length of receiver. Length must be between 1 and 32767.
controls (input/output)
    Pointer to the controls template.
source (input)
    Pointer to the source.
src_length (input)
    The length of the source to be converted. The length must be between 1 and 32767.

Notes on Usage
- The builtin function _CVTCM also provides access to the MI instruction.

Return Code
Values returned by the function cvtcm are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the function cvtcm.

```c
#include <QSYSINC/MIH/CVTCM>
#include <QSYSINC/MIH/CVTM>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {
    char cm[SIZE] = "This is the input string";
    char mc[SIZE];
    _CVTCM_Control_T cvtc_m = {0, 0, 0x00, 0x10, 0x00, 0x00, 0x00,
                              0x0f};
    _CVTM_Control_T cvtmc_t = {0, 0, 0x00, 0x10};
    int cm_rc, mc_rc;

    cm_rc = cvtcm(mc, SIZE, &cvtcm_t, cm, strlen(cm));
    printf("The return code from cvtcm is %d\n", cm_rc);

    mc_rc = cvtmc(cm, SIZE, &cvtmc_t, mc, cvtmc_t.Receiver);
    printf("The return code from cvtmc is %d\n", mc_rc);

    if (memcmp(cm, "This is the input string", strlen(cm)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}
```

Output
The return code from cvtcm is 0
The return code from cvtmc is 0
Conversion successful
Convert a Character String to System Network Architecture (CVTCS)

Format
#include <QSYSINC/MIH/CVTCS>

int cvtcs (_SPCPTTr receiver,
           unsigned int rcvr_length,
           _CVTCS_Control_T *controls,
           _SPCPTRCN source,
           unsigned int src_length);

Description
The cvtcs function converts the source from character format to Systems Network Architecture (SNA) format. The information supplied in controls is used to guide the conversion.

Parameters
receiver (input/output)
  Pointer to the storage location to receive the results of the conversion.

rcvr_length (input)
  Length of the receiver. The length must be between 1 and 32767.

controls (input/output)
  Pointer to the controls template.

source (input)
  Pointer to the source.

src_length (input)
  The length of the source. The length must be between 1 and 32767.

Notes on Usage
- The builtin function _CVTCS also provides access to the MI instruction.

Return Code
Values returned by the function cvtcs are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the cvtcs function.

```c
#include <QSYSINC/MIH/CVTCS>
#include <QSYSINC/MIH/CVTSC>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {

    char          cs[SIZE] = "This is the input string";
    char          sc[SIZE];
    int           cs_rc, sc_rc;
    _CVTCS_Control_T cvtcs_t = {0x00, 0x80, 0x20, 0x00, 0x00, 0x00, 0x34, 0x40, 0x40};
    _CVTSC_Control_T cvtsc_t = {0x00, 0x80, 0x32, 0x34, 0x00, 0x00, _CVTSC_CONVERT_NO_TRANS, 0x00, 0};

    cs_rc = cvtcs(sc, SIZE, &cvtcs_t, cs, strlen(cs));
    printf("The return code from cvtcs is %d\n", cs_rc);

    sc_rc = cvtsc(sc, SIZE, &cvtsc_t, sc, cvtcs_t.Receiver);
    printf("The return code from cvtsc is %d\n", sc_rc);

    if (memcmp(cs, "This is the input string", strlen(cs)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}

Output
The return code from cvtcs is 0
The return code from cvtsc is 0
Conversion successful
Convert Date (CVTD)

Format
#include <QSYSINC/MIH/MIDTM>

void cvtd (_SPCPR result_date,
       _SPCPRCN source_date,
       _INST_Template_T1 *inst_t);

Description
The cvtd function converts one calendar date to another calendar date. The date
specified in source_date is converted to another calendar external or internal pres-
entation and placed in result_date.

Parameters
result_date (input/output)
    Pointer to the location to receive the converted date.

source_date (input)
    Pointer to the source date.

inst_t (input)
    Pointer to the instruction template which defines the data definitional attributes
    for result_date and source_date.

Notes on Usage
• The builtin function _CVTD also provides access to the MI instruction.
• A macro version is available.
Example
This example illustrates the use of the cvtd function.

#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {
  _INST_Template_T1 *inst_t1;
  _DDAT_T *ddat_t1, *ddat_t2;
  _Era_Table_T *era_t1, *era_t2;
  _Calendar_Table_T *cal_t1, *cal_t2;

  int DDAT_Lengt, Calendar_Offset;
  int DDAT_Size, Template_Size;
  int DDAT_Offset1, DDAT_Offset2;
  char result_d[10];
  char source_d[10] = "1993-01-17";

  DDAT_Lengt = sizeof(_DDAT_T) +
              2*sizeof(_Calendar_Table_T)) - sizeof(short) +
              sizeof(_Era_Table_T) - 1;

  Calendar_Offset = offsetof(_DDAT_T, Tables) +
                     sizeof(_Era_Table_T);

  DDAT_Size = 2*DDAT_Lengt +
               2*(sizeof(int)) + /* DDAT_Offset */
               10 /* reserved */ +
               sizeof(short) + /* Num_DDATs */
               sizeof(int); /* DDAT_Size */

  Template_Size = 2*DDAT_Lengt + offsetof(_INST_Template_T1, DDAT) +
                  sizeof(int); /* ddat offset */
DDAT_Offset1  = (offsetof(_INST_Template_T1, DDAT) -
    offsetof(_INST_Template_T1, DDAT_Size)) +
    sizeof(int); /* DDAT offset */

DDAT_Offset2  = (offsetof(_INST_Template_T1, DDAT) -
    offsetof(_INST_Template_T1, DDAT_Size)) +
    DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t1 = (_INST_Template_T1 *)malloc(Template_Size);
/* Fill in Instruction Template */
memset(inst_t1, '\0', sizeof(_INST_Template_T1));

inst_t1->Template_Size  = Template_Size;
inst_t1->DDAT_1        = 1;
inst_t1->DDAT_2        = 2;
instant_t1->Length_1   = 10;
instant_t1->Length_2   = 10;
instant_t1->DDAT_Size  = DDAT_Size;
instant_t1->Num_DDATS  = 2;
instant_t1->DDAT_Offset[0]  = DDAT_Offset1;
*(inst_t1->DDAT_Offset + 1) = DDAT_Offset2;

ddat_t1 = (_DDAT_T *)&(inst_t1->DDAT_Offset + 2);
era_t1 = (_Era_Table_T *)&(ddat_t1->Tables);
cal_t1 = (_Calendar_Table_T *)((char *)ddat_t1 + Calendar_Offset);

ddat_t2 = (_DDAT_T *)((char *)ddat_t1 + DDAT_Offset);
era_t2 = (_Era_Table_T *)&(ddat_t2->Tables);
cal_t2 = (_Calendar_Table_T *)((char *)ddat_t2 + Calendar_Offset);
/* Fill in DDAT1 */

ddat_t1->DDAT_Length   = DDAT_Length;
ddat_t1->Format_Code  = _USA_DATE;
ddat_t1->Calendar_Offset = Calendar_Offset;
/* Fill in Era Table for DDAT1 */

era_t1->Num_Elems   = 1;
era_t1->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t1->Element[0].Era_Name[0] = 'A';
era_t1->Element[0].Era_Name[1] = 'D';
/* Fill in Calendar Table for DDAT1 */

cal_t1->Num_Elems     = 2;
cal_t1->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t1->Element->Type   = 0x0001;
memset(cal_t1->Element->reserved, '\0', 10);
(cal_t1->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t1->Element+1)->Type = 0;
memset((cal_t1->Element+1)->reserved, '\0', 10);
/* Fill in DDAT2 */

ddat_t2->DDAT_Length  = DDAT_Length;
ddat_t2->Format_Code = _ISO_DATE;
ddat_t2->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT2 */

era_t2->Num_Elems = 1;
era_t2->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t2->Element[0].Era_Name[0] = 'A';
era_t2->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT2 */

cal_t2->Num_Elems = 2;
cal_t2->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t2->Element->Type = 0x0001;
memset(cal_t2->Element->reserved, '\0', 10);
(cal_t2->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t2->Element+1)->Type = 0;
memset((cal_t2->Element+1)->reserved, '\0', 10);

cvtd(result_d, source_d, inst_t1);

result_d[10] = '\0';
printf("The converted date is %s\n", result_d);
}

Output
The converted date is 01/17/1993
Convert External Form to Numeric Value (CVTEFN)

Format
```
#include <QSYSINC/MIH/CVTEFN>

int cvtefni (_SPCPTRCN source,
             unsigned int src_length,
             char mask[3]);

double cvtefn (_SPCPTRCN source,
               unsigned int src_length,
               char mask[3]);
```

Description
The cvtefni and cvtefn functions scans a character string for a valid decimal number in display format, removes the display character, and converts the result to integer (in the case of cvtefni) or double (in the case of cvtefn).

Parameters
- **source (input)**
  The character string value. A valid decimal number in display format.

- **src_length (input)**
  Length in bytes of source. The length must be between 1 and 32767.

- **mask (input)**
  The 3-byte character mask used in the conversion. Byte 1 of the mask indicates the byte value that is to be used for the currency symbol. Byte 2 of the mask indicates the byte value to be used for the comma symbol. Byte 3 of the mask indicates the byte value to be used for the decimal point symbol.

Notes on Usage
- The builtin functions _CVTEFN1 and CVTEFN2 also provide access to the MI instruction.
Example
This example illustrates the use of the cvtefn function.

```c
#include <QSYSINC/MIH/CVTEFN>
#include <string.h>
#include <stdio.h>

int main(void) {
    int integer_result;
    double double_result;
    char integer_source[] = "$6,025";
    char double_source[] = "$1,605.25";
    char mask[] = ",."

    integer_result = cvtefn(integer_source, strlen(integer_source), mask);
    printf("The integer result is %d\n", integer_result);

    double_result = cvtefn(double_source, strlen(double_source), mask);
    printf("The double result is %f\n", double_result);
}
```

Output
The integer result is 6025
The double result is 1605.250000
Convert Hex to Character (CVTHC)

**Format**
```
#include <QSYSINC/MI/CVTHC>

void cvthc (_SPCPTR receiver,
          _SPCPTRCN source,
          int size);
```

**Description**
The cvthc function takes each hexadecimal digit (4-bit value) of `source` and converts it to a character digit (8-bit value) and places into `receiver`.

**Parameters**
- **receiver (input/output)**
  Pointer to an 8-bit character value.
- **source (input)**
  Pointer to a 4-bit hexadecimal value.
- **size (input)**
  The length in nibbles of the source.

**Notes on Usage**
- The characters in `source` must relate to valid hexadecimal digits or an exception is signaled.
  
  **Hex Digits     Characters**
  Hex 0-Hex 9 = Hex F0-Hex F9
  Hex A-Hex F = Hex C1-Hex C6
- This is a compatibility function to provide the same semantics as the CVTHC MI instruction.
- If 0 is specified as `size`, no action is taken.

**Exceptions**
If any of the input characters are specified incorrectly, an MCH3601 or MCH0601 will be signalled.

Other exceptions possible from this function are:
- MCH0602 - Boundary alignment
- MCH0801 - Parameter Reference Violation
- MCH3402 - Object Destroyed
- MCH3602 - Pointer Type Invalid
- MCH6801 - Object Domain Violation
Example
This example illustrates the use of the cvthc function.

```c
#include <QSYSINC/MIH/CVTHC>
#include <QSYSINC/MIH/CTVCH>
#include <string.h>
#include <stdio.h>

int main(void) {

    char c_array[9] = "A03BFECB";
    char h_array[6] = {0xA9, 0x3B, 0x1F, 0xEC, 0xD0, 0x00};

    char char_array[9];
    char hex_array[6];

    memcpy(char_array, c_array, sizeof(char_array));
    cvthc(hex_array, char_array, sizeof(char_array));
    if (memcmp(hex_array, h_array, sizeof(hex_array)) != 0) {
        printf("Error in conversion\n");
    }
    cvthc(char_array, hex_array, sizeof(hex_array) * 2);
    if (memcmp(char_array, c_array, sizeof(char_array)) != 0) {
        printf("Error in conversion\n");
    }
}
```

Output
** no screen output **
Convert Multi-Leaving Remote Job Entry to Character (CVTMC)

Format

```c
#include <QSYSINC/MIH/CVTMC>

int cvtmc (_SPCPTTR receiver,
           unsigned int rcvr_length,
           _CVTMC_Control_T *controls,
           _SPCPTRCN source,
           unsigned int src_length);
```

Description

The cvtmc function converts a character string from Multi-Leaving Remote Job Entry (MRJE) compressed format to character format. The information supplied in the `controls` is used to guide the conversion.

Parameters

receiver (input/output)
- Pointer to the storage location to receive the results of the conversion.

rcvr_length (input)
- Length of the receiver. The length must be between 1 and 32767.

controls (input/output)
- Pointer to the controls template.

source (input)
- Pointer to the source.

src_length (input)
- Length of the source, including the null terminator. The length must be between 1 and 32767.

Notes on Usage

- The builtin function _CVTMC also provides access to the MI instruction.

Return Code

Values returned by the function cvtmc are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the cvtmc function.

#include <QSYSINC/MIH/CVTMC>
#include <QSYSINC/MIH/CVTCM>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {
    char cm[SIZE] = "This is the input string";
    char mc[SIZE];

    _CVTMC_Control_T cvtmc_t = {0, 0x00, 0x10, 0x00, 0x00, 0x00, 0xf1};
    _CVTMC_Control_T cvtmc_t = {0, 0, 0x00, 0x10};
    int cm rc, mc rc;

    cm rc = cvtcm(mc, SIZE, &cvtmc t, cm, strlen(cm));
    printf("The return code from cvtcm is %d\n", cm rc);

    mc rc = cvtmc(cm, SIZE, &cvtmc t, mc, cvtmc t. Receiver);
    printf("The return code from cvtmc is %d\n", mc rc);

    if (memcmp(cm, "This is the input string", strlen(cm)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}

Output
The return code from cvtcm is 0
The return code from cvtmc is 0
Conversion successful
Convert System Network Architecture to Character (CVTSC)

Format
#include <QSYSINC/MIH/CVTSC>

int cvtsc (_SPCPTTR receiver,
           unsigned int rcvr_length,
           _CVTSC_Control_T *controls,
           _SPCPTCRN source,
           unsigned int src_length);

Description
The cvtsc function converts a character string from Systems Network Architecture (SNA) format to character format. The information supplied in controls is used to guide the conversion.

Parameters
receiver (input/output)
  Pointer to the storage location to receive the results of the conversion.

rcvr_length (output)
  Length of the receiver. The length must be between 1 and 32767.

controls (input/output)
  Pointer to the controls template.

source (input)
  Pointer to the source.

src_length (input)
  Length of the source. The length must be between 1 and 32767.

Notes on Usage
• The builtin function _CVTSC also provides access to the MI instruction.

Return Code
Values returned by the function cvtsc are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td>0</td>
<td>Source Exhaustered</td>
</tr>
<tr>
<td>1</td>
<td>Escape Code Encountered</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the cvtsc function.

```c
#include <QSYSINC/MIH/CVTSC>
#include <QSYSINC/MIH/CVTCS>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {
    char cs[SIZE] = "This is the input string";
    char sc[SIZE];
    int cs_rc, sc_rc;
    _CVTCS_Control_T cvtcs_t = {0, 0x80, 0x20, 0x00, 0x00, 0x00, 0x00};
    _CVTSC_Control_T cvtsc_t = {0, 0x80, 0x32, 0x34, 0x00, 0x00, _CVTSC_CONVERT_NO_TRANS, 0x00, 0};

    cs_rc = cvtcs(sc, SIZE, &cvtcs_t, cs, strlen(cs));
    printf("The return code from cvtcs is %d\n", cs_rc);

    sc_rc = cvtsc(cs, SIZE, &cvtsc_t, sc, cvtcs_t.Receiver);
    printf("The return code from cvtsc is %d\n", sc_rc);

    if (memcmp(cs, "This is the input string", strlen(cs)) != 0)
        printf("Error in conversion\n");
    else
        printf("Conversion successful\n");
}
```

Output
The return code from cvtcs is 0
The return code from cvtsc is 0
Conversion successful
Convert Time (CVTT)

Format
#include <QSYSINC/MIH/MIDTTM>

void cvtt (_SPCPRTR result_time,
           _SPCPRTRCN source_time,
           _INST_Template_T1 *inst_t);

Description
The cvtt function converts the time from one format to another format. The time specified in source_time is converted to another external or internal presentation and placed in result_time.

Parameters
result_time (input/output)
  Pointer to the location to receive the converted time.

source_time (input)
  Pointer to the source time.

inst_t (input)
  Pointer to the instruction template which defines the data definitional attributes for result_time and source_time.

Notes on Usage
• The builtin function _CVTT also provides access to the MI instruction.
• A macro version is available.
Example

This example illustrates the use of the cvtt function.

```c
/* Example Group: Date/Time/Timestamp <QSYSINC/MIH/MIDTTM> */
/* */
/* Function: cvtt (Convert Time) */
/* */
/* Description: This example uses 'cvtt' to convert the time */
/* from ISO time to USA time. */
/* */
/* */
#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {

  _INST_Template_T1  *inst_t1;
  _DDAT_T             *ddat_t1, *ddat_t2;

  int                  DDAT_Length, Calendar_Offset;
  int                  DDAT_Size, Template_Size;
  int                  DDAT_Offset1, DDAT_Offset2;

  char                 result_t[9];
  char                 source_t[9] = "11.05.30";

  DDAT_Length = sizeof(_DDAT_T) +
               2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
               sizeof(_Era_Table_T) - 1;

  Calendar_Offset = offsetof(_DDAT_T, Tables) +
                    sizeof(_Era_Table_T);

  DDAT_Size = 2+DDAT_Length +
             2*(sizeof(int)) + /* DDAT_Offset */
             10 + /* reserved4 */
             sizeof(short) + /* Num_DDATs */
             sizeof(int); /* DDAT_Size */

  Template_Size = 2+DDAT_Length + offsetof(_INST_Template_T1, DDAT) +
                  sizeof(int); /* ddat offset */

  DDAT_Offset1 = (offsetof(_INST_Template_T1, DDAT) -
                  offsetof(_INST_Template_T1, DDAT_Size)) +
                 sizeof(int); /* DDAT2 offset */

  DDAT_Offset2 = (offsetof(_INST_Template_T1, DDAT) -
                  offsetof(_INST_Template_T1, DDAT_Size)) +
                 DDAT_Length + sizeof(int); /* DDAT2 offset */

  inst_t1 = (_INST_Template_T1 *)malloc(Template_Size);
```

Machine Interface Library Functions   63
/* Fill in Instruction Template */

memset(inst_t1, '\0', sizeof(_INST_Template_T1));

inst_t1->Template_Size = Template_Size;
inst_t1->DDAT_1 = 1;
inst_t1->DDAT_2 = 2;
inst_t1->Length_1 = 8;
inst_t1->Length_2 = 8;
inst_t1->DDAT_Size = DDAT_Size;
inst_t1->Num_DDATs = 2;
inst_t1->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t1->DDAT_Offset + 1)= DDAT_Offset2;

ddat_t1 = (_DDAT_T *)(inst_t1->DDAT_Offset + 2);
 ddat_t2 = (_DDAT_T *)(char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT1 */

ddat_t1->DDAT_Length = DDAT_Length;
 ddat_t1->Format_Code = _USA_TIME;
 ddat_t1->Hour_Zone = 24;
 ddat_t1->Min_Zone = 60;
 ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in DDAT2 */

 ddat_t2->DDAT_Length = DDAT_Length;
 ddat_t2->Format_Code = _ISO_TIME;
 ddat_t2->Hour_Zone = 24;
 ddat_t2->Min_Zone = 60;
 ddat_t2->Calendar_Offset = Calendar_Offset;

cvtt(result_t, source_t, inst_t1);

result_t[8] = '\0';
printf("The converted time is %s\n", result_t);
}

Output
The converted time is 11:05 AM
Convert Timestamp (CVTTS)

Format
#include <QSYSINC/MIH/CVTTS>

void cvtts (_SPC_PTR result_timestamp,
            _SPC_PTRCN source_timestamp,
            _INST_Template_TI *inst_t);

Description
The cvtts function converts the timestamp from one format to another. The
timestamp specified in source_timestamp is converted to another external or
internal presentation and placed in result_timestamp.

Parameters
result_timestamp (input/output)
    Pointer to the location to receive the converted timestamp.

source_timestamp (input)
    Pointer to the source timestamp.

inst_t (input)
    Pointer to the instruction template which defines the data definitional attributes
    for result_timestamp. and source_timestamp.

Notes on Usage
- The builtin function _CVTTS also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the cvtts function.

```c
#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdef.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {

  _INST_Template_T1  *inst_t1;
  _DDAT_T             *ddat_t1, *ddat_t2;
  _Era_Table_T        *era_t1, *era_t2;
  _Calendar_Table_T   *cal_t1, *cal_t2;

  int                     DDAT_Length, Calendar_Offset;
  int                     DDAT_Size, Template_Size;
  int                     DDAT_Offset1, DDAT_Offset2;

  char                    result_ts[27];
  char                    source_ts[15] = "19930117110530";

  DDAT_Length = sizeof(_DDAT_T) +
    2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
    sizeof(_Era_Table_T) - 1;

  Calendar_Offset = offsetof(_DDAT_T, Tables) +
    sizeof(_Era_Table_T);

  DDAT_Size = 2*DDAT_Length +
    2*(sizeof(int)) + /* DDAT_Offset */
    10 + /* reserved4 */
    sizeof(short) + /* Num_DDATs */
    sizeof(int); /* DDAT_Size */

  Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T1, DDAT) +
    sizeof(int); /* ddat offset */

  DDAT_Offset1 = (offsetof(_INST_Template_T1, DDAT) -
```
```c
offsetof(_INST_Template_T1, DDAT_Size) +
sizeof(int); /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T1, DDAT) -
offsetof(_INST_Template_T1, DDAT_Size)) +
DDAT_Len + sizeof(int); /* DDAT2 offset */

inst_t1 = (_INST_Template_T1 *)&malloc(Template_Size);
/* Fill in Instruction Template */
memset(inst_t1, '\0', sizeof(_INST_Template_T1));

inst_t1->Template_Size = Template_Size;
inst_t1->DDAT_1 = 1;
inst_t1->DDAT_2 = 2;
inst_t1->Length_1 = 26;
inst_t1->Length_2 = 14;
inst_t1->DDAT_Size = DDAT_Size;
inst_t1->Num_DDATs = 2;
inst_t1->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t1->DDAT_Offset + 1) = DDAT_Offset2;

ddat_t1 = (_DDAT_T *)&(inst_t1->DDAT_Offset + 2);
era_t1 = (_Era_Table_T *)&(ddat_t1->Tables);
cal_t1 = (_Calendar_Table_T *)&(char *)ddat_t1 + Calendar_Offset);
ddat_t2 = (_DDAT_T *)&(char *)ddat_t1 + DDAT_Length);
era_t2 = (_Era_Table_T *)&(ddat_t2->Tables);
cal_t2 = (_Calendar_Table_T *)&(char *)ddat_t2 + Calendar_Offset);
/* Fill in DDAT1 */

ddat_t1->DDAT_Len = DDAT_Len;
ddat_t1->Format_Code = _SAA_TIMESTAMP;
ddat_t1->Hour_Zone = 24;
ddat_t1->Min_Zone = 60;
ddat_t1->Calendar_Offset = Calendar_Offset;
/* Fill in Era Table for DDAT1 */
era_t1->Num_Elems = 1;
era_t1->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t1->Element[0].Era_Name[0] = 'A';
era_t1->Element[0].Era_Name[1] = 'D';
/* Fill in Calendar Table for DDAT1 */
cal_t1->Num_Elems = 2;
cal_t1->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t1->Element->Type = 0x0001;
memset(cal_t1->Element->reserved, '\0', 10);
(cal_t1->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t1->Element+1)->Type = 0;
memset((cal_t1->Element+1)->reserved, '\0', 10);
/* Fill in DDAT2 */
```

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = __YYYYMMDDHHMMSS;
ddat_t2->Hour_Zone = 24;
ddat_t2->Min_Zone = 60;
ddat_t2->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT2 */

era_t2->Num_Elems = 1;
era_t2->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t2->Element[0].Era_Name[0] = 'A';
era_t2->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT2 */

cal_t2->Num_Elems = 2;
cal_t2->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t2->Element->Type = 0x0001;
memset(cal_t2->Element->reserved, '\0', 10);
(cal_t2->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t2->Element+1)->Type = 0;
memset((cal_t2->Element+1)->reserved, '\0', 10);

cvtts(result_ts, source_ts, inst_t1);

result_ts[26] = '\0';
printf("The converted timestamp is %s
\n", result_ts);
}

Output
The converted timestamp is 1996-12-17-11.05.30.000000
Copy Bytes Left-Adjusted (CPYBLA)

Format
#include <QSYSINC/MIH/CPYBLA>

void cpynbl ( _SPCPRTR receiver, _SPCPTRCN source, int size);

Description
The cpynbl function copies size bytes of source to receiver. No padding is done.

Parameters
receiver (output)
  Pointer to the receiver.

source (input)
  Pointer to the source.

size (input)
  The number of bytes to copy. This must be between 0 and 16,773,104.

Notes on Usage
• This is a compatibility function to provide the same semantics as the CPYBLA MI instruction.
• Pointers within the source are not preserved by the operation.
• Behavior is undefined if copying takes place between overlapping locations.
• The builtin _CPYBYTES also provides access to an equivalent MI instruction. This builtin can only be used to copy non-pointer data.

Exceptions
If any of the input parameters are not valid, an MCH3601 or MCH0601 may be signalled.

Other exceptions possible from this function are:
  MCH0602 - Boundary alignment
  MCH0801 - Parameter Reference Violation
  MCH3402 - Object Destroyed
  MCH3602 - Pointer Type Invalid
  MCH6801 - Object Domain Violation
Example
This example illustrates the use of the cpybla function.

```c
#include <QSINC/MIH/CPYBLA>
#include <string.h>
#include <stdio.h>

int main(void) {
    char receiver[] = "XXXXXXXXXXXXXXX";
    char source[] = "Good Day";

    cpybla(receiver, source, strlen(source));
    printf("The receiver string is %s\n", receiver);
}
```

Output
The receiver string is Good DayXXXXXXXX
Copy Bytes Left-Adjusted with Pad (CPYBLAP)

Format
#include <QSYINC/MIH/CPYBLAP>

void cpyblap (_SPC PTR receiver, 
    int rcvr_size, 
    _SPC PTRCN source, 
    int src_size, 
    char pad);

Description
The cpyblap function copies bytes from source to receiver. If source is shorter than receiver, then src_size bytes from source are copied to receiver, and each excess byte of receiver is assigned the value of pad. If source is longer than receiver, then the leftmost bytes of source (equal in length to rcvr_src) are copied to receiver.

Parameters
receiver (input/output)
    Pointer to the receiver.
rcvr_size (input)
    The length of the receiver.
source (input)
    Pointer to the source.
src_size (input)
    The length of the source. The length must be between 0 and 16 773 104.
pad (input)
    The pad character.

Notes on Usage
- This is a compatibility function to provide the same semantics as the CPYBLAP MI instruction.
- If rcvr_size is ≤0, no action is taken.
- If src_size is ≤0, all bytes of receiver are set to the pad character.
- Behavior is undefined if copying takes place between overlapping locations.
- Pointers within source are not preserved.

Exceptions
If any of the input parameters are not valid, an MCH3601 or MCH0601 may be signalled.

Other exceptions possible from this function are:
    MCH0602 - Boundary alignment
    MCH0801 - Parameter Reference Violation
    MCH3402 - Object Destroyed
    MCH3602 - Pointer Type Invalid
    MCH6801 - Object Domain Violation
Example
This example illustrates the use of the cpyblap function.

```c
#include <QSYSINC/MIH/CPYBLAP>
#include <string.h>
#include <stdio.h>

int main(void) {
    char receiver[] = "XXXXXXXXXXXXXX";
    char source[] = "Good Day";

    cpyblap(receiver, strlen(receiver), source, strlen(source), '.');
    printf("The receiver string is %s\n", receiver);
}
```

Output
The receiver string is Good Day.......
Copy Hex Digit Numeric to Numeric (CPYHEXNN)

Format
#include <QSYSINC/MIH/CPYHEXNN>

char cpyhexnn (char *dest, char source);

Description
The cpyhexnn function copies the numeric hexadecimal digit value (rightmost 4 bits) of the byte referred to by source to the numeric hexadecimal digit value (rightmost 4 bits) of the leftmost byte referred to by dest. The function returns a copy of the modified byte.

Parameters
dest (input/output)
    Pointer to the location whose leftmost byte will be modified.

table
  input
    The source byte.

Notes on Usage
- This is a compatibility function to provide the same semantics as the CPYHEXNN MI instruction.

Exceptions
If dest does not point to a valid storage location, an MCH3601 will be signalled.
Example
This example illustrates the use of the cpyhexnn function.

```c
#include <QSYSINC/MIH/CPYHEXNN>
#include <stdio.h>

int main(void) {
    char    hex_dest[6] = "2A5F6"; /* in hex: 0xf2 0xc1 0xf5 0xc6 0xf6*/
    char    hex_source = 0xf3, h;
    h = cpyhexnn(hex_dest, hex_source);
    printf("The return byte is \%x\n", h);
    printf("The leftmost byte of destination string is \%x\n", *hex_dest);
}
```

Output
The return byte is f3
The leftmost byte of destination string is f3
Copy Hex Digit Numeric to Zone (CPYHEXNZ)

Format
#include <QSYSINC/MIH/CPYHEXNZ>

char cpyhexnz (char *dest, char source);

Description
The cpyhexnz function copies the numeric hexadecimal digit value (rightmost 4 bits) of the byte referred to by source to the zone hexadecimal digit value (leftmost 4 bits) of the leftmost byte referred to by dest. The function returns a copy of the modified byte.

Parameters
dest (input/output)
   Pointer to the location whose leftmost byte will be modified.

source (input)
   The source byte.

Notes on Usage
- This is a compatibility function to provide the same semantics as the CPYHEXNZ MI instruction.

Exceptions
If dest does not point to a valid storage location, an MCH3601 will be signalled.
**Example**

This example illustrates the use of the cpyhexnz function.

```c
#include <QSYSINC/MIH/CPYHEXNZ>
#include <stdio.h>

int main(void) {

    char hex_dest[6] = "2A5F6"; /* in hex: 0xf2 0xc1 0xf5 0xc6 0xf6 */
    char hex_source = 0xf3, h;

    h = cpyhexnz(hex_dest, hex_source);

    printf("The return byte is %x\n", h);
    printf("The leftmost byte of destination string is %x\n", *hex_dest);
}
```

**Output**

The return byte is 32
The leftmost byte of destination string is 32
Copy Hex Digit Zone to Numeric (CPYHEXZN)

**Format**
```c
#include <QSYSINC/MIH/CPYHEXZN>

char cpyhexzn (char *dest, char source);
```

**Description**
The `cpyhexzn` function copies the zone hexadecimal digit value (leftmost 4 bits) of the byte referred to by `source` to the numeric hexadecimal digit value (rightmost 4 bits) of the leftmost byte referred to by `dest`. The function returns a copy of the modified byte.

**Parameters**
- **dest (input/output)**
  - Pointer to the location whose leftmost byte will be modified.
- **source (input)**
  - The source byte.

**Notes on Usage**
- This is a compatibility function to provide the same semantics as the CPYHEXZN MI instruction.

**Exceptions**
- If `dest` does not point to a valid storage location, an MCH3601 will be signalled.
Example
This example illustrates the use of the cpyhexzn function.

```c
#include <QSYSINC/MIH/CPYHEXZN>
#include <stdio.h>

int main(void) {

    char hex_dest[6] = "2A5F6"; /* in hex: 0xf2 0xc1 0xf5 0xc6 0xf6 */
    char hex_source = 0xf3, h;
    h = cpyhexzn(hex_dest, hex_source);

    printf("The return byte is %x\n", h);
    printf("The leftmost byte of destination string is %x\n", *hex_dest);
}
```

Output
The return byte is ff
The leftmost byte of destination string is ff
Copy Hex Digit Zone to Zone (CPYHEXZZ)

**Format**

```
#include <CPYHEXZZ>

char cpyhexzz (char *dest, char source);
```

**Description**

The cpyhexzz function copies the zone hexadecimal digit value (leftmost 4 bits) of the byte referred to by `source` to the zone hexadecimal digit value (leftmost 4 bits) of the leftmost byte referred to by `dest`. The function returns a copy of the modified byte.

**Parameters**

- **dest (input/output)**
  Pointer to the location whose leftmost byte will be modified.
- **source (input)**
  The source byte.

**Notes on Usage**

- This is a compatibility function to provide the same semantics as the CPYHEXZZ MI instruction.

**Exceptions**

If `dest` does not point to a valid storage location, an MCH3601 will be signalled.
Example
This example illustrates the use of the cpyhexzz function.

```c
#include <QSINC/MIH/CPYHEXZZ>
#include <stdio.h>

int main(void) {
    char hex_dest[6] = "2A5F6"; /* in hex: 0xf2 0xc1 0xf5 0xc6 0xf6 */
    char hex_source = 0xf3, h;
    h = cpyhexzz(hex_dest, hex_source);

    printf("The return byte is %x\n", h);
    printf("The leftmost byte of destination string is %x\n", *hex_dest);
}
```

Output
The return byte is f2
The leftmost byte of destination string is f2
Copy Numeric Value (CPYNV)

Format
#include <QSYSINC/MIH/CPYNV>

_SPCPTR cpynv (_NUM_Descr_T rcvr_descr,
   _SPCSTR receiver,
   _NUM_Descr_T src_descr,
   _SPCPTCN source);

Description
The cpynv function copies the numeric value of source to receiver, with the appropriate conversions. The function returns a pointer to the receiver.

Parameters
rcvr_descr (input)
The receiver descriptor. Must be prepared using the NUM_DESCR macro.

receiver (input/output)
Pointer to the receiver location where the result will be placed.

src_descr (input)
The source descriptor. Must be prepared using the NUM_DESCR macro.

source (input)
Pointer to the source value.

Notes on Usage
• If necessary, source is converted to the same type as receiver before being copied.
• The builtin function _LBCPYNV (Late-bound Copy Numeric Value) provides access to the MI instruction.
• The builtin function _CPYNV also provides access to the equivalent MI instruction. When using the _CPYNV instruction, you must specify literal constants for the receiver and source descriptor values. _CPYNV is considerably faster than _LBCPYNV for the following conversions:
  zoned to packed, float, signed integer, or unsigned integer
  packed to zoned
  signed integer to zoned
  unsigned integer to zoned
Example
This example illustrates the use of the cpynv function.

```c
#include <QSINC/MIH/CPNV>
#include <stdio.h>

void convert_double_to_zone(unsigned char *, int, int, double);

int main(void) {
    char    zoned_output[15];
    double  float_input = 36584.89;

    convert_double_to_zone(zoned_output,
                          15,
                          5,
                          float_input);

    printf("The zoned decimal number ZND(15,5) is %.15s\n", zoned_output);
}

void convert_double_to_zone(unsigned char *znd, int digits, int fraction, double db1)
{
    cpynv(NUM_DESCR( _T_ZONED, digits, fraction), znd,
          NUM_DESCR( _T_FLOAT, 8, 0), &db1);
}

Output
The zoned decimal number ZND(15,5) is 000003658489000
Decompress Data (DCPDATA)

Format
#include <QSYSINC/MIH/DCPDATA>

int dcpdata (_SPCPTTR result,
             int result_length,
             _SPCPTRCN source);

Description
The dcpdata function decompresses user data. If the decompressed result is
longer than the result area (as specified by result_length), the decompression is
stopped and only result_length bytes are stored.

The dcpdata function returns the number of bytes in the decompressed result. This
value is always set to the full length of the result, which may be larger than
result_length.

Parameters
result (input/output)
  Pointer to the result area to receive the decompressed data.

result_length (input)
  The result area length.

source (input)
  The data to be decompressed.

Notes on Usage
- The length of the source data is not supplied because this
  length is contained within the compressed data.

- The builtin function _DCPDATA also provides access to the MI instruction.
Example
This example illustrates the use of the dcpdata function.

```c
#include <QSYSINC/MIH/DCPDATA>
#include <QSYSINC/MIH/CPRDATA>
#include <string.h>
#include <stdio.h>

#define SIZE 50

int main(void) {
    char cpr_data[SIZE];
    char dpr_data[SIZE] = "Good Day";
    int clength;
    int dlength;

    clength = cprdata(cpr_data, SIZE, dpr_data, strlen(dpr_data));
    printf("The compressed length = %d\n", clength);

    dlength = dcpdata(dpr_data, SIZE, cpr_data);
    printf("The decompressed length = %d\n", dlength);

    if (memcmp(dpr_data, "Good Day", strlen(dpr_data)) != 0)
        printf("Error in decompression\n");
    else
        printf("Decompression successful\n");
}
```

Output
The compressed length = 26
The decompressed length = 8
Decompression successful
Decrement Date (DECD)

Format
#include <QSYSINC/MIH/DECD>

void decd (_SPCPTR result_date,
   _SPCPTCN source_date,
   _SPCPTCN duration,
   _INST_Template_T3 *inst_t);

Description
The decd function decrements the date by the date duration. The date specified by
source_date is decreased by the date duration specified by duration. The resulting
date from the operation is placed in result_date.

Parameters
result_date (input/output)
   Pointer to the location to receive the result.

source_date (input)
   Pointer to the source date.

duration (input)
   Pointer to the packed decimal duration.

inst_t (input)
   Pointer to the instruction template which defines the data definitional attributes
   for result_date, source_date, and duration.

Notes on Usage
- The DDATs for result_date and source_date must be identical.
- The builtin function _DECD also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the decd function.

```c
#include <QSYSINC/MIH/MIDTMM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <stdint.h>
#include <decimal.h>

#define GREGORIAN_TIMELINE_START 1721424
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {

        _INST_Template_T3 inst_t3;
        _DDAT_T         *ddat_t1, *ddat_t2;
        _Era_Table_T    *era_t1;
        _Calendar_Table_T *cal_t1;

        int             DDAT_Length, Calendar_Offset;
        int             DDAT_Size, Template_Size;
        int             DDAT_Offset1, DDAT_Offset2;

        char            result_d[11];
        decimal(8,0)    duration = 00120310D;

        DDAT_Length = sizeof(_DDAT_T) +
                        2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
                        sizeof(_Era_Table_T) - 1;

        Calendar_Offset = offsetof(_DDAT_T, Tables) +
                        sizeof(_Era_Table_T);

        DDAT_Size = 2*DDAT_Length +
                    2*(sizeof(int)) + /* DDAT_Offset */
                          10 + /* reserved4 */
                        sizeof(short) + /* Num_DDATs */
                          sizeof(int); /* DDAT_Size */

        Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
                        sizeof(int); /* ddat offset */
```
DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
sizeof(int); /* DDAT offset */

DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t3 = (_INST_Template_T3 *)malloc(Template_Size);

/* Fill in Instruction Template */

memset(inst_t3, '\0', sizeof(_INST_Template_T3));

inst_t3->Template_Size = Template_Size;
inst_t3->DDAT_1 = 1;
inst_t3->DDAT_2 = 1;
inst_t3->DDAT_3 = 2;
inst_t3->Length_1 = 10;
inst_t3->Length_2 = 10;
inst_t3->F_Digits = 0;
inst_t3->T_Digits = 8;
inst_t3->Indicators = _OUT_ADJUST | _IN_ADJUST;
inst_t3->DDAT_Size = DDAT_Size;
inst_t3->Num_DDATS = 2;
inst_t3->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t3->DDAT_Offset + 1) = DDAT_Offset2;

/* Set table pointers within instruction template */

ddat_t1 = (_DDAT_T *)&(inst_t3->DDAT_Offset + 2);

era_t1 = (_Era_Table_T *)&(ddat_t1->Tables);
cal_t1 = (_Calendar_Table_T *)&(char *)ddat_t1 + Calendar_Offset);

ddat_t2 = (_DDAT_T *)&(char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT1 */

ddat_t1->DDAT_Length = DDAT_Length;

ddat_t1->Format_Code = _ISO_DATE;

ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT1 */

era_t1->Num_Elems = 1;

era_t1->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;

era_t1->Element[0].Era_Name[0] = 'A';

era_t1->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT1 */

cal_t1->Num_Elems = 2;

cal_t1->Element->Effect_Date = GREGORIAN_TIMELINE_START;

cal_t1->Element->Type = 0x0001;

memset(cal_t1->Element->reserved, '\0', 10);

(cal_t1->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;

(cal_t1->Element+1)->Type = 0;

memset((cal_t1->Element+1)->reserved, '\0', 10);
/* Fill in DDAT 2 */

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = _DATE_DUR;
ddat_t2->Calendar_Offset = Calendar_Offset;

decd(result_d, source_d, &duration, inst_t3);

result_d[10] = '\0';
printf("The resulting date is %s\n", result_d);
}

Output
The resulting date is 1980-10-20
Decrement Time (DECT)

Format
#include <QSYSINC/MIH/DECT>

void dect (_SPCPT result_time,
          _SPCPTCN source_time,
          _SPCPTCN duration,
          _INST_Template_T3 *inst_t);

Description
The dect function decrements the time by the time duration. The time specified by
source_time is decreased by the time duration specified by duration. The resulting
time from the operation is placed in result_time.

Parameters
result_time (input/output)
    Pointer to the location to receive the result.

source_time (input)
    Pointer to the source time.

duration (input)
    Pointer to the packed decimal duration.

inst_t (input)
    Pointer to the instruction template which defines the data definitional attributes
    for result_time, source_time, and duration.

Notes on Usage
- The DDATs for result_time and source_time must be identical.
- The built-in function _DECT also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the dect function.

```c
#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <decimal.h>

int main(void) {
    _INST_Template_T3 *inst_t3;
    _DDAT_T *ddat_t1, *ddat_t2;

    int DDAT_Length, Calendar_Offset;
    int DDAT_Size, Template_Size;
    int DDAT_Offset1, DDAT_Offset2;

    char result_d[9];
    char source_d[9] = "11.05.30";
    decimal(6, 0) duration = 0000205D;

    DDAT_Length = sizeof(_DDAT_T) +
        2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
        sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
        sizeof(_Era_Table_T);

    DDAT_Size = 2*DDAT_Length +
        2*sizeof(int)) + /* DDAT_Offset */
        10 + /* reserved4 */
        sizeof(short) + /* Num_DDATs */
        sizeof(int); /* DDAT_Size */

    Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
        sizeof(int); /* ddat offset */

    DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) -
        offsetof(_INST_Template_T3, DDAT_Size)) +
        sizeof(int); /* DDAT2 offset */

    DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) -
        offsetof(_INST_Template_T3, DDAT_Size)) +
        DDAT_Length + sizeof(int); /* DDAT2 offset */
```
```c
inst_t3 = (_INST_Template_T3 *)malloc(Template_Size);
/* Fill in Instruction Template */
memset(inst_t3, '0', sizeof(_INST_Template_T3));

inst_t3->Template_Size = Template_Size;
inst_t3->DDAT_1 = 1;
inst_t3->DDAT_2 = 1;
inst_t3->DDAT_3 = 2;
inst_t3->Length_1 = 8;
inst_t3->Length_2 = 8;
inst_t3->F_Digits = 0;
inst_t3->T_Digits = 6;
instit_t3->Indicators = _NO_ADJUST_OR_TOLERATE;
instit_t3->DDAT_Size = DDAT_Size;
instit_t3->Num_DDATS = 2;
instit_t3->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t3->DDAT_Offset + 1)= DDAT_Offset2;

ddat_t1 = (_DDAT_T *)(&inst_t3->DDAT_Offset + 2);
ddat_t2 = (_DDAT_T*)((char *)ddat_t1 + DDAT_Length);
/* Fill in DDAT1 */

ddat_t1->DDAT_Length = DDAT_Length;
ddat_t1->Format_Code = _ISO_TIME;
ddat_t1->Hour_Zone = 24;
ddat_t1->Min_Zone = 60;
ddat_t1->Calendar_Offset = Calendar_Offset;
/* Fill in DDAT2 */

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = _TIME_DUR;
ddat_t2->Hour_Zone = 0;
ddat_t2->Min_Zone = 0;
ddat_t2->Calendar_Offset = Calendar_Offset;
dect(result_d, source_d, &duration, inst_t3);

result_d[8] = '\0';
printf("The resulting time is %s\n", result_d);
}

Output
The resulting time is 11.03.25
```
Decrement Timestamp (DECTS)

**Format**
#include <QSYSINC/MIH/MIDTM>

```c
void dects (_SPCPTR result_timestamp,
            _SPCPTRCN source_timestamp,
            _SPCPTRCN duration,
            _INST_Template_T3 *inst_t);
```

**Description**
The dects function decrements the timestamp by the timestamp duration. The timestamp specified by `source_timestamp` is decreased by the date, time, or timestamp duration specified by `duration`. The resulting timestamp from the operation is placed in `result_timestamp`.

**Parameters**
result_timestamp (input/output)
  Pointer to the location to receive the result.

source_timestamp (input)
  Pointer to the source timestamp.

duration (input)
  Pointer to the packed decimal duration.

inst_t (input)
  Pointer to the instruction template which defines the data definitional attributes for `result_timestamp`, `source_timestamp`, and `duration`.

**Notes on Usage**
- The DDATs for `result_timestamp` and `source_timestamp` must be identical.
- The built-in function `_DECTS` also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the dects function.

```c
#include <QSYSINC/MIH/MIDTTM>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <decimal.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {

    _INST_Template_T3 *inst_t3;
    _DDAT_T *ddat_t1, *ddat_t2;
    _Era_Table_T *era_t1;
    _Calendar_Table_T *cal_t1;

    int DDAT_Length, Calendar_Offset;
    int DDAT_Size, Template_Size;
    int DDAT_Offset1, DDAT_Offset2;

    char result_d[27];
    char source_d[27] = "1993-01-17-11.05.30.000000";
    decimal(20,6) duration = 00000203012005.000000D;

    DDAT_Length = sizeof(_DDAT_T) +
        2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
        sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
        sizeof(_Era_Table_T);

    DDAT_Size = 2*DDAT_Length +
        2*sizeof(int) + /* DDAT_Offset */
        10 + /* reserved4 */
        sizeof(short) + /* Num_DDATS */
        sizeof(int); /* DDAT_Size */
```
Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
sizeof(int); /* ddat offset */

DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
sizeof(int); /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t3 = (_INST_Template_T3 *)malloc(Template_Size);

/* Fill in Instruction Template */
memset(inst_t3, '\0', sizeof(_INST_Template_T3));

inst_t3->Template_Size = Template_Size;
inst_t3->DDAT_1 = 1;
inst_t3->DDAT_2 = 1;
inst_t3->DDAT_3 = 2;
inst_t3->Length_1 = 26;
inst_t3->Length_2 = 26;
inst_t3->F_Digits = 6;
inst_t3->T_Digits = 20;
inst_t3->Indicators = _OUT_ADJUST | _IN_ADJUST;
inst_t3->DDAT_Size = DDAT_Size;
inst_t3->Num_DDats = 2;
inst_t3->DDAT_Offset[0] = DDAT_offset1;
*(inst_t3->DDAT_Offset + 1) = DDAT_offset2;
/* Set table pointers within instruction template */
ddat_t1 = (_DDAT_T *)&(inst_t3->DDAT_Offset + 2);
era_t1 = (_Era_Table_T *)&ddat_t1->Tables);
cal_t1 = (_Calendar_Table_T *)((char *)ddat_t1 + Calendar_Offset);
ddat_t2 = (_DDAT_T *)(char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT1 */

ddat_t1->DDAT_Length = DDAT_Length;
ddat_t1->Format_Code = _SAA_TIMESTAMP;
ddat_t1->Hour_Zone = 24;
ddat_t1->Min_Zone = 60;
ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT1 */

era_t1->Num_Elems = 1;
era_t1->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t1->Element[0].Era_Name[0] = 'A';
era_t1->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT1 */
cal_t1->Num_Elems = 2;
cal_t1->Element->Effect_Date = GREGORIAN_TIMELINE_START;
dects

cal_t1->Element->Type = 0x0001;
memset(cal_t1->Element->reserved, '\0', 10);
(cal_t1->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t1->Element+1)->Type = 0;
memset((cal_t1->Element+1)->reserved, '\0', 10);

/* Fill in DDAT 2 */

ddat_t2->DDAT_Length = DDAT_Length;
(ddat_t2->Format_Code = _TIMESTAMP_DUR);
(ddat_t2->Hour_Zone = 0);
(ddat_t2->Min_Zone = 0);
(ddat_t2->Calendar_Offset = Calendar_Offset);

dects(result_d, source_d, &duration, inst_t3);

result_d[26] = '\0';
printf("The resulting timestamp is %s\n", result_d);
}

Output
The resulting timestamp is 1996-12-14-09.45.25.000000
Dequeue (DEQ)

Format
#include <QSYSINC/MIH/DEQ>

void deq (_DEQ_Msg_Prefix_T *msg_prefix,
        _SPCPTR message,
        _SYSPTR queue);

Description
The deq function retrieves a queue message based on the queue type (FIFO, LIFO, or keyed) specified during the queue's creation. If a message is not found that satisfies the dequeue selection criteria, the process waits until a message arrives to satisfy the dequeue or until the dequeue wait time-out expires.

Parameters
msg_prefix (input/output)
    Pointer to the message prefix template. This contains the criteria for message selection.

message (output)
    Pointer to the location to receive the message text.

queue (input)
    Pointer to the queue from which the message is to be dequeued.

Notes on Usage
- The builtin function _DEQWAIT also provides access to the MI instruction.

Example
This example illustrates the use of the deq function.

/*-----------------------------------------------*/
/*
/* Example Group: User Queues <QSYSINC/MIH/DEQ> */
/*
/* Function:   deq (dequeue) */
/*
/* Description: This example will dequeue the next message from */
/* the user queue and output it to the screen. If */
/* there are no entries are on the queue, this */
/* program will wait indefinitely until one appears. */
/*
*/
/*-----------------------------------------------*/

#include <QSYSINC/MIH/DEQ>
#include <QSYSINC/MIH/SLVSP>
#include <stdio.h>

    /* Template used for setting */
    /* options for the dequeue. */
    _DEQ_Msg_Prefix_T message_prefix; /* Also used as the receiver */
    /* for the dequeued message. */
```c
_deq

_SysPtr queue_ptr; /* Pointer to the user queue. */
char message_text[75]; /* Buffer for dequeued message. */

int main(void) {

    /* Get a pointer to the *USRQ. */
    queue_ptr = rslvsp(_usrq, "MYUSRQ", "MYLIB", _AUTH_ALL);

    message_prefix.Wait_Forever = 1; /* Wait indefinitely until an */
    /* entry appears on the queue. */

    /*************************************************************************
    /* Dequeue the next message from the user queue and store it in the */
    /* 'message_text' buffer. If the queue is empty, this program will */
    /* be suspended until an entry appears (by some other job running a */
    /* a program that enqueues an entry on the queue). */
    /*************************************************************************/
    deq(&message_prefix, message_text, queue_ptr);

    /* Display the received message */
    /* to the screen. */

    printf("Each entry/message is %d bytes long.\n",
            message_prefix.Msg_Size);

    printf("The following was received from the User Queue. \n\n");
    printf("> %.75s \n\n", message_text);
}

Output
Each entry/message is 75 bytes long.
The following was received from the User Queue.
> This is the first message being entered.
```
Dequeue Message with Indicator (DEQI)

**Format**

```c
#include "QSYINC/MIH/DEQ"

int deqi (_DEQ_Msg_Prefix_I *msg_prefix,
   _SPCPTR message,
   _SYSPTR queue);
```

**Description**

The deqi function retrieves a queue message based on the queue type (FIFO, LIFO, or keyed) specified during the queue's creation. If a message satisfying the dequeue selection criteria is not available, the function will return immediately with the return code set as indicated below. The process is not placed in a wait state.

**Parameters**

- **msg_prefix (input/output)**
  
  Pointer to the message prefix template. This contains the criteria for message selection.

- **message (output)**
  
  Pointer to the location to receive the message text.

- **queue (input)**
  
  Pointer to the queue from which the message is to be dequeued.

**Notes on Usage**

- The built-in function _DEQI also provides access to the MI instruction.

**Return Code**

Values returned by the function deqi are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Message dequeued</td>
</tr>
<tr>
<td>0</td>
<td>No message meeting dequeue criteria on the queue</td>
</tr>
</tbody>
</table>

**Example**

This example illustrates the use of the deqi function.

```c
/*---------------------------------------------------------------*/
/*                                                               */
/* Example Group: User Queues <QSYINC/MIH/DEQ>                   */
/*                                                               */
/* Function: deqi (dequeue and set indicator)                   */
/*                                                               */
/* Description: This example dequeues the next message from the */
/*              user queue and displays it to the screen. If      */
/*              there are no entries, this program continues and  */
/*              the return code (indicator) from 'deqi' will      */
/*              indicate whether or not there was anything on the */
/*              queue to be dequeued.                            */
/*                                                               */
/*---------------------------------------------------------------*/
```
#include <QSYSINC/MIH/DEQ>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>
#include <stdlib.h>

/* Stores information returned */
_DEQ_Msg_Prefix_T message_prefix; /* by the 'deqi' function. */
_SYSPTR queue_ptr; /* Pointer to the user queue. */
char message_text[ 75 ]; /* Buffer for dequeued message. */
int return_code; /* "indicator" returned by deqi */

int main(void) {
    /* Get a pointer to the *USRQ. */
    queue_ptr = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /*-----------------------------------------------*/
    /* The 'deqi' function unlike the 'deq' function will always return */
    /* right away whether or not an entry was found on the user queue. */
    /*-----------------------------------------------*/

    return_code = deqi( &message_prefix, message_text, queue_ptr );

    if ( return_code == 0 ) {
        printf("The queue was empty. 'deqi' returned 0.\n");
        exit( 1 );
    }
    /* Inform the user of the message */
    /* that was received from the queue*/

    printf("'deqi' returned %d indicating a successful dequeue\n",
          return_code );

    printf("The following was received from the User Queue. \n");
    printf("> %.75s \n", message_text );
}

Output
'deqi' returned 1 indicating a successful dequeue
The following was received from the User Queue.
> This is the second message being entered.
Edit (EDIT)

**Format**
```
#include <QSYSINC/MIH/EDIT>

void edit (_SPCPT receiver,
         unsigned int rcvr_length,
         _SPCPTRCN source,
         _NUM_Descr_T src_descr,
         _SPCPT mask,
         unsigned int mask_length);
```

**Description**
The edit function transforms the value of a numeric `source` from its internal form to character form suitable for display at a source/sink device and places the result in `receiver`. There are a number of general editing functions (such as unconditional insertion of a mask character string), that can be performed during the transformation.

**Parameters**
- **receiver (input/output)**
  Pointer to the location to receive the transformed result.
- **rcvr_length (input)**
  The length of the receiver area. This value must be between 1 and 256.
- **source (input)**
  The source string address.
- **src_descr (input)**
  The source descriptor. Must be prepared using the NUM_DESCR macro.
- **mask (input)**
  The mask string address.
- **mask_length (input)**
  The mask string length. This value must be between 1 and 256.

**Notes on Usage**
- The builtin function _LBEDIT (Late-bound Edit) also provides access to the MI instruction.
Example
This example illustrates the use of the edit function.

```c
#include <QSYSINC/MIH/EDIT>
#include <string.h>
#include <stdio.h>

#define SIZE 11

int main(void) {

    char receiver[SIZE+1];
    char edit_mask[18] =
          {0x34, 0xb1, 0x40, 0x4e, 0x34, 0x60, 0x34,
           0xb2, 0xb2, 0xb2, 0xb2, 0xb2, 0xb2, 0xb2,
           0xb2, 0xb2, 0xb2, 0xb2};

    unsigned mask_len = sizeof(edit_mask);
    int source = 12652;

    edit(receiver, SIZE, &source, NUM_DESCR(_T_UNSIGNED, 4, 0),
         edit_mask, mask_len);
    receiver[SIZE] = '\0';
    printf("The transformed result is \%s\n", receiver);
}
```

Output
The transformed result is 12652
Edit Packed Decimal (EDIT_PACKED)

Format
#include <QSYSINC/MIH/EDIT>

#include <QSYSINC/MIH/EDIT>

void edit_packed (_SPCPT receiver,  unsigned int rcvr_length,  _SPCPTRCN source,  unsigned int src_length,  _SPCPT mask,  unsigned int mask_length);

Description
The edit_packed function transforms the value of a packed decimal source from its internal form to character form suitable for display at a source/sink device and places the result in receiver. There are a number of general editing functions (such as unconditional insertion of a mask character string), that can be performed during the transformation.

Parameters
receiver (input/output)
   Pointer to the location to receive the transformed result.

rcvr_length (input)
   The length of the receiver area. This value must be between 1 and 256.

source (input)
   Pointer to the packed decimal source.

src_length (input)
   The source length. This value must be between 1 and 31.

mask (input)
   The mask string address.

mask_length (input)
   The mask length. This value must be between 1 and 256.

Notes on Usage
- The builtin function _EDITPD also provides access to the MI instruction.
- This function is faster than edit for a packed decimal source.
Example
This example illustrates the use of the edit_packed decimal function.

```c
#include <QSYSINC/MIH/EDIT>
#include <decimal.h>
#include <string.h>
#include <stdio.h>

#define SIZE 11

int main(void) {
    char receiver[SIZE+1];
    char edit_mask[18] =
        {0x34, 0xb1, 0x40, 0x4e, 0x34, 0x60, 0x34,
         0xb2, 0xb2, 0xb2, 0xb2, 0xb2, 0xb2, 0xb2,
         0xb2, 0xb2, 0xb2, 0xb2};

    unsigned mask_len = sizeof(edit_mask);
    _Decimal(10,0) source = 12652D;

    edit_packed(receiver, SIZE, &source, digitof(source), edit_mask, mask_len);
    receiver[SIZE] = '\0';
    printf("The transformed result is >>%s<<\n", receiver);
}

Output
The transformed result is >> +12652<<
Enqueue (ENQ)

**Format**
#include <QSYSINC/MIH/ENQ>

void enq (_SYSPTR queue,
        _ENQ_Msg_Prefix_T *msg_prefix,
        _SPCPTTR message);

**Description**
The enq function enqueues a message according to the queue type attribute specified during the queue's creation.

**Parameters**
**queue** (input)
Pointer to the queue to which the message is to be enqueued.

**msg_prefix** (input)
Pointer to the message prefix template. This contains information about the message being enqueued.

**message** (output)
Pointer to the message text to be enqueued.

**Notes on Usage**
- The builtin function _ENQ also provides access to the MI instruction.

**Example**
This example illustrates the use of the enq function.

```c
#include <QSYSINC/MIH/ENQ>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>

_SYSPTR queue_ptr;    /* Pointer to the user queue. */
_ENQ_Msg_Prefix_T message_prefix; /* Provides description of the */  /* message being enqueued. */
char message_text[75];    /* Stores the input received */  /* from the user which will */  /* then be enqueued. */
```
```c
int main(void) {
    queue_ptr = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );
    printf("Enter some text you wish to have put on the User Queue:\n");
    gets( message_text );

    message_prefix.Msg_Len = 75;
    enq( queue_ptr, &message_prefix, message_text );
    printf("Your message has been enqueued onto the user queue. \n");
}
```

**Output**
Enter some text you wish to have put on the User Queue:
This is the first message being entered.
Your message has been enqueued onto the user queue.
Ensure Object (ENSOBJ)

**Format**

```
#include <QSYSINC/MIH/ENSOBJ>

void ensobj (_SYSPTR object);
```

**Description**

The ensobj function protects the specified object from volatile storage loss.

The machine ensures that any changes made to the specified object are recorded on nonvolatile storage media. The access state of the object is not changed.

No action is taken for temporary objects since temporary objects are not preserved during machine failure.

**Parameters**

- **object (input)**
  - Pointer to the system object to be protected.

**Notes on Usage**

- The built-in function _ENSOBJ also provides access to the MI instruction.
Example
This example illustrates the use of the ensobj function.

```c
#include <QSYSINC/MIH/ENSOBJ>
#include <QSYSINC/MIH/RSLVSP>

_SYSPTR some_object; /* The pointer to the object */
/* that will be "forced" to */
/* auxillary storage (disk). */

int main(void) {
    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );
    ensobj( some_object ); /* Force the object to disk. */
}
```

Output
** no screen output **
Extract Exponent (EXTREXP)

Format
#include <QSYSINC/MIH/EXTREXP>

int extexp (double source);

Description
The extexp function extracts the exponent portion of a floating-point scalar and returns the value as a 4 byte integer.

Parameters
source (input)
An 8-byte floating-point number.

Return Values

<table>
<thead>
<tr>
<th>value of source</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>+/-INFINITY</td>
<td>32767</td>
</tr>
<tr>
<td>NaN</td>
<td>-32768</td>
</tr>
</tbody>
</table>
Example
This example illustrates the use of the extrexp function.

```
/****************************************************************************
/* Example Group: Computation and Branching <QSYSINC/MIH/EXPREXP> */
/* */
/* Function:  extrexp  (Extract Exponent) */
/* */
/* Description: This example uses 'extrexp' to return the */
/* exponent portion of the floating-point value. */
/* e.g. significand * 2 ** exponent */
/* extrexp(6734.219) = 1.6440964 * 2 ** 12 */
/****************************************************************************

#include <QSYSINC/MIH/EXPREXP>
#include <stdio.h>

int main(void) {
    double source = 6734.219;
    int exponent;

    exponent = extrexp(source);
    printf("The exponent = %d\n", exponent);
}
```

Output
The exponent = 12
Find Independent Index Entry (FNDINXEN)

**Format**
```
#include <QSYSINC/MIH/FNDINXEN>

_SPCPTR fndinxen (_SPCPTR receiver,
_SYSPTR index_obj,
_IIX_opt_list_T *option_list,
_SPCPTR search_arg);
```

**Description**
The `fndinxen` function searches the independent index according to the search criteria specified in the option list and the search argument. The desired entry or entries are returned in the receiver. A pointer to the receiver area is also returned by the function.

**Parameters**
- **receiver (input/output)**
  Pointer to the buffer to receive the returned entry or entries.
- **index_obj (input)**
  Pointer to the independent index object to be searched.
- **option_list (input/output)**
  Pointer to the option list template. This template contains additional information on the search.
- **search_arg (input)**
  Pointer to the search argument(s).

**Notes on Usage**
- The built-in function `_FNDINXEN` also provides access to the MI instruction.
Example
This example illustrates the use of the fndinxen function.

```c
/* Example Group: User Indexes <QSYSINC/MIH/FNDINDEX> */
/* */
/* Function:  fndinxen (Find Index Entry) */
/* */
/* Description: Search for an entry using a particular customer */
/* number (stored as an integer). If an entry is */
/* found in the user index, then display the text */
/* of the entry to the screen. */
/* */
/*****************************************************************************/

#include <QSYSINC/MIH/FNDINDEX>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>

typedef struct Customer_Entry { /* The format/layout of each */
    int customer_number; /* entry in the user index. */
    char last_name[ 50 ]; /* */
    char first_name[ 40 ]; /* */
    char phone_number[9]; /* Using phone format: 999-9999 */
} Customer_Entry;

_IIX_Opt_List_T find_options; /* options for the "find". */
/* Also used as the receiver */
/* for any returned information.*/

_SYSPTR index_ptr; /* Pointer to the user index. */

Customer_Entry found_customer; /* entry that may be returned */
/* from the 'fndinxen'. */

int which_customer = 55555; /* This customer number. */

int main(void) {
    /* Get a pointer to the *USRIDX */
    index_ptr = rslvsp( _Usridx, "MYUSRIDX", "MYLIB", _AUTH_ALL );

    find_options.Rule = _FIND_EQUALS; /* Find a matching customer */
    /* number. */

    find_options.Arg_Length = sizeof(int); /* The "search key" is the */
    /* 4-byte integer customer */
    /* number. */
```c
find_options.Occ_Count = 1; /* In case more than one entry */
/* exists with this customer */
/* number, only return the */
/* first entry found. */

/*-----------------------------------------------*/
/* Search for the entry that matches the customer number stored in */
/* 'which_customer' and store the entry in 'found_customer'. The */
/* find options specified in the 'find_options' template are used to */
/* control the search. After the call to 'findinxen', this template */
/* also stores the number of entries found. If one entry is found, */
/* its contents will be displayed to the screen. */
/*-----------------------------------------------*/

findinxen( &found_customer, index_ptr, &find_options, &which_customer );

if ( find_options.Ret_Count == 0 )
    printf("Customer number %d was not found.\n", which_customer);
else
    if (find_options.Ret_Count == 1) {
        printf("An entry was found for customer number: %d \n", which_customer);

        printf("Customer Name is: %s %s \n", found_customer.first_name, found_customer.last_name);

        printf("Phone Number is: %s \n",found_customer.phone_number);
    }
}

Output
An entry was found for customer number: 55555
Customer Name is: Mary Wilson
Phone Number is: 777-7777
```
Find Relative Invocation Number (FNDRINVN)

Format

```c
#include <QSYSINC/MIH/FNDRINVN>

int fndrinvn (_INV_Template_T *inv_t
    int srch_option,
    _SPCPTRCN srch_arg,
    char compare);
```

Description

The fndrinvn function searches a specified range of invocations until an invocation is found which satisfies the search criteria. If no invocation in the specified range satisfies the search criteria, a value of zero is returned by the function. Otherwise, the identity of the first invocation to satisfy the search criteria is returned by the function. The return value is specified relative to the starting invocation. A positive number indicates a displacement in the direction of newer invocations, while a negative indicates a displacement in the direction of older invocations.

Parameters

- **inv_t (input)**
  - Pointer to the search range template or NULL. The search range template identifies the starting invocation and the range of the search. Specifying NULL will default to a range starting with the current invocation and proceeding through all existing older invocations.

- **srch_option (input)**
  - Specifies the invocation attribute to be examined.

- **srch_arg (input)**
  - Pointer to the search argument. This is a value between one and 16 bytes depending on the search option specified.

- **compare (input)**
  - Specifies the compare operation.
    - 0X00 = Match. The function will identify the first invocation which matches the specified search criteria.
    - 0X01 = Mismatch. The function will identify the first invocation which does not match the specified search criteria.

Notes on Usage

- The starting invocation is skipped in the search and no exception is signalled if the search criterion is not satisfied.
- The built-in functions _FNDRINVN1 and _FNDRINVN2 also provide access to the MI instruction.
Example
This example illustrates the use of the fndrnvn function.

```c
/*-----------------------------------------------*/
/* Example Group: Machine Observation <QSYSINC/MIH/FNDRINVN> */
/* Function: fndrnvn (Find Relative Invocation Number) */
/* Description: This example uses 'fndrnvn' to search the call stack for an invocation with the specified invocation mark. Please Note: the 'fndrnvn' function itself has an entry on the call stack and this must be taken into account when calculating relative invocation offsets. This is not necessary if either one of _FNDRINVN1 or _FNDRINVN2 builtins are used. */
/*-----------------------------------------------*/

#include <QSYSINC/MIH/FNDRINVN>
#include <QSYSINC/MIH/MATINVAT>
#include <string.h>
#include <stdio.h>

void func(void);

int      inv_mark, rel_off;
_INV_Template_T inv_t;

int main(void) {
    func();
}

void func(void) {

    memset(&inv_t, 0, sizeof(_INV_Template_T));

    inv_t.Inv_Offset = -2; /* main()'s invocation */
    /* Get invocation mark for main() */
    matinvat(&inv_mark, &inv_t, _MTVA_INV_MARK, sizeof(inv_mark));

    /* Use the invocation mark from matinvat as the search criteria */
    /* for fndrnvn. The relative invocation number returned should */
    /* -2 indicating success at the invocation of main(). */
    rel_off = fndrnvn(NULL, _FNDR_INV_MARK, &inv_mark, _FNDR_MATCH);
    printf("The search is stopped at %d invocations from the current\n", rel_off);
}
Output
The search is stopped at -2 invocations from the current
Increment Date (INCD)

**Format**

```c
#include <QSYSINC/MIH/MIDTTM>

void incd (_SPCPTR result_date,
            _SPCPTCN source_date,
            _SPCPTCN duration,
            _INST_Template_T3 *inst_t);
```

**Description**

The `incd` function increments the date by the date duration. The date specified by `source_date` is increased by the date duration specified by `duration`. The resulting date from the operation is placed in `result_date`.

**Parameters**

- **result_date (input/output)**
  
  Pointer to the location to receive the result.

- **source_date (input)**
  
  Pointer to the source date.

- **duration (input)**
  
  Pointer to the packed decimal duration.

- **inst_t (input)**
  
  Pointer to the instruction template which defines the data definitional attributes for `result_date`, `source_date`, and `duration`.

**Notes on Usage**

- The builtin function `INCD` also provides access to the MI instruction.
- A macro version is available.
- The DDATs for `result_date` and `source_date` must be identical.
Example
This example illustrates the use of the incd function.

```c
/* Example Group: Date/Time/Timestamp <QSYSINC/MIH/MIDTTM> */
/* Function: incd (Increment Date) */
/* Description: This example uses 'incd' to increment the date */
/* by the date duration. */
*/

#include <QSYSINC/MIH/MIDTTM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <decimal.h>

/* Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

/* Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {
    _INST_Template_T3 *inst_t3;
    _DDAT_T *ddat_t1, *ddat_t2;
    _Era_Table_T *era_t1;
    _Calendar_Table_T *cal_t1;

    int DDAT_Length, Calendar_Offset;
    int DDAT_Size, Template_Size;
    int DDAT_Offset1, DDAT_Offset2;

    char result_d[11];
    decimal(8,0) duration = 01610801D;

    DDAT_Length = sizeof(_DDAT_T) +
       2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
       sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
       sizeof(_Era_Table_T);

    DDAT_Size = 2*DDAT_Length +
       2*(sizeof(int)) + /* DDAT_Offset */
       10 + /* reserved4 */
       sizeof(short) + /* Num_DDATs */
       sizeof(int); /* DDAT_Size */

    Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
       sizeof(int); /* ddat offset */
```
DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) - 
                offsetof(_INST_Template_T3, DDAT_Size)) + 
                sizeof(int); /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) - 
                offsetof(_INST_Template_T3, DDAT_Size)) + 
                DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t3 = (_INST_Template_T3 *)&malloc(Template_Size);

/* Fill in Instruction Template */
memset(inst_t3, '\0', sizeof(_INST_Template_T3));

inst_t3-&gt;Template_Size = Template_Size;
inst_t3-&gt;DDAT_1 = 1;
inst_t3-&gt;DDAT_2 = 1;
inst_t3-&gt;DDAT_3 = 2;
inst_t3-&gt;Length_1 = 10;
inst_t3-&gt;Length_2 = 10;
inst_t3-&gt;F_Digits = 0;
inst_t3-&gt;T_Digits = 8;
inst_t3-&gt;Indicators = _OUT_ADJUST | _IN_ADJUST;
inst_t3-&gt;DDAT_Size = DDAT_Size;
inst_t3-&gt;Num_DDATs = 2;
*(inst_t3-&gt;DDAT_Offset[0]) = DDAT_Offset1;
*(inst_t3-&gt;DDAT_Offset + 1) = DDAT_Offset2;

/* Set table pointers within instruction template */

ddat_t1 = (_DDAT_T *)&(inst_t3-&gt;DDAT_Offset + 2);
era_t1 = (_Era_Table_T *)&(ddat_t1-&gt;Tables);
cal_t1 = (_Calendar_Table_T *)((char *)ddat_t1 + Calendar_Offset);

ddat_t2 = (_DDAT_T *)((char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT1 */

ddat_t1-&gt;DDAT_Length = DDAT_Length;
ddat_t1-&gt;Format_Code = _ISO_DATE;
ddat_t1-&gt;Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT1 */

era_t1-&gt;Num_Elems = 1;
era_t1-&gt;Element[0].Origin_Date = GREGORIAN_TIMELINE_START;
era_t1-&gt;Element[0].Era_Name[0] = 'A';
era_t1-&gt;Element[0].Era_Name[1] = '0';

/* Fill in Calendar Table for DDAT1 */

cal_t1-&gt;Num_Elems = 2;
cal_t1-&gt;Element-&gt;Effect_Date = GREGORIAN_TIMELINE_START;
cal_t1-&gt;Element-&gt;Type = 0x0001;
memset(cal_t1-&gt;Element-&gt;reserved, '\0', 10);
(cal_t1-&gt;Element+1-&gt;Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t1-&gt;Element+1-&gt;Type = 0;
memset((cal_t1-&gt;Element+1-&gt;reserved, '\0', 10);
/* Fill in DDAT 2 */

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = _DATE_DUR;
ddat_t2->Calendar_Offset = Calendar_Offset;

incd(result_d, source_d, &duration, inst_t3);

result_d[10] = '\0';
printf("The resulting date is %s\n", result_d);
}

**Output**
The resulting date is 2154-10-01
**Increment Time (INCT)**

**Format**
```c
#include <QSYSINC/MIH/MIDTTM>

void inct (_SPCPTTR result_time,
         _SPCPTRCN source_time,
         _SPCPTRCN duration,
         _INST_Template_T3 *inst_t);
```

**Description**
The inct function increments the time by the time duration. The time specified by `source_time` is increased by the time duration specified by `duration`. The resulting time from the operation is placed in `result_time`.

**Parameters**
- **result_time (input/output)**
  Pointer to the location to receive the result.
- **source_time (input)**
  Pointer to the source time.
- **duration (input)**
  Pointer to the packed decimal duration.
- **inst_t (input)**
  Pointer to the instruction template which defines the data definitional attributes for `result_date`, `source_date` and `duration`.

**Notes on Usage**
- The DDATs for `result_time` and `source_time` must be identical.
- The built-in function _INCT also provides access to the MI instruction.
- A macro version is available.
Example
This example illustrates the use of the *inct* function.

```c
/*---------------------------------------------*/
/* Example Group: Date/Time/Timestamp <QSYSINC/MIH/MIDTMM> */
/* Function: *inct* (Increment Time) */
/* Description: This example uses 'inct' to increment the time */
/* by the time duration. */
/*---------------------------------------------*/

#include <QSYSINC/MIH/MIDTMM>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <decimal.h>

int main(void) {

    _INST_Template_T3    *inst_t3;
    _DDAT_T              *ddat_t1, *ddat_t2;

    int                  DDAT_Length, Calendar_Offset;
    int                  DDAT_Size, Template_Size;
    int                  DDAT_Offset1, DDAT_Offset2;

    char                 result_d[9];
    char                 source_d[9] = "11.05.30";
    decimal(6,0)         duration = 000205D;

    DDAT_Length = sizeof(_DDAT_T) +
                  2*sizeof(_Calendar_Table_T) - sizeof(short) +
                  sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
                      sizeof(_Era_Table_T);

    DDAT_Size = 2*DDAT_Length +
                2*sizeof(int)) + /* DDAT_Offset */
                10 + /* reserved4 */
                sizeof(short) + /* Num_DDATs */
                sizeof(int); /* DDAT_Size */

    Template_Size = 2*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
                    sizeof(int); /* ddat offset */

    DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) -
                    offsetof(_INST_Template_T3, DDAT_Size)) +
                    sizeof(int); /* DDAT2 offset */

    DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) -
                    offsetof(_INST_Template_T3, DDAT_Size)) +
                    DDAT_Length + sizeof(int); /* DDAT2 offset */
```

Machine Interface Library Functions 121
inct

inst_t3  = (_INST_Template_T3 *)malloc(Template_Size);

/* Fill in Instruction Template */
memset(inst_t3, '\0', sizeof(_INST_Template_T3));

inst_t3->Template_Size    = Template_Size;
inst_t3->DDAT_1        = 1;
inst_t3->DDAT_2        = 1;
inst_t3->DDAT_3        = 2;
inst_t3->Length_1     = 8;
inst_t3->Length_2     = 8;
inst_t3->F_Digits     = 0;
inst_t3->T_Digits     = 6;
inst_t3->Indicators  = _NO_ADJUST_OR_TOLERATE;
inst_t3->DDAT_Size  = DDAT_Size;
inst_t3->Num_DDATS = 2;
inst_t3->DDAT_Offset[0]  = DDAT_Offset1;
*(inst_t3->DDAT_Offset + 1)= DDAT_Offset2;

ddat_t1  = (_DDAT_T *)&inst_t3->DDAT_Offset + 2);
ddat_t2  = (_DDAT_T *)((char *)ddat_t1 + DDAT_Length);

/* Fill in DDAT1 */

ddat_t1->DDAT_Length    = DDAT_Length;
ddat_t1->Format_Code   = _ISO_TIME;
ddat_t1->Hour_Zone     = 24;
ddat_t1->Min_Zone      = 60;
ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in DDAT2 */

ddat_t2->DDAT_Length    = DDAT_Length;
ddat_t2->Format_Code   = _TIME_DUR;
ddat_t2->Hour_Zone     = 0;
ddat_t2->Min_Zone      = 0;
ddat_t2->Calendar_Offset = Calendar_Offset;

inct(result_d, source_d, &duration, inst_t3);

result_d[8] = '\0';
printf("The resulting time is %s\n", result_d);
}

Output
The resulting time is 11.07.35
Increment Timestamp (INCTS)

Format
#include <QSYSINC/MIH/MIDTTM>

void incts ( _SPCPTR result_timestamp,
             _SPCPTCN source_timestamp,
             _SPCPTCN duration,
             _INST_Template_T3 *inst_t);

Description
The incts function increments the date by the timestamp. The timestamp specified
by source_timestamp is increased by the date, time, or timestamp duration speci-
fied by duration. The resulting timestamp from the operation is placed in
result_timestamp.

Parameters
result_timestamp (input/output)
   Pointer to the location to receive the result.

source_timestamp (input)
   Pointer to the source timestamp.

duration (input)
   Pointer to the packed decimal duration.

inst_t (input)
   Pointer to the instruction template which defines the data definitional attributes
   for result_timestamp, source_timestamp, and duration.

Notes on Usage
- The DDATs for result_timestamp and source_timestamp must be identical.
- The builtin function _INCTS also provides access to the MI instruction.
- A macro version is available.
Example

This example illustrates the use of the incts function.

```c
#include <QSYSINC/MIH/MIDTMM>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <decimal.h>

// Internal format for start of Gregorian timeline: January 1, 0001 */
#define GREGORIAN_TIMELINE_START 1721424

// Internal format for end of Gregorian timeline: January 1, 10000 */
#define GREGORIAN_TIMELINE_END 5373485

int main(void) {

    _INST_Template_T3 *inst_t3;
    _DDAT_T *ddat_t1, *ddat_t2;
    _Era_Table_T *era_t1;
    _Calendar_Table_T *cal_t1;

    int DDAT_Length, Calendar_Offset;
    int DDAT_Size, Template_Size;
    int DDAT_Offset1, DDAT_Offset2;

    char result_d[27];
    char source_d[27] = "1993-01-17-11.05.30.000000";
    decimal(20,6) duration = 00000203012005.000000D;

    DDAT_Length = sizeof(_DDAT_T) +
        2*(sizeof(_Calendar_Table_T)) - sizeof(short) +
        sizeof(_Era_Table_T) - 1;

    Calendar_Offset = offsetof(_DDAT_T, Tables) +
        sizeof(_Era_Table_T);

    DDAT_Size = 2*DDAT_Length +
        2*(sizeof(int)) + /* DADT_Offset */
        10 + /* reserved4 */
        sizeof(short) + /* Num_DDATs */
        sizeof(int); /* DADT_Size */
```

Machine Interface Library Functions

Template_Size = 2\*DDAT_Length + offsetof(_INST_Template_T3, DDAT) +
sizeof(int); /* ddat offset */

DDAT_Offset1 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
sizeof(int); /* DDAT2 offset */

DDAT_Offset2 = (offsetof(_INST_Template_T3, DDAT) -
offsetof(_INST_Template_T3, DDAT_Size)) +
DDAT_Length + sizeof(int); /* DDAT2 offset */

inst_t3 = (_INST_Template_T3 *)malloc(Template_Size);

/* Fill in Instruction Template */

memset(inst_t3, '\0', sizeof(_INST_Template_T3));

inst_t3->Template_Size = Template_Size;
inst_t3->DDAT_1 = 1;
inst_t3->DDAT_2 = 1;
inst_t3->DDAT_3 = 2;
inst_t3->Length_1 = 26;
inst_t3->Length_2 = 26;
inst_t3->F_Digits = 6;
inst_t3->T_Digits = 20;
inst_t3->Indicators = _OUT_ADJUST | _IN_ADJUST;
inst_t3->DDAT_Size = DDAT_Size;
inst_t3->Num_DDATS = 2;
inst_t3->DDAT_Offset[0] = DDAT_Offset1;
*(inst_t3->DDAT_Offset + 1) = DDAT_Offset2;

/* Set table pointers within instruction template */

ddat_t1 = (_DDAT_T *)&(inst_t3->DDAT_Offset + 2);
era_t1 = (_Era_Table_T *)&(ddat_t1->Tables);
cal_t1 = (_Calendar_Table_T *)&(char *)ddat_t1 + Calendar_Offset);

ddat_t2 = (_DDAT_T *)&(char *)ddat_t1 + DDAT_Length;

/* Fill in DDAT1 */

ddat_t1->DDAT-Length = DDAT-Length;

ddat_t1->Format_Code = _SAA_TIMESTAMP;

ddat_t1->Hour_Zone = 24;

ddat_t1->Min_Zone = 60;

ddat_t1->Calendar_Offset = Calendar_Offset;

/* Fill in Era Table for DDAT1 */

era_t1->Num_Elems = 1;

era_t1->Element[0].Origin_Date = GREGORIAN_TIMELINE_START;

era_t1->Element[0].Era_Name[0] = 'A';

era_t1->Element[0].Era_Name[1] = 'D';

/* Fill in Calendar Table for DDAT1 */

cal_t1->Num_Elems = 2;

cal_t1->Element->Effect_Date = GREGORIAN_TIMELINE_START;
cal_t1->Element->Type = 0x0001;
memset(cal_t1->Element->reserved, '\0', 10);
(cal_t1->Element+1)->Effect_Date = GREGORIAN_TIMELINE_END;
(cal_t1->Element+1)->Type = 0;
memset((cal_t1->Element+1)->reserved, '\0', 10);

/* Fill in DDAT 2 */

ddat_t2->DDAT_Length = DDAT_Length;
ddat_t2->Format_Code = TIMESTAMP_DUR;
ddat_t2->Hour_Zone = 0;
ddat_t2->Min_Zone = 0;
ddat_t2->Calendar_Offset = Calendar_Offset;

incts(result_d, source_d, &duration, inst_t3);

result_d[26] = '\0';
printf("The resulting timestamp is %s\n", result_d);
}

Output
The resulting timestamp is 1996-12-20-12.25.35.000000
Insert Independent Index Entry (INSINXEN)

**Format**
```c
#include <QSYSINC/MIH/INSINXEN>

void insinxen (_SYSPTR new_entry,
               _SPCPTR index_obj,
               _IIX_Opt_List_T *option_list);
```

**Description**
The insinxen function inserts one or more new entries into an independent index, according to the criteria specified. Each entry is inserted into the index based on the binary value of the argument.

**Parameters**
- `index_obj` (input)
  Pointer to the independent index object to receive the new entry or entries.
- `new_entry` (input)
  Pointer to the new entry or entries to be inserted.
- `option_list` (input)
  Pointer to the option list template. This template contains additional information regarding the insertion of the new entry or entries.

**Notes on Usage**
- The builtin function _INSINXEN also provides access to the MI instruction.
- When a NULL is passed as the second operand of insinxen, error message MCH5601 or MCH3601 will be generated.
Example
This example illustrates the use of the insinxen function.

```c
#include <QSYSINC/MIH/INSINXEN>
#include <QSYSINC/MIH/RLVSP>
#include <stdio.h>

typedef struct Customer_Entry {
    int customer_number;        /* entry in the user index. */
    char last_name[ 50 ];       /* Using phone format: 999-9999 */
    char first_name[ 40 ];
    char phone_number[9];
} Customer_Entry;

_IIX_Opt_List_T insert_options; /* Template used for setting */
/* options for the "insert". */
/* Also used as the receiver */
/* for any returned data. */

_SYSPTR index_ptr;        /* Pointer to the user index. */

/* Declare and initialize a */
/* few example entries. */

Customer_Entry customer1 = {9999999, "Smith", "John", "555-5555" },
customer2 = { 55555, "Wilson", "Mary", "777-7777" },
customer3 = { 1, "Chan", "Julie", "123-4567" };

int main(void) {

    index_ptr = rlvsp( _Usridx, "MYUSRIDX", "MYLIB", _AUTH_ALL );

    insert_options.Rule        /* For Keyed User Indexes must */
                     /* specify "insert and replace" */
    = _INSERT_REPLACE;        /* instead of just "insert". */
```
insinxen( index_ptr, &customer1, &insert_options );
if (insert_options.Ret_Count != 1) /* If 1 entry was not inserted, */ /* then it either failed or more*/
    printf("First Insert Failed\n"); /* than 1 entry was inserted */ /* (but we only asked for 1 to */ /* be inserted). */

insinxen( index_ptr, &customer2, &insert_options );
if (insert_options.Ret_Count != 1)
    printf("Second Insert Failed\n");

insinxen( index_ptr, &customer3, &insert_options );
if (insert_options.Ret_Count != 1)
    printf("Third Insert Failed\n");
}

** Output **

** No output **
Lock Object (LOCK)

Format
#include <QSYSINC/MIH/LOCK>

void lock (_SYSPTR object,
           MI_Time wait_time,
           char lock_state);

Description
The lock function requests that a single lock for the system object be allocated to the issuing process.

Parameters
object (input)
  Pointer to the system object to be locked.

wait_time (input)
  The wait time requested. This specifies the maximum amount of time that a process competes for the requested lock. Use the mitime function to form this value.

lock_state (input)
  The lock state requested.

Notes on Usage
- The lock request type will be set for the user by default, depending on what is specified as the wait time. The following rule will be used: If wait_time = 0, _LOCK_NORMAL will be used as the request type. If wait_time > 0, _LOCK_SYNCH will be used as the request type.
- The lock entry will always be marked as active by the lock function. So, for the function version it is not necessary to OR the lock state with _LOCK_ENTRY_ACTIVE.
- The builtin function _LOCK and macro Lock also provide access to the MI instruction.
Example

This example illustrates the use of the lock function.

```c
#include <QSYSINC/MIH/LOCK>
#include <QSYSINC/MIH/RSLVSP>
#include <miptrnam.h>
#include <stdio.h>
#include <stdlib.h>

_SYSPTR some_object;  /* The object being locked. */
_MI_Time timeout;  /* Amount of time to wait for */
                    /* the lock to be placed. */
int hours  =  0,  /* Time components used to */
int minutes =  0,  /* create an _MI_Time value */
int seconds = 12,  /* that can be passed to the */
int hundredths = 0;  /* 'lock' function. */

int main(void) {
    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Format an 'mitime' value to represent the timeout to be used by */
    /* the 'lock' function when trying to obtain an Exclusive-No-Read */
    /* (LENR) lock on the object. If the lock request cannot be satisfied */
    /* in the specified amount of time, then an exception will be raised. */
    /* The CL DSPJOB command is called to display all of the object locks */
    /* owned by the job/process that runs this program. */
    /******************************************************************/
```
lock

mitime( &timeout, hours, minutes, seconds, hundredths );

lock( some_object, timeout, _LENR_LOCK );

system( "DSPJOB OPTION(*JOBLCK)" );

{

Output

Display Job Locks
System: TORA4YL

Job: OMXNA3S4 User: VISCA Number: 009616

Job status: ACTIVE

Type options, press Enter.
5=Display job member locks

Opt Object Library Type Lock Status Member
MAIN QSYS *MENU *SRNUP HELD
    *SRNUP HELD
MYUSRQ MYLIB *USRQ *EXCL HELD
OMXNA3S4 QSYS *DEVD *EXCLRC HELD
    *EXCLRC HELD
    *EXCLRC HELD
QADM QSYS *LIB *SHRRD HELD

F3=Exit  F5=Refresh  F10=Display job record locks  F12=Cancel

More...
Lock Space Location (LOCKSL)

Format
#include <QSYSINC/MIH/LOCKSL>

void locksl (_SPCPTR location, 
            char lock_state);

Description
The locksl function grants the space location to the issuing process according to
the lock request. For this function, if the requested lock cannot be immediately
granted, the process will enter a synchronous wait for the lock for a period up to
the interval specified by the process default time-out value.

Parameters
location (input)
  Pointer to the space location to be locked.

lock_state (input)
  The lock type request.

Notes on Usage
• The builtin functions _LOCKSL1 and _LOCKSL2 also provide access to the MI
  instruction.
• The _LOCKSL2 builtin supports multiple space location lock requests.
**Example**

This example illustrates the use of the locksl function.

```c
#include <QSINC/MIH/LOCKSL>
#include <QSINC/MIH/RSLVSP>
#include <QSINC/MIH/SETSPFP>
#include <QSINC/MIH/MATOBJLK>
#include <stdio.h>

_SYSPTR some_object; /* The object from which some */
             /* byte (space) location will */
             /* be locked. */

_SPCPTR some_byte; /* The pointer to the location */
                  /* that is being locked. */

_MOBJL_Template_T allocated_locks; /* The receiver template used */
                                    /* by 'matobjlk' */

int main(void) {

    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Set a space pointer from */
    /* the system pointer. */
    some_byte = setsppfp( some_object );

    /* Request the Exclusive-No-Read (LENR) lock on the space location. */
    /* A default timeout called the "Process Default Timeout" is used as */
    /* the timeout value. If the lock request is not satisfied in that */
    /* time, an exception is raised. */
    locksl( some_byte, _LENR_LOCK );
```

---

Example Group: Locks <QSINC/MIH/LOCKSL> */

Function: locksl (Lock Space Location) */

Description: Place an Exclusive-No-Read (LENR) on a particular */

byte (space location) of some object (a user */

queue in this example). This type of lock is */

"symbolic" in that it does not prevent the space */

location from being referenced or updated. */

---

#include <QSINC/MIH/LOCKSL>
#include <QSINC/MIH/RSLVSP>
#include <QSINC/MIH/SETSPFP>
#include <QSINC/MIH/MATOBJLK>
#include <stdio.h>

_SYSPTR some_object; /* The object from which some */
             /* byte (space) location will */
             /* be locked. */

_SPCPTR some_byte; /* The pointer to the location */
                  /* that is being locked. */

_MOBJL_Template_T allocated_locks; /* The receiver template used */
                                    /* by 'matobjlk' */

int main(void) {

    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Set a space pointer from */
    /* the system pointer. */
    some_byte = setsppfp( some_object );

    /* Request the Exclusive-No-Read (LENR) lock on the space location. */
    /* A default timeout called the "Process Default Timeout" is used as */
    /* the timeout value. If the lock request is not satisfied in that */
    /* time, an exception is raised. */
    locksl( some_byte, _LENR_LOCK );
```
/* In order to verify that the LENR lock was placed on the space */
/* location (byte), we can look at the bit pattern returned by */
/* 'matobjlk' in the 'Lock_Alloc' field of the template. The fifth */
/* bit (bit 4) of this field represents the Lock-Exclusive-No-Read */
/* (LENR) lock. By using the bitwise AND (&) operator, we can */
/* determine if the bit is set by ANDing it with the provided bit */
/* mask, _LENR_LOCK (which is defined in <milock.h> as hex 08 - bit */
/* pattern 0000 1000). */
/* */

allocated_locks.Template_Size = sizeof(_MOBJL_Template_T);
matobjlk( &allocated_locks, some_byte );

if ( allocated_locks.Lock_Alloc & _LENR_LOCK ) {
    printf("There is an Exclusive-No-Read lock on the space \n");
    printf("location with address: %p \n", some_byte );
} else
    printf("Error. The LENR lock request was not satisfied. \n");
}

Output
There is an Exclusive-No-Read lock on the space
location with address: SPP:0401MYLIB :0a02MYUSRQ :0:0:e8
Lock Space Location with Time-Out (LOCKSL2)

Format
#include <QSYSINC/MIH/LOCKSL2>

void locksl2 (_SPCPT location, 
  _MI_Time wait_time, 
  char lock_state);

Description
The locksl2 function grants the space location to the issuing process according to the lock request.

Parameters
location (input)
  Pointer to the space location to be locked.

wait_time (input)
  Wait_time value. Specifies the amount of time that the process competes for the requested lock.

lock_state (input)
  The lock type request.

Notes on Usage
- The lock request type will be set for the user by default, depending on what is specified as the wait time. The following rule will be used: If wait_time = 0, _LOCK_NORMAL will be used as the request type. If wait_time > 0, _LOCK_SYNCH will be used as the request type.
- The builtin functions _LOCKSL1 and _LOCKSL2 also provide access to the MI instruction.
- The _LOCKSL2 builtin supports multiple space location lock requests.
Example
This example illustrates the use of the locks12 function.

```c
#include <QSYSINC/MIH/LOCKSL>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/MIH/SETCPPFP>
#include <QSYSINC/MIH/MATOBJLK>
#include <stdio.h>

_MOBJL_T allocated_locks; /* The receiver template used */
by 'matobjlk'.

_SYSPTR some_object; /* The object from which some */
/* byte (space) location will */
/* locked. */

_SPCPTR some_byte; /* The pointer to the location */
/* that is being locked. */

_MI_Time timeout; /* Amount of time to wait for */
/* the lock to be placed. */

int hours = 0, /* Time components used to */
minutes = 0, /* create an _MI_Time value */
seconds = 10, /* that can be passed to the */
hundredths = 0; /* 'lock' function. */

int main(void) {
    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Set a space pointer from */
    /* the system pointer. */
    some_byte = setcppfp( some_object );
```
locksl2

/*@-----------------------------------------------------------*/
/*@ Format an 'mtime' value to represent the timeout to be used by */
/*@ the 'lock' function when trying to obtain an Exclusive-Allow-Read */
/*@ (LEAR) lock on the object. If the lock request cannot be satisfied */
/*@ in the specified amount of time, then an exception will be raised. */
/*@ The CL DSPJOB command is called to display all of the object locks */
/*@ owned by the job/process that runs this program. */
/*@-----------------------------------------------------------*/

mitime( &timeout, hours, minutes, seconds, hundredths );

locksl2( some_byte, timeout, _LEAR_LOCK );

/*@-----------------------------------------------------------*/
/*@ In order to verify that the LEAR lock was placed on the space */
/*@ Location (byte), we can look at the bit pattern returned by */
/*@ 'matoblk' in the 'Lock_Alloc' field of the template. The fourth */
/*@ bit (bit 3) of this field represents the Lock-Exclusive-Allow-Read */
/*@ (LEAR) lock. By using the bitwise AND (&) operator, we can */
/*@ determine if the bit is set by ANDing it with the provided bit */
/*@ mask, _LEAR_LOCK (which is defined in <milock.h> as hex 10 - bit */
/*@ pattern 0001 0000). */
/*@-----------------------------------------------------------*/

allocated_locks.Template_Size = sizeof( _MOBJL_Template_T );

matoblk( &allocated_locks, some_byte );

if ( allocated_locks.Lock_Alloc & _LEAR_LOCK ) {
    printf("There is an Exclusive-Allow-Read lock on the space \n");
    printf("Location with address: %p \n\n", some_byte );
}
else
    printf("Error. The LEAR lock request was not satisfied. \n");

Output
There is an Exclusive-Allow-Read lock on the space
location with address: SPP:0401MYLIB :0a02MYUSRQ :0:0:eda
Materialize Activation Attribute (MATACTAT)

**Format**

```
#include <QSYSINC/MIH/MATACTAT>

void matactat (_MTACT_Template_T *receiver
               unsigned int act_mark,
               char attr_selection);
```

**Description**

The matactat function materializes the information for the program activation into the receiver template.

**Parameters**

- `receiver (input/output)`
  - Pointer to the receiver template.

- `act_mark (input)`
  - The activation mark of the activation whose attributes are to be materialized. A value of zero indicates a request for information about the activation of the invoking program.

- `attr_selection (input)`
  - Identifies the information to be materialized.

**Notes on Usage**

- The builtin function _MATACTAT also provides access to the MI instruction.
Example
This example illustrates the use of the matactat function.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <QSYSINC/MIH/MATACTAT>
#include <QSYSINC/MIH/MATINVAT>

#define MY_CALLER -3 /* -2 = _C_PEP (program entry procedure) */  
                 /* -1 = main()               */  
                 /*  0 = matinvat library function */

int main(void) {

    unsigned       act_mark;
    _INV_Template_T inv_t;
    _MTACT_Template_T mtact_t;

    memset(&inv_t, 0, sizeof(_INV_Template_T));

    inv_t.Inv_Offset = MY_CALLER; /* caller's invocation */

    /* Materialize the activation mark of our caller */
    matinvat(&act_mark, &inv_t, _MTVA_ACT_MARK, sizeof(act_mark));

    mtact_t.Template_Size = sizeof(_MTACT_Template_T);
    matactat(&mtact_t, act_mark, _MTACT_BASIC_ACT_ATTR);

    /* Program Models: OPM = 0x00, ILE = 0x01 */
    printf("The program model of our caller is 0x\n",
          mtact_t.Data.Basic_Attr.Program_Model);
}

Output
The program model of our caller is 1
Materialize Access Group Attributes (MATAGAT)

Format
#include <QSYSINC/MIH/MATAGAT>

void matagat (_MAGAT_Template_T *receiver,
  _SYSPTR access_group);

Description
The matagat function materializes the attributes of the access group and the identification of objects currently contained in the access group.

Parameters
receiver (input/output)
  Pointer to the receiver template.

access_group (input)
  Pointer to the access group whose attributes are to be materialized.

Notes on Usage
- The builtin function _MATAGAT also provides access to the MI instruction.
Example
This example illustrates the use of the matagat function.

```c
/*---------------------------------------------------------------*/
/* */
/* Example Group: Resource Management Data(RMD) <QSYSINC/MIH/MATAGAT> */
/* */
/* Function: matagat (Materialize Access Group Attributes) */
/* */
/* Description: This example uses 'matagat' to materialize all */
/* of the objects that are part of an Access Group. */
/* Since the Process Access Group (PAG) is a readily */
/* available Access Group (using 'matpratr'), it */
/* will be used as the Access Group of which the */
/* attributes will be materialized. */
/* */
/* Access Groups should not be confused with */
/* Activation Groups. */
/* */
/* Note: This example will only work on a system running */
/* with security level 30 or lower since it attempts */
/* to materialize the PAG. */
/* */
/*---------------------------------------------------------------*/

#include <QSYSINC/MIH/MATAGAT>
#include <QSYSINC/MIH/MATPRATR>
#include <stdio.h>
#include <stdlib.h>

_MPRT_Template_T process_attributes; /* Receiver template that will*/
/* contain a pointer to the */
/* Process Access Group (PAG).*/

*access_group_attributes; /* Pointer to the receiver */
/* template for 'matagat'. */

_SYSPTR access_group_ptr; /* Pointer to an access group */
/* (specifically the PAG in */
/* this example). */

_object_pointer; /* Pointer used to loop */
/* through each system pointer*/
/* returned by 'matagat'. */

int object_count, /* The number of objects in */
/* the access group. */

object, /* Loop counter for displaying*/
/* each object pointer in the */
/* Access Group. */

new_size; /* Number of bytes required */
/* to hold all information */
/* returned by 'matagat'. */

int main(void) {
/* ***************************************************************************/
/* We need to get a pointer to an access group, so 'matpratr' */
/* (Materialize Process Attributes) is used to get a pointer to */
/* the Process Access Group (PAG), which is just a special kind of */
/* access group. */
/* ***************************************************************************/

process_attributes.Ptr_Attr3.Template_Size = sizeof( _MPRT_PTR_T );
matpratr( &process_attributes, NULL, _MPRT_PAG );

access_group_ptr = process_attributes.Ptr_Attr3.Mptr_Ptr;

/* ***********************************************************************/
/* Call 'matagat' first just to find out how many objects there are */
/* in this access group so that we can evaluate the amount of space */
/* required to accommodate information on each of the objects that */
/* are part of this access group. The amount of storage required */
/* for the complete fixed-portion of the template will be the exact */
/* size of the template. */
/* ***********************************************************************/

access_group_attributes =
    (_MAGAT_Template_T *) malloc( sizeof( _MAGAT_Template_T ) );
access_group_attributes->Template_Size = sizeof( _MAGAT_Template_T );
matagat( access_group_attributes, access_group_ptr );

    /* output some of the attributes */

printf("Characteristics of this access group: \n\n");

printf("Object Name: %30s \n", access_group_attributes->Object_ID.Name );

printf("Type and Subtype: %10.4X \n", access_group_attributes->Object_ID.Type_Subtype );

printf("Access Group Size: %10d \n", access_group_attributes->Group_Size );

printf("Available space in the access group: %10d \n", access_group_attributes->Group_Space );

printf("Number of objects in the access group: %4d \n", access_group_attributes->Object_Count );

printf("\n\n");
/*----------------------------------------------*/
/* Now that we know exactly how many objects there are, we can */
/* determine the amount of space needed to receive all of the */
/* information 'matagat' can provide. The additional space required */
/* has to accommodate one system pointer for each object in the */
/* access group. */
/*----------------------------------------------*/

object_count = access_group_attributes->Object_Count;

new_size = sizeof( _MAGAT_Template_T ) +
  ( object_count * sizeof( _SYSPTR ) );

access_group_attributes = (_MAGAT_Template_T *)
realloc( access_group_attributes, new_size );

access_group_attributes->Template_Size = new_size;

matagat( access_group_attributes, access_group_ptr );

printf("The following are the pointers to the objects that make up\n");
printf("the Process Access Group (PAG) for this job. \n\n");

object_pointer = &(access_group_attributes->Object_Ptr);

for ( object=0; object < object_count; object++, object_pointer++ ) {
  printf("System pointer to object %3d is: %p \n", object+1,
     , object_pointer );
}

Output
Characteristics of this access group:
Object Name: PAG
Type and Subtype: 01EF
Access Group Size: 3342336
Available space in the access group: 1072128
Number of objects in the access group: 9
The following are the pointers to the objects that make up
the Process Access Group (PAG) for this job.
System pointer to object 1 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17bd
System pointer to object 2 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17be
System pointer to object 3 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17bf
System pointer to object 4 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c0
System pointer to object 5 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c1
System pointer to object 6 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c2
System pointer to object 7 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c3
System pointer to object 8 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c4
System pointer to object 9 is: SPP:0000 :laefQPDADEV0025VISCA 003997 :17c5
Materialize Activation Group Attributes (MATAGPAT)

**Format**
#include <QSYSINC/MIH/MATAGPAT>

void matagpat (_MGAP_Template_I *receiver,
               unsigned int act_grp_mark,
               char attr_selection);

**Description**
The matagpat function materializes the attributes for the specified activation group into the receiver template.

**Parameters**
**receiver (input/output)**  
Pointer to the receiver template.

**act_grp_mark (input)**
The activation group mark of the activation group whose attributes are to be materialized. An activation group mark of zero will materialize the attributes of the activation group associated with the current invocation.

**attr_selection (input)**
Specifies the information to be materialized.

**Notes on Usage**
- The built-in function _MATAGPAT also provides access to the MI instruction.
Example

This example illustrates the use of the matagpat function.

```c
#include <stdio.h>
#include <stdlib.h>
#include <QSYSINC/MIH/MATAGPAT>
#include <QSYSINC/MIH/MATPRAG>
#include <QSYSINC/MIH/MATPTR>

int main(void) {
    _MPTR_Template_T mpt;
    _MPRAG_Template_T *mprgp;
    int num_AGs;
    unsigned AG_mark;
    int mpragp_req_size;
    _MAGP_Template_T magp;
    char attr;

    /* Use matprag to obtain list of AG marks within the process */
    mprgp = (_MPRAG_Template_T *)malloc(sizeof(_MPRAG_Template_T));
    mprgp->Template_Size = sizeof(_MPRAG_Template_T);
    matpragp(mprgp);
    num_AGs = mprgp->Act_Grp_Count;
    mpragp_req_size = sizeof(_MPRAG_Template_T) + ((num_AGs - 1)*
                          sizeof(unsigned));
    mprgp = (_MPRAG_Template_T *)malloc(mpragp_req_size);
    mprgp->Template_Size = mpragp_req_size;
    matpragp(mprgp);
    AG_mark = mprgp->Act_Grp_List[0];

    /* Materialize the program pointer to the root program of the */
    /* first AG in the list. */
    magp.Template_Size = sizeof(_MAGP_Template_T);
    matagpat(&magp, AG_mark, _MAGP_BASIC_AG_ATTR);

    /* Materialize the program name */
    mpt.Obj_Ptr.Template_Size = sizeof(_OBJPTR_T);
    matptr(&mpt, magp.Data.Basic_Attr.Program);

    printf("The root program of AG %u is %.10s\n", AG_mark,
            mpt.Obj_Ptr.Object_ID.Name);
```
Output
The root program of AG 74499 is MATAGPAT
Materialize Allocated Object Locks (MATAOL)

Format
#include <QSYSINC/MI/MATAOL>

void mataol (_MAOL_Template_I *receiver,
        _ANYPTR pointer);

Description
The mataol function materializes the current allocated locks on a designated
system object or space location.

Parameters
receiver (input/output)
   Pointer to the receiver template where the materialized locks will be returned.

pointer (input)
   Pointer to the system object or space location whose allocated locks are to be
   materialized. The argument must be a system pointer or a space pointer.

Notes on Usage
• The builtin function _MATAOL also provides access to the MI instruction.

Example
This example illustrates the use of the mataol function.

/include <QSYSINC/MI/MATAOL>
#include <QSYSINC/MI/RSLVSP>
#include <stdio.h>
#include <stdlib.h>

    /* The receiver template for 'mataol'. */
    _MAOL_Template_I allocated_locks;

    /* The object being locked. */
    _SYSPTR some_object;

int main(void) {
    /* Get a pointer to the object. */
    some_object = rslvsp(_Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL);

/* Use the CL ALCOBJ command to place a lock on the user queue (this */
/* could also be done with the 'lock' function). 'mataol' is then */
/* used to materialize all of the object locks held for the queue. */
/* *************************************************************************/

system( "ALCOBJ OBJ((MYLIB/MYUSRQ *USRQ *EXCL)) ");

allocated_locks.Template_Size = sizeof( _MAOL_Template_T );

mataol( &allocated_locks, some_object );

/* *************************************************************************/
/* In order to verify that the Exclusive lock was placed on the */
/* object, we can look at the bit pattern returned by 'mataol'. */
/* The bit pattern in the first byte (0th element of the array) */
/* of 'Lock_Status' represents "Current Cumulative Locks" and */
/* the fifth bit (bit 4) of this field represents the */
/* Lock-Exclusive-No-Read (LENR) lock. By using the bitwise AND */
/* operator, we can determine if the bit is set by ANDing it with */
/* provided bit mask, _LENR_LOCK (which is defined in <milock.h> as */
/* hex 08 - bit pattern of 0000 1000). */
/* *************************************************************************/

if ( allocated_locks.Lock_Status[ 0 ] & _LENR_LOCK ) {
    printf("There is an Exclusive-No-Read lock on the object.\n");
} else
    printf("Error. An Exclusive-No-Read lock is not held on the object.\n");

Output
There is an Exclusive-No-Read lock on the object.
Materialize Invocation Attributes (MATINVAT)

**Format**
```c
#include <QSYSINC/MIH/MATINVAT>

void matinvat (_SPCPTR receiver,
               _INV_Template_T *inv_t,
               int attr_id,
               int rcvr_length);
```

**Description**
The `matinvat` function materializes the attributes of the invocation into the receiver template. The attributes materialized and the source invocation are specified through `attr_id` and `inv_t`, respectively.

**Parameters**

- **receiver (output)**
  Pointer to the location to receive the materialized attribute.

- **inv_t (input)**
  Pointer to the invocation identification template or NULL. Specifying NULL indicates that the current identification template invocation is to be used for the source and originating invocations.

- **attr_id (input)**
  The attribute identifier. Indicates which attribute of the source invocation is to be materialized.

- **rcvr_length (input)**
  The length of the receiver.

**Notes on Usage**
- May be used to materialize a single attribute only.
- The builtin functions _MATINVAT1 and _MATINVAT2 also provide access to the MI instruction. Use the builtin form to materialize more than 1 attribute.
Example
This example illustrates the use of the matinvat function.

```c
#include <QSYSINC/MIH/MATINVAT>
#include <QSYSINC/MIH/FNDRINVN>
#include <string.h>
#include <stdio.h>

char srch_arg = _OPM_PROGRAM, inv_type;
int rel_off;
_INV_Template_T inv_t;

int main(void) {
    /* Find relative offset of first OPM program invocation in stack */
    rel_off = fnendrinvn(NULL, _FNDR_RTN_TYPE, &srch_arg, _FNDR_MATCH);
    memset(&inv_t, 0, sizeof(_INV_Template_T));
    inv_t.Inv_Offset = rel_off; /* previous invocation */

    /* What is the invocation type of this OPM program? */
    matinvat(&inv_type, &inv_t, _MTVA_INV_TYPE, sizeof(inv_type));

    switch(inv_type) {
    case _CALLX:
        printf("The nearest OPM program is a CALLX\n");
        break;
    case _XCTL:
        printf("The nearest OPM program is a XCTL\n");
        break;
    case _EVENT_HDLR:
        printf("The nearest OPM program is an event handler\n");
        break;
    case _OPM_CALLX_HDLR:
        printf("The nearest OPM program is an external exception handler\n");
        break;
    case _PROCESS_PROB_STATE:
        printf("The nearest OPM program is an IP in problem state\n");
    ```
matinvat

break;
case _PROCESS_INIT_STATE:
    printf("The nearest OPM program is an IP in init state\n");
    break;
case _PROCESS_TERM_STATE:
    printf("The nearest OPM program is an IP in term state\n");
    break;
case _OPM_INV_EXIT:
    printf("The nearest OPM program is an invocation exit\n");
    break;
case _RETURN:
    printf("The nearest OPM program is a return/XCTL trap handler\n");
    break;
case _CALLPGM:
    printf("The nearest OPM program is a CALLPGM \n");
    break;
default:
    printf("invalid type\n");
    break;
}

Output
The nearest OPM program is a XCTL
Materialize Independent Index Entries (MATINXAT)

Format
```
#include <QSYSINC/MIH/MATINXAT>

void matinxat (_IIX_Template_T *receiver,
              _SYSPTR index_obj);
```

Description
The matinxat function materializes the creation attributes and current operational statistics of the independent index.

Parameters
receiver (input/output)
  Pointer to the receiver template

index_obj (input)
  Pointer to the independent index whose attributes are to be materialized.

Notes on Usage
- The builtin function _MATINXAT also provides access to the MI instruction.

Example
This example illustrates the use of the matinxat function.
```
#include <QSYSINC/MIH/MATINXAT>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>

_IIX_Template_T index_attributes; /*'matinxat'. */
_SYSPTR index_ptr; /* Pointer to the user index. */

int main(void) {
  /* Get a pointer to the *USRIDX */
  index_ptr = rslvsp(_Usridx, "MYUSRIDX", "MYLIB", _AUTH_ALL);
```
matinxat

/*---------------------------------------------------------------*/
/* Materialize the attributes of the user index into the receiver  */
/* template and output the number of entries inserted, removed and */
/* searched for, as well as the number currently in the user index. */
/*---------------------------------------------------------------*/

index_attributes.Template_Size = sizeof(_IIIX_Template_T);
matinxat( &index_attributes, index_ptr );

printf("Number of entries inserted: %d \n",
 index_attributes.Count_Insert );

printf("Number of entries removed: %d \n",
 index_attributes.Count_Remove );

printf("Number of entries currently in the index: %d \n",
 index_attributes.Count_Insert
 - index_attributes.Count_Remove );

printf("Number of find requests: %d \n",
 index_attributes.Count_Find );

}

Output
Number of entries inserted: 4
Number of entries removed: 1
Number of entries currently in the index: 3
Number of find requests: 1
Materialize Machine Attributes (MATMATR)

Format
#include <QSYSINC/MIH/MATMATR>

void matmatr (_MMTR_Template_T *receiver, short attr);

Description
The matmatr function makes available the unique values of machine attributes. Through the attribute selection value, up to 22 different machine attributes may be selected for materialization.

Parameters
receiver (input/output)
    Pointer to the materialization template.

attr (input)
    The attribute selection value. This argument identifies which group of machine attributes are to be materialized.

Notes on Usage
- The builtin function _MATMATR1 also provides access to the MI instruction.
Example
This example illustrates the use of the matmatr function.

```c
#include <QSYSINC/MIH/MATMATR>
#include <stdio.h>

_MMTR_Template_T machine_attributes;

int main(void) {
    machine_attributes.Options.Template_Size = 16;
    matmatr( &machine_attributes, _MMTR_SERIAL );

    printf("Serial Number of this AS/400 is: %.8s \n", machine_attributes.Options.Data.Serial);
}
```

Output
Serial Number of this AS/400 is: 1015013
Materialize Machine Data (MATMDATA)

**Format**

```c
#include <QSYSINC/MIH/MATMDATA>

void matmdata (_MDATA_Template_T *receiver, 
                short options);
```

**Description**
The matmdata function materializes the values of various machine data.

**Parameters**

- **receiver (output)**
  - Pointer to the machine data template.

- **options (input)**
  - The machine data options. This argument determines which machine data are materialized. The option must be a literal.

**Notes on Usage**
- The builtin function _MATMDATA also provides access to the MI instruction.
Example
This example illustrates the use of the matmdta function.

```
#include <QSYSINC/MIH/MATMDATA>
#include <stdio.h>
_MDATA_Template_T machine_data; /*'matmdta'.

int main(void) {

/* Materialize only the integrity flag bit setting and store the resulting machine data receiver template.

    matmdta( &machine_data, _MDATA_INTEGRITY_FLAG );

/* Output whether the flag is on or off.

    printf("The system Parameter Integrity Validation flag is %s \n", machine_data.Integ_Flag ? "On" : "Off");
}

Output
The system Parameter Integrity Validation flag is Off
Materialize Object Locks (MATOBJLK)

Format
#include <QSYSINC/MIH/MATOBJLK>

void matobjlk (_MOBJL_Template_T *receiver,
                _ANYPTR object);

Description
The matobjlk function materializes the current lock status of the designated system object or space location.

Parameters
receiver (input/output)
    Pointer to the receiver template where the materialized lock status will be returned.

object (input)
    Pointer to the system object or space location whose lock status is to be materialized. The argument must be a system pointer or a space pointer.

Notes on Usage
- The builtin function _MATOBJLK also provides access to the MI instruction.

Example
This example illustrates the use of the matobjlk function.

```c
#include <QSYSINC/MIH/MATOBJLK>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>
#include <stdlib.h>

_MOBJL_Template_T allocated_locks; /* matobjlk'. */
_SYSPTR some_object; /* The object being locked. */
```
int main(void) {
    /* Get a pointer to the object. */
    some_object = rslvsp(_Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL);

    /* Use the CL ALCOBJ command to place a lock on the user queue (this
    /* could also be done with the 'lock' function). 'matobjlk' is then
    /* used to materialize all of the object locks held for the queue. */
    */
    system("ALCOBJ OBJ((MYLIB/ MYUSRQ *USRQ *EXCL)) ");

    allocated_locks.Template_Size = sizeof(_MOBJL_Template_T);

    matobjlk(&allocated_locks, some_object);

    /* In order to verify that the Exclusive lock was placed on the
    object, we can look at the bit pattern returned by 'matobjlk' in
    the 'Lock Alloc' field of the template. The fifth bit (bit 4) of
    this field represents the Lock-Exclusive-No-Read (LENR) lock. By
    using the bitwise AND (&) operator, we can determine if the bit is
    set by ANDing it with the provided bit mask, _LENR_LOCK (which
    is defined in <milock.h> as hex 08 - bit pattern 0000 1000).
    */
    if (allocated_locks.Lock_Alloc & _LENR_LOCK) {
        printf("There is an Exclusive-No-Read lock on the object \n");
    }
    else
        printf("Error. An Exclusive-No-Read lock is not held on the object.\n");
}

Output
There is an Exclusive-No-Read lock on the object.
Materialize Program (MATPG)

**Format**
```
#include <QSYSINC/MIH/MATPG>

void matpg (_MATPG_Template_T *receiver,
            _SYSPTR program);
```

**Description**
The matpg function materializes the program into the receiver template. The values in the materialization relate to the current attributes of the program. Components of the program template, other than the control information component, may not be available for materialization because they were removed with the deletion of observability or were absent when the program was created.

**Parameters**
- **receiver (input/output)**
  - Pointer to the receiver template.
- **program (input)**
  - Pointer to the program object to be materialized.

**Notes on Usage**
- The matpg function can only be used with OPM programs.
- The builtin function _MATPG also provides access to the MI instruction.
Example
This example illustrates the use of the matpg function.

```c
#include <stdio.h>
#include <QSYSINC/MIH/MATPG>
#include <QSYSINC/MIH/RSLVSP>
#include <stdlib.h>

int main(void) {
    _SYSPTR pgm_ptr;
    _MATPG_Template_T mpg_t;

    memset(&mpg_t, 0, sizeof(_MATPG_Template_T));

    /* Create an OPM program to pass to matpg */
    system("CRTCLPGM QTEMP/T1520CL1 QCLE/QACLSSRC");
    pgm_ptr = rslvsp(_Program, "T1520CL1", "QTEMP", _AUTH_ALL);

    mpg_t.Template_Size = sizeof(_MATPG_Template_T);
    matpg(&mpg_t, pgm_ptr);

    printf("The program was created for version/release: %x\n",
            mpg_t.Lang_Version);
}
```

Output
The program was created for version/release: 370
Materialize Pointer (MATPTR)

Format
#include <QSYSINC/MIH/MATPTR>

void matptr (_MPTR_Template_T *receiver, _ANYPTR pointer);

Description
The matptr function returns the materialized form of pointer in the receiver template.

Parameters
receiver (input/output)
   Pointer to the receiver template.

pointer (input)
   The pointer to materialize. May be any one of the ILE pointer types: system pointer, space pointer, invocation pointer, procedure pointer, label pointer, or suspend pointer.

Notes on Usage
- The builtin function _MATPTR also provides access to the MI instruction.
- Constants for pointer types are defined in <milib.h>.
Example
This example illustrates the use of the matptr function.

```c
#include <QSYSINC/MIH/MATPTR>
#include <QSYSINC/MIH/RLSVSP>
#include <stdio.h>

int main(void) {
    _SYSPTR  lib_ptr;
    _MPTR_Template_T mpt;

    /* Resolve to program MYPGM in the current library list */
    lib_ptr = rslvsp(_Program, "MYPGM", "*LIBL", _AUTH_OBJ_MGMT);

    /* Materialize the program pointer */
    mpt.Obj_Ptr.Template_Size = sizeof(_OBJPTR_T);
    matptr(&mpt, lib_ptr);

    printf("Object name : %.10s\n", mpt.Obj_Ptr.Object_ID.Name);
    printf("Library : %.10s\n", mpt.Obj_Ptr.Library_ID.Name);
    printf("Type : %x\n", mpt.Obj_Ptr.Ptr_Type);
    printf("Authority : %u\n",
            mpt.Obj_Ptr.Auth_Or_Off.Ptr_Authorization);
}
```

Output
Object name : MYPGM
Library : QTEMP
Type : 1
Authority : 0
Materialize Pointer Locations (MATPTRL)

Format
#include <QSYSINC/MIH/MATPTRL>

void matptrl (_MPTL_Template_T *receiver, 
_SPCPTRCN source, 
int length);

Description
The matptrl function finds the pointers in a subset of a space and produces a bit mapping of their relative locations.

The area addressed by the source space pointer is scanned for a length equal to that specified in length. A bit in receiver is set for each 16 bytes of source. A bit in receiver is set to binary 1 if a pointer exists in the source space, or the bit is set to binary 0 if no pointer exists in the source space.

Bits are set from left to right (bit0, bit1,...) in receiver as the 16-byte areas in source are interrogated from left to right.

Parameters
receiver (input/output)
    Pointer to the receiver template.

source (input)
    Pointer to the source area. The area addressed must be 16-byte aligned.

length (input)
    The length of area to scan.

Notes on Usage
- The builtin function _MATPTRL also provides access to the MI instruction.
Example
This example illustrates the use of the matptrl function.

```c
#include <QSYSINC/MIH/MATPTRL>
#include <QSYSINC/MIH/RLVSP>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma linkage(PGMA, OS)
void PGMA(void);
void print_bitmap(void);

#define SIZE 128

_MPTL_Template_T mpt1;

int main(void) {
    _SYSPTR pgm_ptr, qtemp_ptr, spc_ptr;
    _SPCPRTR source;

    /* Allocate initialized storage for source */
    source = (_SPCPRTR)malloc(SIZE, 1);

    /* Produce a bit map of source before storing pointers. */
    mpt1.Template_Size = sizeof(_MPTL_Template_T);
    matptrl(&mpt1, source, SIZE);

    print_bitmap();

    /* Resolve to user space MYSPACE in the current library list */
    spc_ptr = rslvsp(_Usrspc, "MYSPACE", "*LIBL", _AUTHNONE);

    /* Store pointers in source at offsets 16, 80 and 112. This */
    /* will cause bits 1, 5 and 7 (starting with bit 0 on the */
    /* left) to be set to binary 1 in the resulting bit map. */

    pgm_ptr  = (_SYSPTR)PGMA;
    qtemp_ptr = _QTEMP_POINTER;

    memcpy(source + 16, &pgm_ptr, sizeof(pgm_ptr));
    memcpy(source + 80, &qtemp_ptr, sizeof(qtemp_ptr));
    memcpy(source + 112, &spc_ptr, sizeof(spc_ptr));

    /* Produce a bit map of source */
    matptrl(&mpt1, source, SIZE);
}
```
```c
print_bitmap();
}

void print_bitmap(void) {
    unsigned pos = 7;

    printf("Offset in source (bytes)  Pointer Found\n");
    printf("  0 - 15  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  16 - 31  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  32 - 47  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  48 - 63  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  64 - 79  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  80 - 95  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf("  96 - 111  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
    printf(" 112 - 127  %s\n",
            (mpt1.Locations & (1 << pos--)) ? "Yes" : "No");
}

Output
Offset in source (bytes)  Pointer Found
  0 - 15                   No
  16 - 31                  No
  32 - 47                  No
  48 - 63                  No
  64 - 79                  No
  80 - 95                  No
  96 - 111                 No
 112 - 127                 No

Offset in source (bytes)  Pointer Found
  0 - 15                   No
  16 - 31                  Yes
  32 - 47                  No
  48 - 63                  No
  64 - 79                  No
  80 - 95                  Yes
  96 - 111                 No
 112 - 127                 Yes
```
Materialize Process Activation Groups (MATPRAGP)

**Format**
```
#include <QSYSINC/MIH/MATPRAGP>

void matpragp (_MPRAG_Template_T *receiver);
```

**Description**
The matpragp function provides a list of the activation groups which exist in the current process.

**Parameters**
receiver (input/output)
  Pointer to the receiver template. The materialization template identified by the receiver must be 16-byte aligned in memory.

**Notes on Usage**
- A macro version is available.
- The builtin function _MATPRAGP also provides access to the MI instruction.
Example

This example illustrates the use of the matprag function.

```c
#include <QSYSINC/MIH/MATPRAGP>
#include <stdio.h>
#include <stdlib.h>

_MPRAG_Template_T *template;  /* Pointer to the receiver */

int actgrp,  /* Loop counter for displaying */
    /* each activation group */
    /* number. */

new_size;  /* Number of bytes required */
    /* to hold all information */
    /* returned by 'matprag'. */

int main(void) {

    /* Call 'matprag' first just to find out how many activation groups */
    /* there are in the job/process that is running this program. */
    template = (_MPRAG_Template_T *) malloc( sizeof(_MPRAG_Template_T) );
    template->Template_Size = sizeof(_MPRAG_Template_T);
    matprag( template );
    printf("Number of activation groups associated with this job: \%d \n",
            template->Act_Grp_Count );

    /* Now that we know exactly how many activation groups there are, */
    /* we can allocate enough space to accommodate the "variable-length" */
    /* portion of the template and then call 'matprag' again to get all */
    /* of the activation group numbers. Each activation group number is */
    /* 4-bytes long. */
    new_size = sizeof(_MPRAG_Template_T) + (template->Act_Grp_Count * 4 );

```
template = (_MPRAG_Template_T *) realloc( template, new_size );

template->Template_Size = new_size;

matpragp( template );

/*-------------------------------------------------------------*/
/* Loop through and output each activation group number in an output */
/* field of 8 positions.                                       */
/*-------------------------------------------------------------*/

for (actgrp=0; actgrp < template->Act_Grp_Count; actgrp++) {
    printf("%8d \n", template->Act_Grp_List[ actgrp ]);
}

/*-------------------------------------------------------------*/
/* You can also see the activation groups by using the CL DSPJOB */
/* command (but using 'matpragp' is faster). To call the command */
/* from this program you would code the following:              */
/*    system( "DSPJOB OPTION(\#ACTGRP)" );                     */
/*-------------------------------------------------------------*/

Output
Number of activation groups associated with this job:  3
  3977
   2
   1
Materialize Process Attributes (MATPRATR)

Format
#include <QSYSISNC/MIH/MATPRATR>

void matpratr (_MPRT_Template_I *receiver,
             _SYSPTR control_spc,
             char options);

Description
The matpratr function causes either one specific attribute or all the attributes of the
designated process to be materialized.

Parameters
receiver (input/output)
  Pointer to the receiver template.

control_spc (input)
  The process control space pointer or NULL. A value of NULL indicates that the
  process issuing the instruction is the subject process.

options (input)
  The materialization options.

Notes on Usage
- The builtin functions _MATPRATR1 and _MATPRATR2 also provide access to
  the MI instruction.
Example
This example illustrates the use of the matpratr function.

```c
/*---------------------------------------------*/
/* */
/* Example Group: Processes <SYSINC/MIH/MATPRATR> */
/* */
/* Function: matpratr (Materialize Process Attributes) */
/* */
/* Description: This function can return a lot of different */
/* information regarding the job/process in which it */
/* is run or for some other process. This example */
/* shows how to return information on the number */
/* of synchronous read and write operations */
/* performed by the specified job/process. */
/* */
/*---------------------------------------------*/
#include <SYSINC/MIH/MATPRATR>
#include <stdio.h>

_MPRT_Template_T process_attributes; /* The receiver template for */
/*  'matpratr'. */
int main(void) {

    /*---------------------------------------------*/
    /* Materialize performance information for this job/process (since */
    /* NULL is used for the second argument of 'matpratr'). */
    /*---------------------------------------------*/

    process_attributes<Scalar_Attr.Template_Size = sizeof(_MPRT_Template_T);
    matpratr( &process_attributes, NULL, _MPRT_PROC_PERF);

    printf("Currently this process has the following number of \n");
    printf("Reads and Writes: \n\n");

    printf("Synchronous Database Reads: %d\n", 
            process_attributes<Scalar_Attr.Data.Proc_Perf.Num_Read_DB_S >);;

    printf("Synchronous Non-Database Reads: %d\n", 
            process_attributes<Scalar_Attr.Data.Proc_Perf.Num_Read_NDB_S >);

    printf("Total Synchronous Writes (both DB and NDB): %d\n", 
            process_attributes<Scalar_Attr.Data.Proc_Perf.Num_Write_S >);
}

Output
Currently this process has the following number of
Reads and Writes:

Synchronous Database Reads: 438
Synchronous Non-Database Reads: 765
Total Synchronous Writes (both DB and NDB): 1203
Materialize Process Locks (MATPRLK)

Format
#include <QSYSINC/MIH/MATPRLK>

void matprlk (_MPRL_Template_T *receiver,
    _SYSPTR pcs);

Description
The matprlk function materializes the lock status of the specified process. This
information identifies each object or space location for which the process has a lock
allocated or for which the process is in a synchronous or asynchronous wait.

Parameters
receiver (input/output)
    Pointer to the receiver template where the materialized lock status will be
    returned.

pcs (input)
    Pointer to the process control space of the process whose lock status is to be
    materialized or NULL. If NULL is specified, the lock status of the current
    process is materialized.

Notes on Usage
• The builtin functions _MATPRLK1 and _MATPRLK2 also provide access to the
  MI instruction.

Example
This example illustrates the use of the matprlk function.

/**************************************************************************
/* Example Group:    Locks    <QSYSINC/MIH/MATPRLK>
/* Function:        matprlk (Materialize Process Locks)
/* Description:     The CL ALCOBJ (Allocate Object) command is used to place
                  a lock on some object. Very much like the 'mataol' and 'matobjlk'
                  examples, we will use 'matprlk' to materialize and display the locks.
                  However, unlike the other two functions, 'matprlk' provides
                  information about all of the objects locked by the job/process that runs
                  this program as opposed to lock information for a particular
                  object.
/**************************************************************************/

#include <QSYSINC/MIH/MATPRLK>
#include <QSYSINC/MIH/RLSVSP>
#include <stdio.h>
#include <stdlib.h>

    _MPRL_Template_T  *process_locks;   /* Pointer to receiver template.*/
```c
int new_size, /* Number of bytes needed for */
    /* the fixed and variable parts */
    /* of the 'matprlk' template. */
lock_count; /* Loop counter to loop through */
    /* each lock held by the job. */

int main(void) {
    /* Use the CL ALCOBJ command to place an Exclusive, Allow Read lock */
    /* on a user queue (we could have used 'lock' to place a LEAR lock on */
    /* the object to accomplish the same thing). */
    system( "ALCOBJ OBJ((MYLIB/MYUSRQ *USRQ *EXCLRD)) ");

    process_locks = (_MPRL_Template_T *) malloc( sizeof(_MPRL_Template_T) );
    process_locks->Template_Size = sizeof(_MPRL_Template_T);
    matprlk( process_locks, NULL );

    printf("Number of objects that have locks owned by this job/process:");
    printf("%d \n\n", process_locks->Num_Entry_Exp );

    new_size = sizeof(_MPRL_Template_T) +
               ( process_locks->Num_Entry_Exp * sizeof(_LOCK_Descript_T ) );
    process_locks = (_MPRL_Template_T *) realloc( process_locks, new_size );
    process_locks->Template_Size = new_size;
    matprlk( process_locks, NULL );
```
printf("This job/process holds locks to the objects pointed to \n");
printf("by the following pointers: \n");

for ( lock_count = 0;
    lock_count < process_locks->Num_Entry_Exp;
    lock_count++ ) {
    printf("Object %d: pointed to by: %p \n",
            lock_count+1, process_locks->Locks [ lock_count ] );
}

Output
Number of objects that have locks owned by this job/process: 18

This job/process holds locks to the objects pointed to by the following pointers:
Object 1: pointed to by: SYP:8100:0401QUSRYSYS:0:1247
Object 2: pointed to by: SYP:8100:0401QSYS2:0:1247
Object 3: pointed to by: SYP:8100:0401QHLPSYS:0:1247
Object 4: pointed to by: SYP:8100:0401QADM:0:1247
Object 5: pointed to by: SYP:8100:04c1QTEMP:0:1247
Object 6: pointed to by: SYP:8100:0401QSYS:0:1247
Object 7: pointed to by: SYP:8100:0801VISCA:0:1247
Object 8: pointed to by: SYP:8100:0401QPDA:0:1247
Object 9: pointed to by: SYP:0000:18a0QJOBMSGQ:0:1247
Object 10: pointed to by: SYP:8100:1001QPADEV0003:0:1247
Object 11: pointed to by: SYP:0401QSYS:1916MAIN:0:1247
Object 12: pointed to by: SYP:8100:0401SALMI:0:1247
Object 13: pointed to by: SYP:0000:19dfQMMESSAGEQUEUELOCKOBJECT:0:1247
Object 14: pointed to by: SYP:0401QSYS:1901QDU180:0:1247
Object 15: pointed to by: SYP:8100:0401GPL:0:1247
Object 16: pointed to by: SYP:0401MYLIB:0a02MYUSRQ:0:0:1247
Object 17: pointed to by: SYP:0401QSYS:1901QSNB0:0:1247
Object 18: pointed to by: SYP:0401QPDA:1901QDUODSPF:0:1247
Materialize Queue Attributes (MATQAT)

Format

```c
#include <QSYSINC/MIH/MATQAT>

void matqat (_MQAT_Template_T *receiver,
             _SYSPTR queue);
```

Description
The matqat function materializes the attributes of the specified queue.

Parameters

receiver (input/output)
Pointer to the receiver template.

queue (input)
Pointer to the queue whose attributes are to be materialized.

Notes on Usage
- The builtin function _MATQAT also provides access to the MI instruction.

Example
This example illustrates the use of the matqat function.

```c
#include <QSYSINC/MIH/MATQAT>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>

_MQAT_Template_T queue_attributes; /* The receiver template for */
   /* 'matqat'. */
_SYSPTR queue_ptr; /* Pointer to the user queue. */

int main(void) {
    /* Get a pointer to the +USRQ. */
    queue_ptr = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Materialize the attributes of the user queue and output some of */
    /* them to the screen. */
    /* -------------------------------------------------------------------------*/
```
queue_attributes.Template_Size = sizeof(_MQAT_Template_T);

matqat( &queue_attributes, queue_ptr );

printf("Number of entries currently on the queue: %d\n",
    queue_attributes.Num_Msgs );

printf("Initial number of entries allowed: %d\n",
    queue_attributes.Max_Msgs );

printf("Additional number of entries allowed: %d\n",
    queue_attributes.Extension);;

printf("Maximum size of any particular entry: %d\n",
    queue_attributes.Max_Size );
}

**Output**

- Number of entries currently on the queue: 3
- Initial number of entries allowed: 10
- Additional number of entries allowed: 50
- Maximum size of any particular entry: 75
Materialize Queue Messages (MATQMSG)

Format
#include <QSYSINC/MIH/MATQMSG>

void matqmsg (_MQMS_Template_T *receiver
  _SYSPTR queue,
  char selection,
  int max_key,
  int max_msg,
  _SPCPTTR key);

Description
The matqmsg function materializes selected messages on a queue. The number of
messages materialized and the amount of key and message text materialized for
each message are controlled through the function's parameters.

Parameters
receiver (input/output)
  Pointer to the receiver template where the materialized message and attributes
  will be placed. This template must be 16-byte aligned.

queue (input)
  Pointer to the queue.

selection (input)
  The message selection criteria. Constants for the allowable values are defined
  in the header file.

max_key (input)
  The number of key bytes to materialize. It must be a multiple of 16.

max_msg (input)
  The number of message text bytes to materialize. It must be a multiple of 16.

key (input)
  Pointer to the message key. This must be a null-terminated string or consist of
  exactly 256 bytes.

Notes on Usage
  • The builtin function _MATQMSG provides access to the MI instruction.
**Example**

This example illustrates the use of the matqmsg function.

```c
/* Example Group: User Queues <QSYSINC/MIH/MATQMSG> */
/* Function: matqmsg (Materialize Queue Messages) */
/* Description: Materialize selected messages from a user queue. */
/* */

#include <QSYSINC/MIH/MATQMSG>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

_MQMS_Template_T *queue_messages; /* The pointer to the receiver */
char *this_message_data; /* Pointer used to loop */
/* through each message. */
/sysptr queue_ptr; /* Pointer to the user queue. */
int message, /* Loop counter for looping */
/* through each message. */
key_length = 0, /* Not materializing by key so */
/* set the key length to 0. */
message_length = 80, /* Even though the message size */
/* is 75, this value must be a */
/* multiple of 16, so use 80. */
new_size; /* Number of bytes needed for */
/* both the fixed and variable */
/* portion of the template. */
char *key = " "; /* Not a keyed User Queue so */
/* this key will not be used. */

int main(void) {
    /* Get a pointer to the *USRQ. */
    queue_ptr = rslvsp(_Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL);
    /* Allocate just enough space for the fixed portion of the receiver */
    /* template. Then call 'matqmsg' to fill in the template with */
    /* information about all of the messages on the queue. */
    queue_messages = (_MQMS_Template_T *)
        malloc( sizeof(_MQMS_Template_T ) );
    queue_messages->Template_Size = sizeof( _MQMS_Template_T );
}```
matqmsg

matqmsg( queue_messages, queue_ptr, _MQMS_ALL,
        key_length, message_length, key );

printf("Number of messages materialized: %d\n",
        queue_messages->Mat_Mgs );

printf("Number of messages on the queue: %d\n",
        queue_messages->Num_Msgs );

printf("Maximum message size: %d\n",
        queue_messages->Max_Size );

/**********************************************************/
/* Now that we know exactly how many messages there are, we can */
/* allocate enough space to accommodate the variable-length portion */
/* of the template providing us with information about each message. */
/* Since this is a "fixed-message-length" user queue, we know the */
/* size of each "message" will be the fixed size + the number of */
/* bytes required for the other fields in _MQMS_Data_T. */
/**********************************************************/

new_size = sizeof(_MQMS_Template_T) +
            ( queue_messages->Num_Msgs *
              (sizeof(_MQMS_Data_T) + queue_messages->Max_Size )
            );

queue_messages = (_MQMS_Template_T *)
                    realloc( queue_messages, new_size );

queue_messages->Template_Size = new_size;

matqmsg( queue_messages, queue_ptr, _MQMS_ALL,
        key_length, message_length, key );

/**********************************************************/
/* Output each message that was returned to us in the storage area */
/* pointed to by 'queue_messages'. 'Message_Data' is the first byte */
/* of the first message structure (_MQMS_Data_T). 'this_message_data' */
/* pointer will be used to access the actual message in this */
/* structure. The pointer is then moved through all of the */
/* subsequent message data structures to access the queued messages. */
/* Since the _MQMS_Data_T structure only has one byte for the message */
/* we have to add the length of the message to reset the pointer to */
/* the next message data structure. Even though the message length */
/* is 75 bytes, we have to round up to a 16-byte boundary, so we use */
/* 80 as the message length. To get to the next message in the next */
/* message data structure, we also need to add the offset at which */
/* the 'Message' member is defined within the _MQMS_Data_T structure. */
/**********************************************************/

this_message_data = &(queue_messages->Message_Data) +
                    offsetof( _MQMS_Data_T, Message );
for ( message=0; message < queue_messages->Mat_Msgs; message++ ) {
    printf("Message %3d: %.75s\n", message+1, this_message_data);
    this_message_data += 80 + offsetof( _MQMS_Data_T, Message );
}

**Output**
Number of messages materialized: 2
Number of messages on the queue: 2
Maximum message size: 75
Message 1: This is the first message being entered.
Message 2: This is the second message being entered.
Materialize Resource Management Data (MATRMD)

**Format**

```c
#include <QSYSINC/MIH/MATRMD>

void matrmd (_MATRMD_Template_T *receiver,
             _SPCPTR control);
```

**Description**

The matrmd function materializes the resource management data.

**Parameters**

**receiver (input/output)**

Pointer to the receiver template where the resource management data will be
materialized.

**control (input)**

Pointer to the 8-byte character control data. This argument identifies the type
of information to be materialized.

**Notes on Usage**

- A macro version is available.
- The builtin function _MATRMD also provides access to the MI instruction.

**Example**

This example illustrates the use of the matrmd function.

```c
#include <QSYSINC/MIH/MATRMD>
#include <stdio.h>
#include <stdlib.h>

_MATRMD_Template_T *RMD_template; /* template for 'matrmd'. */

char control[ 8 ]; /* The control option that tells 'matrmd' what information */
                        /* should be returned. */

int main(void) {
```
/***************************************************************************/
/* Call 'matrmd' to get the number of Auxiliary Storage Pools (ASPs) */
/* and the number of allocated auxiliary storage units. This */
/* information is contained in the first 32 bytes of the template */
/* so there is no need to materialize the whole thing. */
/****************************************************************************/

control[ 0 ] = _MATRMD_AUX_STORAGE;
RMD_template = ( _MATRMD_Template_T * ) malloc( 32 );
RMD_template->Template_Size = 32;
matrmd( RMD_template, control );

printf("Number of Auxillary Storage Pools (ASPs): %3d \n",
       RMD_template->MATRMD_Data.Aux_Storage.Num_ASP );
printf("Number of Allocated Auxillary Storage Units: %3d \n",
       RMD_template->MATRMD_Data.Aux_Storage.Num_Alloc_Aux );
}

**Output**
Number of Auxiliary Storage Pools (ASPs): 1
Number of Allocated Auxiliary Storage Units: 8
Materialize Space Attributes (MATS)

**Format**
```c
#include <QSYSINC/MIH/MATS>

void mats (_SPC_Template_T *receiver
            _SYSPTR space_object);
```

**Description**
The mats function materializes the current attributes of the space object into the receiver template.

**Parameters**
- `receiver (input/output)`
  Pointer to the template where the attributes of the space object are materialized.

- `space_object (input)`
  Pointer to the space object whose attributes are to be materialized.

**Notes on Usage**
- The builtin function _MATS also provides access to the MI instruction.
Example
This example illustrates the use of the mats function.

```c
/* Example Group: Space Management <QSYSINC/MIH/MATS> */
/* Function: mats (Materialize Space Attributes) */
/* Description: This example uses 'mats' to materialize the */
/* size of a space object. (Note that the size is */
/* always at least as large as requested and a power */
/* of two.) */
/* */
/*****************************************************************************/

#include <stdio.h>
#include <QSYSINC/MIH/MATS>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/H/QUSCRTUS>

#define CREATION_SIZE 65500

int main(void)
{

_SPC_Template_T space_t;
_SYSPTR ptr_to_space;
int error_code = 0;

QUSCRTUS("MYSPACE QTEMP ",
"MYSPACE ",
CREATION_SIZE,
"\0",
"*ALL ",
"MYSPACE example for Programmer's Reference ",
"*YES ",
&error_code);

ptr_to_space = rslvsp(_Usrspc, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);

space_t.TmpSize = sizeof(_SPC_Template_T);

mats(&space_t, ptr_to_space);

printf("The actual size of MYSPACE is %d bytes\n", space_t.Size);
}
```

Output
The actual size of MYSPACE is 65536 bytes
Materialize Selected Locks (MATSELLLK)

Format
#include <QSYSINC/MIH/MATSELLLK>

void matsellk (_MSLL_Template_T *receiver,
                _ANYPTR pointer);

Description
The matsellk function materializes the locks held by the process issuing this instruction for the object or space location specified.

Parameters
receiver (input/output)
Pointer to the receiver template where the materialized locks will be returned.

pointer (input)
Pointer to the system object or space location whose locks are to be materialized. The argument must be a system pointer or a space pointer.

Notes on Usage
• The builtin function _MATSELLLK also provides access to the MI instruction.

Example
This example illustrates the use of the matsellk function.
/*-------------------------------------------------------------*/
/*
/* Example Group: Locks <QSYSINC/MIH/MATSELLLK>
/*
/* Function: matsellk (Materialize Selected Locks)
/*
/* Description: Use the CL ALCOBJ (Allocate Object) command to
/* place a lock on some object. Very much like
/* the 'mataol' and 'matobj1k' examples, 'matsellk'
/* will be used to verify the lock.
/*
/*-------------------------------------------------------------*/
#include <QSYSINC/MIH/MATSELLLK>
#include <QSYSINC/MIH/RLVSP>
#include <stdio.h>
#include <stdlib.h>

 _MSLL_Template_T allocated_locks; /* 'matsellk'. */
 _SYSPTER some_object; /* The object being locked. */

int main(void) {
    /* Get a pointer to the object. */
    some_object = rlvsp(_Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL);
}
/* Use the CL ALCOBJ command to place a lock on the user queue (this */ */ could also be done with the 'lock' function). 'matobjlk' is then */ */ used to materialize all of the object locks held for the queue.  * /

system( "ALCOBJ OBJ((MYLIB/MYUSRQ *USRQ *EXCL)) ");

allocated_locks.Template_Size = sizeof( _MSLL_Template_T );

matsellk( &allocated_locks, some_object );

/* In order to verify that the Exclusive lock was placed on the */ /* object, we can look at the bit pattern returned by 'matsellk'.  */ /* The 'Cum_Lock_Status' field in the template is the bit pattern */ /* that represents the "Current Cumulative Locks" and the fifth bit */ /* (bit 4) of this field represents the Lock-Exclusive-No-Read (LENR) */ /* lock. By using the bitwise AND (&) operator, we can determine if */ /* the bit is set by ANDing it with the provided bit mask, _LENR_LOCK */ /* (which is defined in <milock.h> as hex 08 - bit pattern 0000 1000).*/

if ( allocated_locks.Cum_Lock_Status & _LENR_LOCK ) {
    printf("There is an Exclusive-No-Read lock on the object \n");
} else
    printf("Error. An Exclusive-No-Read lock is not held on the object.\n");

Output
There is an Exclusive-No-Read lock on the object.
Materialize System Object (MATSOBJ)

Format
#include <QSYSINC/MIH/MATSOBJ>

void matsobj (_MSOB_Template_T *receiver, 
(SYSPTR object);

Description
The matsobj function materializes the identity and size of a system object.

Parameters
receiver (input/output)
  Pointer to the receiver template.

object (input)
  Pointer to the system object whose attributes are to be materialized.

Notes on Usage
- The builtin function _MATSOBJ also provides access to the MI instruction.

Example
This example illustrates the use of the matsobj function.

#include <QSYSINC/MIH/MATSOBJ>
#include <stdio.h>
#include <stdlib.h>

#pragma linkage (SAMPLE, OS)
void SAMPLE(void);

int main(void) {
  _MSOB_Template_T msob_template;

  /* Create an ILE C program called SAMPLE. */
  system("CRTBNDC QTEMP/SAMPLE QCLE/QACSRC");

  msob_template.Template_Size = sizeof(_MSOB_Template_T);
  matsobj(&msob_template, SAMPLE);

  /* Is SAMPLE an ILE program? */
  printf("SAMPLE is an %s program\n", 
      (msob_template.Pgm_Type ? "ILE" : "OPM"));
}
Output
SAMPLE is an ILE program
Materialize Time-of-Day Clock (MATTOD)

Format

```c
#include <QSYSIN/MIH/MATTOD>

void mattrd (_MI_Time time_of_day);
```

Description
The mattrd function materializes the time of day clock.

Parameters

- `time_of_day` (output)
  An 8-byte character array into which the time-of-day clock is materialized.

Notes on Usage
- A macro version is available.
- The built-in function _MATTOD also provides access to the MI instruction.
Example
This example illustrates the use of the mattod function.

```c
/* Example Group: Machine Interface <QSYSINC/MIH/MATTOD> */
/* */
/* Function: mattod (Materialize Time-Of-Day) */
/* */
/* Description: Use the 'mattod' MI function to get the current */
/* MI Time-Of-Day (TOD) value. The function returns */
/* an 8-byte value represented by the type _MI_Time */
/* (defined in <milib.h>) which represents a large */
/* "counter" in which bit 41 (starting at offset 0) */
/* is incremented approximately once every */
/* millisecond (1/1000th of a second). The TOD is */
/* returned by several MI functions and is also used */
/* (as input) by other MI functions. */
/* */
#include <QSYSINC/MIH/MATTOD>
#include <stdio.h>

_MI_Time time_of_day;

int byte;

int main(void) {

    mattod(time_of_day);

    printf("Time of Day returned by 'mattod' in the _MI_Time format: ");

    for (byte=0; byte < sizeof(_MI_Time); byte++)
        printf("%2.2X", time_of_day[byte] );

    printf("\n");
}
```

Output
Time of Day returned by 'mattod' in the _MI_Time format: 74803E7FEB400017
Machine Interface Time (MITIME)

**Format**

```c
#include <QSYSINC/MIH/MICOMMON>
_MI_TIME *mitime (_MI_Time *receiver, int hours,
   int minutes, int seconds, int hundredths);
```

**Description**

The mitime function takes, as parameters, hours, minutes, seconds, and hundredths of seconds, and converts them to the AS/400 system value for time which has the data type _MI_Time. Many of the MI library functions use this _MI_Time data type.

**Parameters**

**receiver(output)**

Pointer to an 8-byte character array which is to receive the system value for the specified time.

**hours(input)**

The number of hours to be converted to the system value for time.

**minutes(input)**

The number of minutes to be converted to the system value for time.

**seconds(input)**

The number of seconds to be converted to the system value for time.

**hundredths(input)**

The number of 1/100 seconds to be converted to the system value for time.

**Notes on Usage**

- The maximum system time that can be input to mitime is such that the total number of hours, minutes, seconds and hundredths of seconds specified must not exceed (UINT_MAX * 1024 / 1,000,000) seconds which is about 50 days. If this value is exceeded, the results from mitime are undefined.

**Example**

This example illustrates the use of the mitime function.
/************************* /
/*
/* Example Group: Job Information <QSYSINC/MIH/MICOMMON> */
/*
/* Function: mitime (convert time to AS/400 system value */
/* for time)
/*
/* Description: Use the 'mitime' MI function to convert */
/* specific values for the components of time to */
/* the AS/400 system value for time. Then use the */
/* formatted _MI_Time value to suspend or make */
/* the job go to sleep for the specified amount */
/* of time.
/*
*/
/************************* /

#include <QSYSINC/MIH/MICOMMON>
#include <QSYSINC/MIH/WAITTIME>
#include <stdio.h>

_MI_Time time_to_wait; /* The amount of time to wait. */
int hours = 0, /* Time components used to */
minutes = 0, /* create an _MI_Time value */
seconds = 15, /* that can be passed to the */
hundredths = 0; /* 'waittime' function */

short wait_option;

int main(void) {

/************************* /
/* Format an 'mitime' value to represent the amount of time to wait. */
/* When 'waittime' is called, the job that is running this program */
/* will be suspended. */
/************************* /

mitime( &time_to_wait, hours, minutes, seconds, hundredths );

/* Tells the system to use normal */
wait_option = _WAIT_NORMAL; /* handling of a suspended job. */

waittime( &time_to_wait, wait_option );
}
Output
** no output **
Modify Automatic Storage Allocation (MODASA)

**Format**

```c
#include <QSYSINC/MIH/MODASA>

_SPCPTR modasa (unsigned int size);
```

**Description**

The modasa function extends the size of the automatic storage frame (ASF) assigned to the invocation of the currently executing program. The function returns the address of the first byte of the ASF extension. This extension might not be contiguous with the original allocation.

**Parameters**

- **size (input)**
  - The size of the adjustment. This value must be greater than 0.

**Notes on Usage**

- A macro version is available.
- The builtin function _MODASA also provides access to the MI instruction.
- The _MODASA builtin is non-resumable following an exception.
- If an attempt is made to resume execution without first changing the resume point, an MCH2204 is signalled.
Example
This example illustrates the use of the modasa function.

```c
#include <QSYSINC/MIH/MODASA>
#include <string.h>

int additional_bytes = 2000; /* The size by which to extend */
char *ptr; /* A pointer that will be set */
int main(void) {
    ptr = modasa( additional_bytes );
    strcpy( ptr, "Some string" );
}
```

Output
** no screen output **
Modify Independent Index (MODINX)

**Format**

```c
#include <QSYSINC/MIH/MODINX>

void modinx (_SYSPTR index_obj,
             char mod_option);
```

**Description**

The modinx function modifies the selected attributes of the independent index.

**Parameters**

**index_obj (input)**

Pointer to the independent index object whose attributes are to be modified.

**mod_option (input)**

The modification option. The valid values are:

- 0 = No immediate update
- 1 = Immediate update

**Notes on Usage**

- The builtin function _MODINX also provides access to the MI instruction.

**Example**

This example illustrates the use of the modinx function.

```c
#include <QSYSINC/MIH/MODINX>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/MIH/MATINXAT>
#include <stdio.h>
#include <string.h>

_IIX_Template_T   index_template;

_SYSPTR           index_ptr;  /* Pointer to the user index */

/* The index attributes returned by 'matinxat' will be copied into */
/* the following structure of bit-fields so that the single bit we */
/* want to check is readily available by name. */

struct index_attributes {
    int entry_type : 1;    /* Fixed or Variable-length entries*/
    int when_updated : 1;  /* Whether or not Immediate Update */
    int insert_type : 1;   /* Keyed or Sequential inserts? */
    int type_of_data : 1;  /* Contain pointers or not? */
    int optimize : 1;      /* Optimized for Random/Sequential */
    int reserved : 3;      /* 3 bits unused */
} index_attributes;

#define IMMEDIATE_UPDATE  1
#define NO_IMMEDIATE_UPDATE 0

int main(void) {
    /* Get a pointer to the +USRIDX. */
    index_ptr = rslvsp( _Usridx, "MYUSRIDX", "MYLIB", _AUTH_ALL );

    /******************************************************************************
     * Materialize the index attributes and store the "attribute" settings (8-bits).
     * Check the "immediate update" bit (the second bit-field in the structure as
     * shown in the 'index_attributes' structure above) and turn it on to represent
     * the "immediate update" option if it is not already set.
     ******************************************************************************/

    index_template.Template_Size = sizeof( _IIX_Template_T );
    matinxat( &index_template, index_ptr );

    /* Copy the 1-byte attribute field into a structure of bit-fields so we can
    * isolate the "immediate update" flag by name (rather than using the logical
    * AND (&) operator with a bit mask).
    */
    memcpy( &index_attributes, &index_template.Attributes, 1 );

    if ( index_attributes.when_updated != IMMEDIATE_UPDATE ) {
        /* Since the user index did not have the "immediate update" attribute, modify it. */
        modinx( index_ptr, IMMEDIATE_UPDATE );

        printf("The user index attributes were modified so that any update (eg: entry inserts or removals) will cause the index to be written to auxiliary storage.\n");
    }
    else {
        printf("The user index already had the 'immediate update'\n");
        printf("option set on.\n");
    }
}
Output
The user index attributes were modified so that any update (e.g. entry inserts or removals) will cause the index to be written to auxiliary storage.
Modify Space Attributes (MODS)

Format
#include <QSYSINC/MIH/MODS>

void mods (_SYSPTR space_object,
        int size);

Description
The mods function modifies the size of the space associated with the system object. The current allocation of the space is extended or truncated accordingly to match as closely as possible the specified size. The modified space size will be at least the size specified.

Parameters
space_object (input)
    Pointer to the system object whose associated space size is to be modified.

size (input)
    Size in bytes to which the space size is to be modified.

Notes on Usage
- The functions mods and mods2 provide equivalent semantics as the MODS MI instruction.
- The builtin function _MODS also provides access to the MI instruction.
Example
This example illustrates the use of the mods function.

```c
#include <stdio.h>
#include <QSYSINC/MIH/MODS>
#include <QSYSINC/MIH/MATS>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/H/QUSCRTUS>
#include <QSYSINC/H/QUSEC>

#define INCREMENT 4096
#define CREATION_SIZE 65500

int main(void)
{
    _SPC_Template_T space_t;
    _SYSPTR ptr_to_space;
    Qus_EC_t error_code;

    QUSCRTUS("MYSPACE QTEMP ",
             "MYSPACE ",
             CREATION_SIZE,
             "\0",
             "ALL ",
             "MYSPACE example for Programmer's Reference ",
             "YES ",
             &error_code);

    ptr_to_space = rslvsp(_Usrspc, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);
    space_t.TmpSize = sizeof(_SPC_Template_T);
    mats(&space_t, ptr_to_space);
    printf("The current size of MYSPACE is %d bytes\n", space_t.Size);
    mods(ptr_to_space, INCREMENT + space_t.Size);
    mats(&space_t, ptr_to_space);
    printf("The new larger size of MYSPACE is %d bytes\n", space_t.Size);
}```
Output
The current size of MYSPACE is 65536 bytes
The new larger size of MYSPACE is 69632 bytes
Modify Space Attributes - Long Form with Template (MODS2)

Format
#include <QSYSINC/MIH/MODS>

void mods2 (_SYSPTR space_object,
            _SPC_MOD_T *space_t);

Description
The mods2 function modifies the attributes of the space associated with the system object.

Parameters
space_object (input)
Pointer to the system object whose associated space attributes are to be modified.

space_t (input)
Pointer to the space modification template that contains a selection of space attribute values to be used to modify the attributes of the space.

Notes on Usage
- The functions mods and mods2 provide equivalent semantics as the MODS MI instruction.
- The builtin function _MODS2 also provides access to the MI instruction.
Example

This example illustrates the use of the mods2 function.

```c
#include <string.h>
#include <QSYSINC/MIH/MODS>
#include <QSYSINC/MIH/RLVSP>
#include <QSYSINC/H/QUSCRTUS>
#include <QSYSINC/H/QUSEIC>

#define CREATION_SIZE 65500

int main(void)
{
    _SPC_MOD_T mod_t;
    _SYSPTR ptr_to_space;
    Qus_EC_t error_code;

    QUSCRTUS("MYSPACE QTEMP ",
              "MYSPACE ",
              CREATION_SIZE,
              "\0",
              "*ALL ",
              "MYSPACE example for Programmer's Reference ",
              "*YES ",
              &error_code);

    /* Resolve to existing space */
    ptr_to_space = rslvsp(_Usrspc, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);

    memset(&mod_t, 0, sizeof(_SPC_MOD_T));

    /* Re-initialize the space to blanks */
    mod_t.Modify_Init_Val = 1;
    mod_t.Re_Init_Space = 1;
    mod_t.InitCh = 0x40;

    mods2(ptr_to_space, &mod_t);
}
```
Output
** no screen output **
The rslvsp function locates an object identified by a symbolic address and stores the object's addressability in a system pointer. The symbolic address consists of the object type, object name, and library. The system pointer returned by the function points to the first object encountered with the designated type/subtype code, object name, and library without regard to the authorization currently available to the process.

Parameters

obj_type (input)
A member of the enumerated list of object types. The enumeration is supplied in the <milib.h> header file.

obj_name (input)
A null terminated string specifying the name of the object.

lib_name (input)
A null terminated string specifying the name of the library where the object is stored. You can specify either a specific name for the library, or the character string "*LIBL" (or an empty string), which indicates that the current library list is to be searched to find the library where the object is stored.

auth (input)
Constructed from supplied bit mask macros in the <milib.h> header file. Programs executing in user-domain may not assign authority in the resulting system pointer. The value in auth is ignored; authority is set to the not set state. Otherwise, the object authority states are set as specified by auth.

Notes on Usage

- The function will first resolve to the library. If successful, a resolve to the object is made.
- The builtin functions _RSLVSP1 through _RSLVSP8 also provide access to the MI instruction.
Example
This example illustrates the use of the rslvsp function.

```c
#include <QSYSINC/MIH/RLSVSP>
#include <QSYSINC/MIH/MATPTR>
#include <stdio.h>

int main(void) {
    _SYSPTR          pgm_ptr;
    _MPTR_Template_T mpt;

    /* Resolve to program MYPGM in the current library list */
    pgm_ptr = rslvsp(_Program, "MYPGM", "*LIBL", _AUTH_OBJ_MGMT);

    /* Materialize the program pointer */
    mpt.Obj_Ptr.Template_Size = sizeof(_OBJPTR_T);
    matptr(&mpt, pgm_ptr);
    printf("Object name : %.10s\n", mpt.Obj_Ptr.Object_ID.Name);
    printf("Library     : %.10s\n", mpt.Obj_Ptr.Library_ID.Name);
    printf("Type        : %d\n", mpt.Obj_Ptr.Ptr_Type);
    printf("Authority   : %d\n", mpt.Obj_Ptr.Auth_Or_Off.Ptr_Authorization);
}
```

Output
Object name : MYPGM
Library     : QTEMP
Type        : 1
Authority   : 0
Retrieve Computational Attributes (RETCA)

Format
#include <QSYSINC/MIH/RETCA>

unsigned int retca (unsigned int mask);

Description
The retca builtin retrieves from the machine and returns the 4-byte value containing the selected computational attributes.

Parameters
mask (input)
Selection mask specifying which floating-point computational attributes are to be retrieved from the machine. The mask, which must be a literal, is constructed by OR’ing together a combination of the following least significant bits of the 4-byte mask.

<table>
<thead>
<tr>
<th>Mask Bit</th>
<th>Portion of Computational attributes value to load from the machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>x’08’</td>
<td>Load the Exception Mask byte</td>
</tr>
<tr>
<td>x’04’</td>
<td>Reserved (binary 0)</td>
</tr>
<tr>
<td>x’02’</td>
<td>Load the Exception Occurrence byte</td>
</tr>
<tr>
<td>x’01’</td>
<td>Load the Rounding Mode byte</td>
</tr>
</tbody>
</table>

For the format of the computational attributes returned by this builtin see the layout of the computation attribute bytes in Figure 1 on page 209.
The format of the 4-byte computational attributes is:

```
00 31
```

<table>
<thead>
<tr>
<th>byte 0</th>
<th>byte 1</th>
<th>byte 2</th>
<th>byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>07 00</td>
<td>07 00</td>
<td>07 00</td>
</tr>
<tr>
<td>low address</td>
<td>high address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Byte 0**: Exception Mask
  - 0 = disabled (exception is masked)
  - 1 = enabled (exception is unmasked)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>2</td>
<td>Floating-point Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Floating-point Underflow</td>
</tr>
<tr>
<td>4</td>
<td>Floating-point Zero Divide</td>
</tr>
<tr>
<td>5</td>
<td>Floating-point Inexact Result</td>
</tr>
<tr>
<td>6</td>
<td>Floating-point Invalid Operand</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

- **Byte 1**: Reserved (binary zero)

- **Byte 2**: Exception Occurrence
  - 0 = exception has not occurred
  - 1 = exception has occurred

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>2</td>
<td>Floating-point Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Floating-point Underflow</td>
</tr>
<tr>
<td>4</td>
<td>Floating-point Zero Divide</td>
</tr>
<tr>
<td>5</td>
<td>Floating-point Inexact Result</td>
</tr>
<tr>
<td>6</td>
<td>Floating-point Invalid Operand</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

- **Byte 3**: Computational Mode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>1-2</td>
<td>Rounding Mode</td>
</tr>
<tr>
<td>00</td>
<td>Round towards positive infinity</td>
</tr>
<tr>
<td>01</td>
<td>Round towards negative infinity</td>
</tr>
<tr>
<td>10</td>
<td>Round towards zero</td>
</tr>
<tr>
<td>11</td>
<td>Round to nearest</td>
</tr>
<tr>
<td>3-7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

*Figure 1. Layout of the computational attribute bytes*

**Notes on Usage**
- `retca` is available as a macro only.
Example
This example illustrates the use of the retca macro.

```c
#include <QSYSINC/MIH/RETCMA>
#include <stdio.h>

#define POSITIVE_INFINITY 0x00
#define NEGATIVE_INFINITY 0x20
#define ZERO 0x40
#define NEAREST 0x60

int main(void) {
    unsigned rounding_mode;
    rounding_mode = retca(_SRC_RA_ROUNDING_MODE);
    switch (rounding_mode) {
        case POSITIVE_INFINITY:
            printf("The current setting is round towards positive infinity\n");
            break;
        case NEGATIVE_INFINITY:
            printf("The current setting is round towards negative infinity\n");
            break;
        case ZERO:
            printf("The current setting is round towards zero\n");
            break;
        case NEAREST:
            printf("The current setting is round towards nearest\n");
            break;
        default:
            printf("Error: unrecognized setting\n");
    }
}
```

Output
The current setting is round towards nearest
Remove Independent Index Entry (RMVINXEN)

Format
#include <QSYSINC/MIH/RMVINXEN>

_SPCPTR rmvinxen (_SPCPTR receiver,
    _SYSPTR index_obj,
    _IIX_Opt_List_T *option_list,
    _SPCPTR search_arg);

Description
The rmvinxen function removes the specified index entries from the independent index and returns these in the receiver. A pointer to the receiver is returned by the function.

Parameters
receiver (input/output)
    Pointer to the buffer to receive the removed entry or entries. A NULL is not supported for this parameter.

index_obj (input)
    Pointer to the independent index object from which the entry or entries are to be removed.

option_list (input)
    Pointer to the option list template. This template contains additional information on the entry or entries to be removed.

search_arg (input)
    Pointer to the search argument(s).

Notes on Usage
- The builtin functions _RMVINXEN1 and _RMVINXEN2 also provide access to the MI instruction.
Example
This example illustrates the use of the rmvinxen function.

```c
/*---------------------------------------------*/
/* */
/* Example Group: User Indexes <QSYSINC/MIH/RMVINXEN> */
/* */
/* Function: rmvinxen (Remove Index Entry) */
/* */
/* Description: This example illustrates how the 'rmvinxen' */
/* function can be used to remove a particular */
/* entry from a user index. The entry being */
/* removed, will be found using the customer */
/* number key. Upon removal from the user index, */
/* the entry will be displayed to the screen. */
/* */
/* */

#include <QSYSINC/MIH/RMVINXEN>
#include <QSYSINC/MIH/RLVSP>
#include <stdio.h>

typedef struct Customer_Entry { /* The format/layout of each */
    int customer_number; /* entry on the user index. */
    char last_name[50];
    char first_name[40];
    char phone_number[9]; /* Using phone format: 555-5555 */
} Customer_Entry;

_IIX_Opt_List_T remove_option; /* Template used for setting */
/* options for the "remove". */
/* Also used as the receiver */
/* for any returned data. */

_SYSPTR index_ptr; /* Pointer to the user index. */

Customer_Entry removed_customer; /* Buffer to receive any entry */
/* that matches our criteria. */

int which_customer /* Our criterion is to remove */
    = 9999999; /* the entry that has this */
    /* customer number. */

int main(void) {
    /* Get a pointer to the *USRIDX */
    index_ptr = rlvsp(_Usridx, "MYUSRIDX", "MYLIB", _AUTH_ALL);

    remove_option.Rule = _FIND_EQUALS; /* Find a customer number that */
    /* matches the specified one. */

    remove_option.Arg_Length = sizeof(int); /* The customer search "key" */
    /* is an integer. */
```
/* Maximum number of entries */
remove_option.Occ_Count = 1;
/* to return that match the */
/* criterion. In this case, */
/* we want to find the one */
/* unique entry that has the */
/* particular customer number. */

/*---------------------------------------------------------------*/
/* Remove the entry for this particular customer if it exists in the */
/* user index. If the entry was successfully removed, output the */
/* entry to the screen. */
/*---------------------------------------------------------------*/

rmvinxen( &removed_customer, index_ptr, &remove_option, 
            &which_customer);

if ( remove_option.Ret_Count == 0 ) {
    printf("Could not remove the entry for customer number: %d \n", 
            which_customer );
    printf("since the entry could not be found in the user index.\n");
}
else { /* an entry was found and removed */
    printf("The removed entry for customer number %d was:\n", 
            which_customer );
    printf("Customer Name: %s %s \n", removed_customer.first_name, 
            removed_customer.last_name  );
    printf("Phone Number: %s \n", removed_customer.phone_number );
}
}

Output
The removed entry for customer number 9999999 was:
Customer Name:  John Smith
Phone Number:   555-5555
Scan with Control (SCANWC)

**Format**
```
#include <QSYSINC/MIH/SCANWC>

int scanwc (char *base-locator,
            _SCWC_Control_T *controls,
            char options);
```

**Description**
The scanwc function scans a base string of single or double-byte characters for occurrences of a character value satisfying the criteria in the controls and options parameters.

The scanwc function returns -1 if the scan is unsuccessful. Otherwise, scanwc returns a value which is the offset of the character which terminated the scan relative to the base string.

On return from the scan, the base locator is still pointing to the first character in the base string. This behavior is different from the SCANWC MI which updates the base locator.

**Parameters**
- **source (input)**
  - Pointer to the base string to scan.
- **controls (input/output)**
  - Pointer to the controls template which specifies additional information to be used to control the scan.
- **options (input)**
  - The option indicators.

**Notes on Usage**
- This is a compatibility function to provide the same semantics as the SCANWC MI instruction.
- There is no equivalent to the escape target operand of the SCANWC MI provided on this interface.

**Exceptions**
If arguments 1 and 2 do not point to valid storage locations, an MCH3601 or an MCH0601 exception will be signalled.

If any of the input parameters are not valid, an MCH3601 or MCH0601 may be signalled.

Other exceptions possible from this function are:
- MCH0602 - Boundary alignment
- MCH0801 - Parameter Reference Violation
- MCH3402 - Object Destroyed
- MCH3602 - Pointer Type Invalid
- MCH6801 - Object Domain Violation
Example
This example illustrates the use of the scanwc function.

```c
/* Example Group: Computation and Branching <QSYSINC/MIH/SCANWC> */
/* Function: scanwc (Scan With Control) */
/* Description: This example uses 'scanwc' to scan the single- */
/* byte base string for a character to which 'k' is 'greater than or equal'. */
/* */
/******************************/

#include <QSYSINC/MIH/SCANWC>
#include <string.h>
#include <stdio.h>

int main(void) {

    char base[] = "This is the base string";
    _SCWC_Control_T t;
    int offset;
    char options;

    memset(&t, 0, sizeof(t));
    t.Start_Scan = 1;
    t.Comp_Char[1] = 'k';
    t.Base_Length = sizeof(base);

    offset = scanwc(base, &t, _SCWC_NONMIXED_GREATER | _SCWC_NONMIXED_EQUAL);

    printf("The scan was stopped at character %c, offset %d\n", base[offset], offset);
}
```

Output
The scan was stopped at character h, offset 1
Set Access State (SETACST)

**Format**

```c
#include <QSYSINC/MIH/SETACST>

void setacst (_ANYPTR object,
               char state_code,
               int pool_id,
               int space_length);
```

**Description**

The setacst function specifies the access state (which specifies the desired speed of access) that the issuing process has for a set of objects or sub-object elements in the execution interval following the execution of the function.

The specification of an access state for an object momentarily preempts the machine's normal management of an object.

**Parameters**

- **object (input)**
  - The pointer to object.

- **state_code (output)**
  - The access state code.

- **pool_id (input)**
  - The access pool ID.

- **space_length (input)**
  - The space length.

**Notes on Usage**

- The setacst function performs a single object Set Access State only. You must use the _SETACST builtin to perform a multiple object Set Access State.

- For access states 0x10 and 0x18, the operational object size is returned through the template. The template is used on the builtin version only. For this reason, it is recommended that the _SETACST builtin be used for access states 0x10 and 0x18.

- A macro version is available.
Example
This example illustrates the use of the setacst function.

```c
#include <QSYSINC/MIH/SETACST>
#include <QSYSINC/MIH/RSLVSP>
#include <stdio.h>

_SYSPTR some_object; /* The object that is being put into main memory. */
int pool, /* Main memory storage pool in which to put the object. */
size; /* Number of bytes of the object to be moved into main memory. */

int main(void) {
    /* Get a pointer to the object. */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Have the first 64K bytes (arbitrary value for this example) of the object brought into main memory asynchronously (moved in by a "background" task, so this process/job is not necessarily suspended - unlike a synchronous "bring"). */
    size = 65536; /* 0 specifies to use the pool */
    pool = 0; /* in which the job running this program is using. */

    setacst( some_object, _SETACSTASYNCH_REQ, pool, size );
}
```
setacst

Output

** no screen output **
Set Bit in String (SETBTS)

**Format**

```c
#include <QSYSINC/MIH/SETBTS>

void setbts (_SPCPTP bit_string,
             unsigned int bit_offset);
```

**Description**
The setbts function sets `bit_string` as indicated by `bit_offset`.

**Parameters**

- **bit_string** (input)
  A pointer to `bit_string` with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

- **bit_offset** (input)
  Indicates which bit of `bit_string` is to be set, with an offset of zero indicating the leftmost bit of the leftmost byte of `bit_string`. This value must be less than 64k.

**Notes on Usage**

- If the selected bit is beyond the end of the string, or the value of `bit_offset` is greater than or equal to 64k, the result of the operation is undefined.

- A macro version is available.

- The builtin function _SETBTS also provides access to the MI instruction.

**Exceptions**

In some circumstances, if the selected bit is beyond the end of allocated storage, an MCH3203 exception may be signalled.
**Example**

This example illustrates the use of the `setbts` function.

```c
#include <QSYSINC/MIH/SETBTS>
#include <QSYSINC/MIH/TSTBTS>
#include <QSYSINC/MIH/CCLRBT>
#include <stdio.h>

#define FALSE 0
#define TRUE !FALSE

int main(void) {
    unsigned bit_string = 0;
    unsigned offset = 17;
    unsigned flag = FALSE;

    setbts(&bit_string, offset);

    flag = tstbts(&bit_string, offset);
    if (flag)
        printf("The %u'th bit has been set\n", offset);

    clrbts(&bit_string, offset);
    flag = tstbts(&bit_string, offset);
    if (!flag)
        printf("The %u'th bit has been cleared\n", offset);
}
```

**Output**

The 17'th bit has been set
The 17'th bit has been cleared
Set Computational Attributes (SETCA)

**Format**
#include <QSYSINC/MIH/SETCA>

    void setca (unsigned int new_value,
                unsigned int mask);

**Description**
The setca builtin sets the computational attributes selected and stores them into the machine.

**Parameters**

*new_value*(input)
The value which is to be used to update the machine computational attributes.

*mask*(input)
Selection mask specifying which floating-point computational attributes are to be set in the machine. The mask, which must be a literal, is constructed by ORing together a combination of the following least significant bits of the 4-byte mask.

<table>
<thead>
<tr>
<th>Mask Bit</th>
<th>Portion of Computational attributes value to store in the machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>x'08'</td>
<td>Store the Exception Mask byte</td>
</tr>
<tr>
<td>x'04'</td>
<td>Reserved (binary 0)</td>
</tr>
<tr>
<td>x'02'</td>
<td>Store the Exception Occurrence byte</td>
</tr>
<tr>
<td>x'01'</td>
<td>Store the Rounding Mode byte</td>
</tr>
</tbody>
</table>

For the format of the computational attributes returned by this builtin see the layout of the computation attribute bytes in Figure 2 on page 222.
The format of the 4-byte computational attributes is:

```
00 31
```

<table>
<thead>
<tr>
<th>byte 0</th>
<th>byte 1</th>
<th>byte 2</th>
<th>byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>07 00</td>
<td>07 00</td>
<td>07 00</td>
</tr>
</tbody>
</table>

low address       high address

- **Byte 0**: Exception Mask
  - 0 = disabled (exception is masked)
  - 1 = enabled (exception is unmasked)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>2</td>
<td>Floating-point Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Floating-point Underflow</td>
</tr>
<tr>
<td>4</td>
<td>Floating-point Zero Divide</td>
</tr>
<tr>
<td>5</td>
<td>Floating-point Inexact Result</td>
</tr>
<tr>
<td>6</td>
<td>Floating-point Invalid Operand</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

- **Byte 1**: Reserved (binary zero)

- **Byte 2**: Exception Occurrence
  - 0 = exception has not occurred
  - 1 = exception has occurred

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>2</td>
<td>Floating-point Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Floating-point Underflow</td>
</tr>
<tr>
<td>4</td>
<td>Floating-point Zero Divide</td>
</tr>
<tr>
<td>5</td>
<td>Floating-point Inexact Result</td>
</tr>
<tr>
<td>6</td>
<td>Floating-point Invalid Operand</td>
</tr>
<tr>
<td>7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

- **Byte 3**: Computational Mode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved (binary zero)</td>
</tr>
<tr>
<td>1-2</td>
<td>Rounding Mode</td>
</tr>
<tr>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td>3-7</td>
<td>Reserved (binary zero)</td>
</tr>
</tbody>
</table>

*Figure 2. Computational attributes*

**Notes on Usage**

- setca is available as a macro only.
- When setca is used to change the computational attributes, it is the programmer's responsibility to save the prior attributes and to restore these on abnormal/normal termination.
Example
This example illustrates the use of the setca macro.

```c
/*---------------------------------------------*/
/* */
/* Example Group: Computation and Branching <SYSINC/MIH/SETCA> */
/* */
/* Function: setca (Set Computational Attributes) */
/* */
/* Description: This example uses 'setca' to temporarily mask */
/* the floating point overflow, underflow, and */
/* zero divide exceptions. Please Note: it is the */
/* responsibility of the programmer to save and */
/* restore any prior attribute settings. */
/* */
/*---------------------------------------------*/

#include <SYSINC/MIH/SETCA>
#include <SYSINC/MIH/RETCA>
#include <except.h>

#define NEW_ATTRIBUTES 0x02000000 /* Mask the following exceptions: */
/* floating point overflow */
/* floating point underflow */
/* floating point zero divide */
/* floating point inexact result*/

static void cancel_handler (_CNL_HndlrParms_T *parms) {

    /* Restore the saved exception mask attributes passed to the */
    /* handler through the communications area. */
    setca (*(unsigned *)parms->Com_Area, _SRCA_EXCEPTION_MASK);
}

int main(void) {

    volatile unsigned int old_attributes;

    /* Save the prior exception mask attributes. */
    old_attributes = retca(_SRCA_EXCEPTION_MASK);

    /* Set the new exception mask attributes. */
    setca(NEW_ATTRIBUTES, _SRCA_EXCEPTION_MASK);

    /* Set up a cancel handler to restore prior exception mask */
    /* attributes in the event of a cancellation. */
    #pragma cancel_handler(cancel_handler, old_attributes)

    /* Floating point computations using new attributes would go here...*/

    /* Restore the saved exception mask attributes. */
```
setca

    setca(old_attributes, _SRCA_EXCEPTION_MASK);
}

**Output**

**no screen output**
Set System Pointer from Pointer (SETSPFP)

Format

```c
#include <QSYSINC/MIH/SETSPFP>

_SYSPTR setsfp (_ANYPTR pointer);
```

Description

The setsfp function returns a system pointer to the system object addressed by `pointer`.

If `pointer` is a system pointer, then a system pointer addressing the same object is returned containing the same authority as the input pointer.

If `pointer` is a space pointer, then a system pointer addressing the system object that contains the associated space addressed by `pointer` is returned.

Parameters

`pointer (input)`

A space pointer or system pointer.

Notes on Usage

- A macro version is available.
- The builtin function _SETSPFP also provides access to the MI instruction.

Exceptions

If the pointer argument is not set, an MCH3601 is signalled. If the pointer argument is not a space pointer, system pointer, or an open pointer containing a space pointer or system pointer, an MCH3602, and an MCH3601 are signalled.
Example
This example illustrates the use of the setspfp function.

```c
#include <QSYSINC/MIH/SETSPFP>
#include <QSYSINC/MIH/RLYSP>
#include <stdio.h>

#define CREATION_SIZE 65536

int main(void) {
  _SPCPTR myspace_spp;
  _SYSPTR myspace_sysp, sysp;
  int error_code = 0;

  QUSCRIPTUS("MYSPACE QTEMP ", /* Create user space */
              "MYSPACE ",
              CREATION_SIZE,
              ",0",
              "\*ALL",
              "MYSPACE example for Programmer's Reference",
              "\*YES",
              &error_code);
  QUSPTRUS("MYSPACE QTEMP ", /* Retrieve pointer to user space */
              &myspace_spp);

  /* Case 1: source pointer is a space pointer */
  myspace_sysp = rlsvsp(_Usrsps, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);
  sysp = setspfp(myspace_spp);
  if (sysp != myspace_sysp)
    printf("Case 1: the pointers are not equal\n");
  else
    printf("Case 1: the pointers are equal\n");

  /* Case 2: source pointer is a system pointer */
  sysp = setspfp(myspace_sysp);
  if (sysp != myspace_sysp)
    printf("Case 2: the pointers are not equal\n");
  else
```
printf("Case 2 : the pointers are equal\n");
}

**Output**
Case 1: the pointers are equal
Case 2: the pointers are equal
Set Space Pointer from Pointer (SETSPFP)

Format

```
#include <QSYSINC/MIH/SETSPFP>

_SPCPTR setsppfp (_ANYPTR pointer);
```

Description

The setsppfp function returns the address of a space object from the source pointer specified.

If the source pointer is a space pointer, the pointer returned is set to the address of the leftmost byte of the storage location addressed by the source pointer.

If the source pointer is a system pointer, the pointer returned is set to address the first byte of the space contained in the system object addressed by the system pointer.

Parameters

pointer (input)

A space pointer or system pointer.

Notes on Usage

- A macro version is available.
- The builtin function _SETSPFP also provides access to the MI instruction.

Exceptions

If the source pointer is not set, an MCH3601 is signalled. If the pointer is not a space pointer, system pointer, or an open pointer containing a space pointer or system pointer, an MCH3602 is signalled.
Example
This example illustrates the use of the setsppfp function.

```c
/* Example Group: Space Object Addressing  <QSYSINC/MIH/SETSPFP> */
/* Function: SETSPFP (Set Space Pointer from Pointer) */
/* Description: This example uses 'setsppfp' to obtain a pointer */
/* to the leftmost byte of the space addressed by */
/* both a space pointer and a system pointer. */
/*****************************************************************************/

#include <QSYSINC/MIH/SETSPFP>
#include <QSYSINC/MIH/RSLVSP>
#include <string.h>
#include <stdio.h>

#include <QSYSINC/H/QUSCRTUS>
#include <QSYSINC/H/QUSPTRUS>

#define CREATION_SIZE  65536

int main(void) {
    _SPCPTTR  myspace_spp, sp;
    _SYSPTR   myspace_sysp;
    int       error_code = 0;

    QUSCRTUS("MYSPACE  QTEMP  ", /* Create user space */
             "MYSPACE  " ,
             CREATION_SIZE,
             "\0",
             "ALL",
             "MYSPACE example for Programmer's Reference ",
             "YES",
             &error_code);

    QUSPTRUS("MYSPACE  QTEMP  ", /* Retrieve pointer to user space */
                &myspace_spp);

    /* Case 1: source pointer is a space pointer */
    sp = setsppfp(myspace_spp);
    if (sp != myspace_spp)
        printf("Case 1: the pointers are not equal\n");
    else
        printf("Case 1: the pointers are equal\n");

    /* Case 2: source pointer is a system pointer */
    myspace_sysp = rslvsp(_Usrspc, "MYSPACE", "QTEMP", _AUTH_OBJ_MGMT);
    sp = setsppfp(myspace_sysp);
    if (sp != myspace_spp)
        printf("Case 2: the pointers are not equal\n");
    else
        printf("Case 2: the pointers are equal\n");
}
```

```c
printf("Case 2 : the pointers are not equal\n");
else
    printf("Case 2 : the pointers are equal\n");
}

Output
Case 1 : the pointers are equal
Case 2 : the pointers are equal
```
Set Space Pointer Offset (SETSPPO)

Format
#include <QSYSINC/MIH/SETSPPO>

_SPCPTR setsppo (_SPCPTR pointer,
    int offset);

Description
The setsppo function takes the value of offset and assigns it to the offset portion
pointer. The resulting pointer is returned by the function.

Parameters
pointer (input)
    The space pointer whose offset is to be set.

offset (input)
    The space pointer offset value.

Notes on Usage
- The resulting space pointer continues to address the same space object.
- pointer is left unchanged.

Exception
If the pointer argument is not set, or the offset argument is not valid (too large, or
negative), an MCH3601 is signalled.
Example
This example illustrates the use of the setsppo function.

```c
#include <stdio.h>
#include <QSYSINC/MIH/SETSPPO>
#include <QSYSINC/MIH/STSPPO>

int main(void) {  
    int buffer[50];
    _SPCPTTR sp1 = buffer + 25, sp2 = buffer;
    sp2 = setsppo(sp2, stsppo(sp1));
    if (sp1 != sp2)  
        printf("Error: the pointers are not equal\n");
    else  
        printf("The pointers are equal\n");
}

Output
The pointers are equal
Store Space Pointer Offset (STSPPO)

Format
#include <QSYINC/MIH/STSPPO>

int stsppo (_SPCPtr pointer);

Description
The stsppo function returns the offset value of pointer.

Parameters
pointer (input)
    Space pointer to extract offset from

Exceptions
If the pointer argument is not set, an MCH3601 is signalled.

Example
This example illustrates the use of the stsppo function.

```c
#include <stdio.h>
#include <QSYINC/MIH/STSPPO>

#define OFFSET 25

int main(void) {
    int buffer[50], offset1, offset2;

    _SPCPtr sp1 = buffer + OFFSET, sp2 = buffer;

    offset1 = stsppo(sp1);
    offset2 = stsppo(sp2);
    printf("offset1 - offset2 = %d bytes\n", offset1 - offset2);
}
```

Output
offset1 - offset2 = 100 bytes
Test Authority (TESTAU)

Format
#include <QSYSINC/MIH/TESTAU>
short testau (_SYSPTR object,
short offset,
short req_auth);

Description
The testau function returns information on the object authorities and/or ownership
rights currently available to the process for the object specified. The operation is
performed relative to the invocation whose offset is specified.

Parameters
object (input)
   Pointer to the system object for which authority is to be tested.

offset (input)
   Identifies an invocation relative to the current at which the authority verification
   is to be performed.

req_auth (input)
   Indicates the required authority and/or ownership rights to be tested and
   returned by the function (if currently available to the process).

Notes on Usage
• The builtin function _TESTAU1 or _TESTAU2 also provides access to the MI
  instruction.

Example
This example illustrates the use of the testau function.

/******************************************************************************
/* Example Group: Authorizations <QSYSINC/MIH/TESTAU> */
/* Function: testau (Test Authorization) */
/* Description: This example shows how 'testau' can be used to */
/* determine what kind of authority the job in which */
/* this program runs in, has to some object. This */
/* example determines whether the job can delete or */
/* update a particular program object (QCMD *PGM */
/* in QSYS). */
/******************************************************************************
#include <QSYSINC/MIH/TESTAU>
#include <QSYSINC/MIH/RLSVSP>
#include <stdio.h>
 _SYSPTR some_object; /* Pointer to the object that */
 /* will be tested for certain */
 /* authority. */
short test_authority, /* Authority being checked. */
return_authority, /* Actual authority this job */
/* has to the object. */

relative_invocation;

int main(void) {
    /* Get a pointer to the object */
    /* that is being tested. */
    some_object = rslvsp(_Program, "QCMD", "QSYS", _AUTH_NONE);

    /* 'testau' will be used to check if the job that is running this */
    /* program has both delete and update authority to the object. */
    */

    test_authority = _AUTH_DELETE | _AUTH_UPDATE;
    relative_invocation = 0; /* Use the current invocation. */

    return_authority = testau( some_object, relative_invocation,
                              test_authority );

    /* If the actual authority does not match the authority we were */
    /* testing for, then output this information to the screen. */
    */

    if ( return_authority != test_authority ) {
        printf("This job/process does not have Delete or Update\n");
        printf("authority to the QSYS/QCMD program. \n");
    }
}

Output
This job/process does not have Delete or Update authority to the QSYS/QCMD program.
Trim Length (TRIML)

**Format**

```
#include <QSYSINC/MIH/TRML>

int triml (char *string,
           char trim_char);
```

**Description**

The `triml` function returns the length of `string` after `trim_char` has been trimmed from the end.

`trim_char` is trimmed from the end of `string` as follows: if the rightmost (last) character of `string` is equal to the character specified by `trim_char`, the length of the trimmed `string` is reduced by 1. This operation continues until the rightmost character is no longer equal to `trim_char` or the trimmed length is zero.

`string` is not changed by this function.

**Parameters**

- **string (input)**
  - The source string.

- **trim_char (input)**
  - The trim character.

**Notes on Usage**

- The `triml` function operates on null-terminated strings.
- This is a compatibility function to provide the same semantics as the TRIML MI instruction.

**Exceptions**

If invalid input values are passed to the function, an MCH3601 and MCH0601 exception may be signalled.

Other exceptions possible from this function are:

- MCH0602 - Boundary alignment
- MCH0801 - Parameter Reference Violation
- MCH3402 - Object Destroyed
- MCH3602 - Pointer Type Invalid
- MCH6801 - Object Domain Violation
Example
This example illustrates the use of the triml function.

```c
#include <QSYSINC/MIH/TRIML>
#include <string.h>
#include <stdio.h>

int main(void) {
    char trim_str[] = "String with lots of punctuation!!!!!!!!!!";
    int trim_len;

    trim_len = triml(trim_str, '!');
    printf("The length of the untrimmed string is \%d\n", strlen(trim_str));
    printf("The length of the trimmed string is \%d\n", trim_len);
}
```

Output
The length of the untrimmed string is 42
The length of the trimmed string is 31
**Test Bit in String (TSTBTS)**

**Format**

```c
#include <QSYSINC/MIH/TSTBTS>

int tstbts (_SPCPTR bit_string, 
            unsigned int bit_offset);
```

**Description**

The `tstbts` function tests the bit in `bit_string`, as indicated by `bit_offset`, to determine if the bit is set or not set. The `tstbts` function returns a nonzero value if the selected bit is set; otherwise zero is returned.

**Parameters**

- `bit_string` (input)
  - Pointer to `bit_string` with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

- `bit_offset` (input)
  - Indicates which bit of `bit_string` is to be tested with an offset of 0 indicating the leftmost bit of the leftmost byte of `bit_string`. This value must be less than 64k.

**Notes on Usage**

- If the selected bit is beyond the end of the string, or the value of `bit_offset` is greater than or equal to 64k, the result of the operation is undefined.

- A macro version is available.

- The builtin function _TSTBTS also provides access to the MI instruction.

**Exceptions**

In some circumstances, if the selected bit is beyond the end of allocated storage, an MCH3203 exception may be signalled.
**Example**

This example illustrates the use of the `tstbts` function.

```c
#include <QSYSINC/MIH/TSTBTS>
#include <QSYSINC/MIH/SETBTS>
#include <QSYSINC/MIH/CLR BTS>
#include <stdio.h>

#define FALSE 0
#define TRUE !FALSE

int main(void) {

  unsigned bit_string = 0;
  unsigned offset = 17;
  unsigned flag = FALSE;

  setbts(&bit_string, offset);
  flag = tstbts(&bit_string, offset);
  if (flag)
    printf("The %u'th bit has been set\n", offset);

  clrbts(&bit_string, offset);
  flag = tstbts(&bit_string, offset);
  if (!flag)
    printf("The %u'th bit has been cleared\n", offset);
}
```

**Output**

The 17'th bit has been set
The 17'th bit has been cleared
Unlock Object (UNLOCK)

Format
```
#include <QSYSINC/MIH/UNLOCK>

void unlock (_SYSPTR object, 
            char unlock_option, 
            char lock_state);
```

Description
The unlock function releases a single lock for the system object specified.

Parameters
**object (input)**
Pointer to the system object to be unlocked.

**unlock_option (input)**
The unlock option specifies if locks are to be released or outstanding lock requests are to be cancelled.

**lock_state (input)**
The lock state to unlock.

Notes on Usage
- The lock entry will always be marked as active by the unlock function. So, for the function version it is not necessary to OR the lock state with _LOCK_ENTRY_ACTIVE.
- The builtin function _UNLOCK and macro Unlock also provide access to the MI instruction.
Example
This example illustrates the use of the unlock function.

```c
/* Example Group:   Locks  <QSYSINC/MIH/UNLOCK> */
/* Function:       unlock  (Unlock Object) */
/* Description:    This example illustrates the use of the 'unlock' function. The 'lock'
                function will first be used to place a Shared Read (LSRD) lock on the object
                (a user queue in this example). The CL DSPJOB command will then be used to display
                all of the locks held by the job/process that runs this program. After using the
                'unlock' function to remove the lock, DSPJOB is used again to allow the user to
                verify that the lock was removed. */

#include <QSYSINC/MIH/UNLOCK>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/MIH/LOCK>
#include <stdio.h>
#include <stdlib.h>

_SYSPTR    some_object;          /* The object being locked. */
_MI_Time   timeout;             /* Amount of time to wait for the lock to be placed. */

int hours    = 0,              /* Time components used to create an _MI_Time value */
        minutes = 0,          /* that can be passed to the 'lock' function. */
        seconds = 20,         /* 'lock' function. */
        hundredths = 0;

int main(void) {

    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Format an 'mitime' value to represent the timeout to be used by
    the 'lock' function when trying to obtain a Shared-Read (LSRD) lock
    on the object. If the lock request cannot be satisfied in the
    specified amount of time, then an exception will be raised. The
    CL DSPJOB command is called to display all of the object locks
    owned by the job/process that runs this program. */

    mitime( &timeout, hours, minutes, seconds, hundredths );
    lock( some_object, timeout, _LSRD_LOCK );
    system( "DSPJOB OPTION(*JOBLCNK)" );

    return 0;
```
unlock( some_object, _UNLOCK_SPECIFIC, _LSRD_LOCK );

system( "DSPJOB OPTION(+JOBLCK)" );

**Output**

Display Job Locks
System: TORAS4YL

Job: OMXNA3S4 User: VISCA Number: 009616

Job status: ACTIVE

Type options, press Enter.
5-Display job member locks

<table>
<thead>
<tr>
<th>Opt</th>
<th>Object</th>
<th>Library</th>
<th>Type</th>
<th>Lock</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>QSYS</td>
<td>*MENU</td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>MYUSRQ</td>
<td>MYLIB</td>
<td>*USRQ</td>
<td>*SHRRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>OMXNA3S4</td>
<td>QSYS</td>
<td>*DEVD</td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>QADM</td>
<td>QSYS</td>
<td>*LIB</td>
<td>*SHRRD</td>
<td>HELD</td>
<td></td>
</tr>
</tbody>
</table>

F3=Exit F5=Refresh F10=Display job record locks F12=Cancel

More...

Display Job Locks
System: TORAS4YL

Job: OMXNA3S4 User: VISCA Number: 009616

Job status: ACTIVE

Type options, press Enter.
5-Display job member locks

<table>
<thead>
<tr>
<th>Opt</th>
<th>Object</th>
<th>Library</th>
<th>Type</th>
<th>Lock</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>QSYS</td>
<td>*MENU</td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>OMXNA3S4</td>
<td>QSYS</td>
<td>*DEVD</td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>QADM</td>
<td>QSYS</td>
<td>*LIB</td>
<td>*SHRRD</td>
<td>HELD</td>
<td></td>
</tr>
</tbody>
</table>

F3=Exit F5=Refresh F10=Display job record locks F12=Cancel

More...
Unlock Space Location (UNLOCKSL)

**Format**

```c
#include <QSYSINC/MIH/UNLOCKSL>

void unlocksl (_SPCPT location,
               char lock_state);
```

**Description**
The `unlocksl` function removes a single lock type from the space location.

**Parameters**

- **location (input)**
  Pointer to the space location to be unlocked.

- **lock_state (input)**
  The lock state to unlock.

**Notes on Usage**
- The builtin functions _UNLOCKSL1 and _UNLOCKSL2 also provide access to the MI instruction.
- The _UNLOCKSL2 builtin supports multiple space location unlock requests.
unlocksl

Example
This example illustrates the use of the unlocksl function.

```c
#include <QSYSINC/MIH/UNLOCKSL>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/MIH/LOCKSL>
#include <QSYSINC/MIH/SETSPPFP>
#include <QSYSINC/MIH/MATOBJLK>
#include <stdio.h>

(SYSPTR some_object; /* The object from which some */
 /* byte (space) location will */
 /* locked. */

_SPCPTR some_byte; /* The pointer to the location */
 /* that is being locked. */

葫MOBJL_Template_T allocated_locks; /* The receiver template used */
 /* by 'matobjlk'. */

int main(void) {
    /* Get a pointer to the object */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Set a space pointer from */
    /* the system pointer. */
    some_byte = setsppfp( some_object );

    locksl( some_byte, _LENR_LOCK );

    /* Request the Exclusive-No-Read (LENR) lock on the space location. */
    /* A default timeout called the "Process Default Timeout" is used as */
    /* the timeout value. If the lock request is not satisfied in that */
    /* time, an exception is raised. */

    return 0;
}
```
unlocksl

/*-------------------------------------------------------------*/
/* In order to verify that the Exclusive lock was placed on the space */
/* location (byte), we can look at the bit pattern returned by */
/* 'matobjlk' in the 'Lock_Alloc' field of the template. The fifth */
/* bit (bit 4) of this field represents the Lock-Exclusive-No-Read */
/* (LENR) lock. By using the bitwise AND (&) operator, we can */
/* determine if the bit is set by ANDing it with the provided bit */
/* mask, _LENR_LOCK (which happens to be defined in <milock.h> as */
/* hex 08 which is bit pattern 0000 1000). */
/*-------------------------------------------------------------*/

allocated_locks.Template_Size = sizeof(_MOBJL_Template_T);

matobjlk( &allocated_locks, some_byte );

if ( allocated_locks.Lock_Alloc & _LENR_LOCK ) {
    printf("There is an Exclusive-No-Read lock on the space \n");
    printf("location with address: %p \n\n", some_byte );

    /* For illustration purposes, we will now remove the lock we */
    /* just placed on the space location and use 'matobjlk' again */
    /* to verify that it gets removed. */

    printf("The lock will now be removed... \n");

    unlocksl( some_byte, _LENR_LOCK );
    matobjlk( &allocated_locks, some_byte );

    /* If the LERN lock is still set */
    if ( allocated_locks.Lock_Alloc & _LENR_LOCK )
        printf("Error. The LERN lock still exists.\n");
    else
        printf("The LERN lock was removed.\n");
}

Output
There is an Exclusive-No-Read lock on the space
location with address: SPP:0401MYLIB :0a02MYUSRQ :0:0:d35
The lock will now be removed...
The LERN lock was removed.
Wait on Time (WAITTIME)

**Format**

```
#include <QSYSINC/MIH/WAITTIME>

void waittime (_MI_Time *wait_interval,
               short options);
```

**Description**
The waittime function causes the current process to be placed in a wait state for the amount of time specified by the wait interval in accordance with the specified wait options.

**Parameters**

- **wait_interval (input)**
  Pointer to an 8-byte time interval prepared using the mitime function.

- **options (input)**
  The wait options.

**Notes on Usage**

- The builtin function _WAITTIME also provides access to the MI instruction.
Example
This example illustrates the use of the waittime function.

/**************************************************************************
/* Example Group: Processes <QSYSINC/MIH/WAITTIME> */
/* Function: waittime (wait or sleep for a specified time) */
/* Description: This example shows how you can suspend or make your job go to sleep for a specified amount of time. This is similar to the CL DLYJOB (delay job) command. */
/**************************************************************************/
#include <QSYSINC/MIH/WAITTIME>
#include <stdio.h>

_MI_Time time_to_wait; /* The amount of time to wait. */
int hours = 0, /* Time components used to create an _MI_Time value */
minutes = 0,
seconds = 15, /* that can be passed to the 'waittime' function */
hundredths = 0;
short wait_option;

int main(void) {
    /**************************************************************************
    /* Format an 'mitime' value to represent the amount of time to wait. */
    /* When 'waittime' is called, the job that is running this program will be suspended. */
    /**************************************************************************
    mitime( &time_to_wait, hours, minutes, seconds, hundredths );
    /* Tells the system to use normal handling of a suspended job. */
    wait_option = _WAIT_NORMAL;
    waittime( &time_to_wait, wait_option );
}
** no output **
Transfer Object Lock (XFRLOCK)

Format
#include <QSYSINC/MIH/XFRLOCK>

void xfrlock (_SYSPTR process,
              _SYSPTR object,
              char lock_state_to_transfer);

Description
The xfrlock function allocates a single lock to the receiving process. Upon com-
pletion of the transfer request, the current process no longer holds the transferred
lock.

Parameters
process (input)
    Pointer to the receiving process control space.

object (input)
    Pointer to the system object whose lock is to be transferred.

lock_state_to_transfer (input)
    Identifies the lock state to be transferred to the receiving process.

Notes on Usage
- The lock entry will always be marked as active by the xfrlock function. So, for
  the function version, it is not necessary to OR the lock state with
  _LOCK_ENTRY_ACTIVE.
- The builtin function _XFRLOCK, which supports multiple lock transfer requests,
  also provides access to the MI instruction.
Example
This example illustrates the use of the xfrlock function.

```c
/*------------------------------------------*/
/*
/* Example Group: Locks <QSYSINC/MIH/XFRLOCK>
/*
/* Function: xfrlock (Transfer Object Lock)
/*
/* Description:
This example shows how the job/process in which
this program is run can obtain an object lock
and then transfer it to another job. In order

(CPCS) of the other job/process needs to be known. */
/*
/* One way to accomplish this would be for an ILE */
/* 'matpratr' (materialize process attributes)
*/
/* function to obtain the PCS pointer. This pointer */
/* could then be put into a User Space or onto a */
/* User Queue. This program would then have to */
/* resolve to the User Space or User Queue and get */
/* the PCS pointer. The pointer could then be copied */
/* into the 'to_PCS' variable defined in this program */
/*
/* to pass to the 'xfrlock' function. */
/*
/* In order to not complicate this example too much, */
/* the 'to_PCS' variable will be filled in with the */
/* Process Control Space of the job/process that */
/* runs this program. This effectively means that */
/* the lock is not being transferred to another */
/* job, but this example should still illustrate the */
/* use of the 'xfrlock' function. */
/*
/*------------------------------------------*/

#include <QSYSINC/MIH/XFRLOCK>
#include <QSYSINC/MIH/RSLVSP>
#include <QSYSINC/MIH/LOCK>
#include <QSYSINC/MIH/MATPRATR>
#include <stdlib.h>

_SYSPTR some_object; /* The object being locked. */
_SYSPTR to_PCS; /* Process Control Space (PCS) */
/* (job) that the lock will be */
/* transferred to. */

_MI_Time timeout; /* The amount of time to wait */
/* for the lock to be placed. */

int hours = 0, /* Time components used to */
minutes = 0, /* create an _MI_Time value */
seconds = 0, /* that can be passed to the */
hundredths = 0; /* 'lock' function */
/*
/* The receiver template for */
/* The call to 'matpratr' to */

_MPRT_Template_T process_attributes; /* The call to 'matpratr' to */
```

Machine Interface Library Functions 249
/* get the PCS pointer. */

int main(void) {

    /* Get a pointer to the object. */
    some_object = rslvsp( _Usrq, "MYUSRQ", "MYLIB", _AUTH_ALL );

    /* Format an 'mitime' value to represent the timeout to be used by */
    /* the 'lock' function when trying to obtain an Exclusive-No-Read */
    /* (LENR) lock on the object. If the lock request cannot be satisfied */
    /* in the specified amount of time, then an exception will be raised. */
    /* The CL DSPJOB command is called to display all of the object locks */
    /* owned by the job/process that runs this program. */
    mitime( &timeout, hours, minutes, seconds, hundredths );

    lock( some_object, timeout, _LENR_LOCK );

    system( "DSPJOB OPTION(*JOBLCK)" );

    /* Obtain the PCS of the job/process that is running this program */
    /* by using the Materialize Process Attributes function with the */
    /* PCS selection option. */
    process_attributes_Ptr_Attr3.Template_Size = sizeof( _MPRT_PTR_T );
    matpratr( &process_attributes, NULL, _MPRT_CTRL_SPC );

    to_PCS = process_attributes_Ptr_Attr3.Mptr_Ptr;

    /* Now transfer the LENR lock on the object to the other job. DSPJOB */
    /* will be used again to allow the user to verify that the lock */
    /* disappears from this job. Since this example does not really */
    /* transfer the lock to another job, the lock will still appear on */
    /* the Display Job Locks display. */
    xfrlock( to_PCS, some_object, _LENR_LOCK );

    system( "DSPJOB OPTION(*JOBLCK)" );
    }

}
## Output

Display Job Locks

<table>
<thead>
<tr>
<th>Job: OMXNA3S4</th>
<th>User: VISCA</th>
<th>Number: 009616</th>
</tr>
</thead>
</table>

Job status: ACTIVE

Type options, press Enter.
5=Display job member locks

<table>
<thead>
<tr>
<th>Opt</th>
<th>Object</th>
<th>Library</th>
<th>Type</th>
<th>Lock</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>QSYS</td>
<td>*MENU</td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>MYUSRQ</td>
<td>MYLIB</td>
<td>*USRQ</td>
<td>*EXCL</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>OMXNA3S4</td>
<td>QSYS</td>
<td>*DEVD</td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>QADM</td>
<td>QSYS</td>
<td>*LIB</td>
<td>*SHRRD</td>
<td>HELD</td>
<td></td>
</tr>
</tbody>
</table>

F3=Exit  F5=Refresh  F10=Display job record locks  F12=Cancel

More...

Display Job Locks

<table>
<thead>
<tr>
<th>Job: OMXNA3S4</th>
<th>User: VISCA</th>
<th>Number: 009616</th>
</tr>
</thead>
</table>

Job status: ACTIVE

Type options, press Enter.
5=Display job member locks

<table>
<thead>
<tr>
<th>Opt</th>
<th>Object</th>
<th>Library</th>
<th>Type</th>
<th>Lock</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>QSYS</td>
<td>*MENU</td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*SHRNUP</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>MYUSRQ</td>
<td>MYLIB</td>
<td>*USRQ</td>
<td>*EXCL</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>OMXNA3S4</td>
<td>QSYS</td>
<td>*DEVD</td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*EXCLRD</td>
<td>HELD</td>
<td></td>
</tr>
<tr>
<td>QADM</td>
<td>QSYS</td>
<td>*LIB</td>
<td>*SHRRD</td>
<td>HELD</td>
<td></td>
</tr>
</tbody>
</table>

F3=Exit  F5=Refresh  F10=Display job record locks  F12=Cancel

More...
Translate with Table (XLATEWT)

Format
#include <QSYSINC/MIH/XLATEWT>

void xlatewt (char *receiver,
             const char *source,
             const char *table);

Description
The xlatewt function translates the source characters under control of the translate
table and places the translated characters into the receiver. The operation begins
with the leftmost character of the source and proceeds character-by-character, left-
to-right, until all source characters have been translated.

Parameters
rcvr_string (input/output)
Pointer to the receiver location to receive the translated string. The resulting
string will be null-terminated.

crcr_string (input)
The source string.

table (input)
Pointer to the translation table. Must be exactly 256 bytes in length. This table
specifies the translated values for the 256 possible byte values. Results are
undefined if the table provided is less than 256 bytes.

Notes on Usage
• The xlatewt function operates on null-terminated strings.
• This is a compatibility function with the same semantics as the XLATEWT MI
  instruction.
• The length of the string to translate must be between 0 and 16 773 104.
• The builtin function _XLATEB also provides access to the MI instruction.
  _XLATEB does the translation in place.

Exceptions
If any of the input parameters are invalid, an MCH3601 or MCH0601 exception may
be signalled.

Other exceptions possible from this function are:
  MCH0602 - Boundary alignment
  MCH0801 - Parameter Reference Violation
  MCH3402 - Object Destroyed
  MCH3602 - Pointer Type Invalid
  MCH6801 - Object Domain Violation
Example
This example illustrates the use of the xlatewt function.

```c
#include <QSYSINC/MIH/XLATEWT>
#include <string.h>
#include <stdio.h>

#define SIZE 100

char xtable[256] = {
    /* 0  1  2  3  4  5  6  7 */
    /* 8  9  A  B  C  D  E  F */
    '0', 0, 0, 0, 0, 0, 0, 0, /* 00-07 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 08-0F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 10-17 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 18-1F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 20-27 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 28-2F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 30-37 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 38-3F */
    ' ', 0, 0, 0, 0, 0, 0, 0, /* 40-47 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 48-4F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 50-57 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 58-5F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 60-67 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 68-6F */
    0, 0, 0, 0, 0, 0, 0, 0, /* 70-77 */
    0, 0, 0, 0, 0, 0, 0, 0, /* 78-7F */
    0, 'A', 'B', 'C', 'D', 'E', 'F', 'G', /* 80-87 */
    'H', 'I', 0, 0, 0, 0, 0, 0, /* 88-8F */
    0, 'J', 'K', 'L', 'M', 'N', 'O', 'P', /* 90-97 */
    'Q', 'R', 0, 0, 0, 0, 0, 0, /* 98-9F */
    0, 0, 'S', 'T', 'U', 'V', 'W', 'X', /* A0-A7 */
    'Y', 'Z', 0, 0, 0, 0, 0, 0, /* A8-AF */
    0, 0, 0, 0, 0, 0, 0, 0, /* B0-B7 */
    0, 0, 0, 0, 0, 0, 0, 0, /* B8-BF */
    0, 0, 0, 0, 0, 0, 0, 0, /* C0-C7 */
    0, 0, 0, 0, 0, 0, 0, 0, /* C8-CF */
    0, 0, 0, 0, 0, 0, 0, 0, /* D0-D7 */
    0, 0, 0, 0, 0, 0, 0, 0, /* D8-DF */
    0, 0, 0, 0, 0, 0, 0, 0, /* E0-E7 */
    0, 0, 0, 0, 0, 0, 0, 0, /* E8-EF */
    0, 0, 0, 0, 0, 0, 0, 0, /* F0-F7 */
    0, 0, 0, 0, 0, 0, 0, 0, /* F8-FF */
};
```
```c
int main(void) {
    char source[SIZE] = "good day";
    char receiver[SIZE];
    xlatewt(receiver, source, xtable);
    printf("The translated string is %s\n", receiver);
}
```

Output
The translated string is GOOD DAY
Appendix: Reference Summary

Table 1 briefly describes the MI functions. For each instruction the function prototype and description is provided.

The prototype for the associated builtin is also provided if it is identical to that of the function. The table also provides the prototypes for builtins which do not have a function interface.

The interface type of each function is also identified. The possible interface types are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Provides access to the MI instruction or performs an equivalent function.</td>
</tr>
<tr>
<td>Builtin</td>
<td>Provides access to the MI instruction.</td>
</tr>
</tbody>
</table>

The naming convention for the ILE C/400 MI library functions is as follows:

- Function names are all lowercase characters, for example, matinvat.
- Builtin names are in uppercase with an initial underscore character, for example, _MATS.

### Table 1 (Page 1 of 18). Machine Interface Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATIEXIT</td>
<td>#include &lt;mipgexec.h&gt;</td>
<td></td>
</tr>
</tbody>
</table>
| Function    | int atiexit (_OS_func_t *exit_handler,  
|             | void *parm)          | Establish an invocation exit handler by copying pointers exit_handler and parm to internal buffers. |
| Date        | date1                | date2       |
|             | date_duration        |             |
|             | _SCPCPRN date_duration,  
|             | _SCPCPRN date1,      |             |
|             | _SCPCPRN date2,      |             |
|             | _INST_Template_T2 *inst_t); | The date specified by date2 is subtracted from the date specified by date1 and the value of the result is placed in date_duration. |
| CDD         | #include <QSYSINC/MIH/MIDTTM>  |
| Function    | void cdd (_SCPCPRN date_duration,  
|             | _SCPCPRN date1,      |             |
|             | _SCPCPRN date2,      |             |
|             | _INST_Template_T2 *inst_t); |             |
|             | _CDD (_SCPCPRN date_duration,  
|             | _SCPCPRN date1,      |             |
|             | _SCPCPRN date2,      |             |
|             | _INST_Template_T2 *inst_t); |             |
| CLRBTS      | #include <QSYSINC/MIH/CLRBTS>  |
| Function    | void clrbts (_SCPCPRN bit_string,  
|             | unsigned int bit_offset); | Clear the bit in bit_string corresponding to bit_offset. |
|             | _SCPCPRN bit_string,  
|             | unsigned int bit_offset); |             |
|             | _CLRBTS (_SCPCPRN bit_string,  
|             | unsigned int bit_offset); |             |
| CMPPSPAD    | #include <QSYSINC/MIH/CMPPSPAD>  |
| Function    | int cmppspad (_SCPCPRN space1,  
<p>|             | _SCPCPRN space2); | Compares space addressability of space1 and space2. |
| Date        | space1               | space2      |</p>
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
</table>
| CMPPTRA     | #include <QSYSINC/MIH/CMPPTRA>  
Function
int cmptra (ANYPTR pointer1,  
ANYPTR pointer2);  
Builtin
int _CMPPTRA (ANYPTR pointer1,  
ANYPTR pointer2); | The objects addressed by `pointer1` and `pointer2` are compared to determine if they are addressing the same object. |
| CMPPTRT     | #include <QSYSINC/MIH/CMPPTRT>  
Function
int cmptrt (ANYPTR pointer,  
char type);  
Builtin
int _CMPPTRT (char type,  
ANYPTR pointer); | Compares the pointer type of `pointer` with the character type. |
| CPRDATA     | #include <QSYSINC/MIH/CPRDATA>  
Function
int cprdata (_SPCPRTR result,  
int result_length,  
_SPCPRTCH source,  
int src_length);  
Builtin
void _CPRDATA (_CPRD_Template_T *cprd_t); | Compresses user data of specified length. |
| CRTMTX      | #include <QSYSINC/MIH/CRTMTX>  
Builtin
int _CRTMTX (Mutex_T *,  
Mutex_Create_T *); | Creates a mutex. |
| CTD         | #include <QSYSINC/MIH/MIDTTM>  
Function
void ctd (_SPCPRTR time_duration,  
_SPCPRTCH time1,  
_SPCPRTCH time2,  
_INST_Template_T2 *inst_t);  
Builtin
void _CTD (_SPCPRTR time_duration,  
_SPCPRTCH time1,  
_SPCPRTCH time2,  
_INST_Template_T2 *inst_t); | The time specified by `time2`, is subtracted from the time specified by `time1` and the value of the result is placed in `time_duration`. |
| CTSD        | #include <QSYSINC/MIH/MIDTTM>  
Function
void ctsd (_SPCPRTR timestamp_duration,  
_SPCPRTCH timestamp1,  
_SPCPRTCH timestamp2,  
_INST_Template_T2 *inst_t);  
Builtin
void _CTSD (_SPCPRTR timestamp_duration,  
_SPCPRTCH timestamp1,  
_SPCPRTCH timestamp2,  
_INST_Template_T2 *inst_t); | The timestamp specified by `timestamp2`, is subtracted from the timestamp specified by `timestamp1` and the result is placed in `timestamp_duration`. |
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVTBC</td>
<td>#include &lt;QSYSINC/MIH/CVTBC&gt;</td>
<td>Converts source from the BSC compressed format to character, under control of controls, and places the result into receiver.</td>
</tr>
<tr>
<td></td>
<td>int cvtbc (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTBC_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td>BuiltIn</td>
<td>void _CVTBC (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTBC_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int *return_code);</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Completed Record</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Truncated Record</td>
</tr>
<tr>
<td>CVTCB</td>
<td>#include &lt;QSYSINC/MIH/CVTCB&gt;</td>
<td>Converts source from character to the BSC compressed format, under control of controls, and places the result into receiver.</td>
</tr>
<tr>
<td></td>
<td>int cvtcb (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTBC_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td>BuiltIn</td>
<td>void _CVTCB (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTBC_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int *return_code);</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td>CVTCH</td>
<td>#include &lt;QSYSINC/MIH/CVTCH&gt;</td>
<td>Each 8-bit character of source is converted to a 4-bit hex digit and placed in receiver.</td>
</tr>
<tr>
<td></td>
<td>void cvtch (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int size);</td>
<td></td>
</tr>
<tr>
<td>CVTCM</td>
<td>#include &lt;QSYSINC/MIH/CVTCM&gt;</td>
<td>Converts source from character format to MRJE compressed format under control of controls, and places the results in receiver.</td>
</tr>
<tr>
<td></td>
<td>int cvtcm (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTCM_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td>BuiltIn</td>
<td>void _CVTCM (_SPCPTR receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int rcvr_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_CVTCM_Control_T *controls,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTRCN source,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int src_length,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int *return_code);</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CVTCS</td>
<td>#include &lt;QSYSINC/MIH/CVTCS&gt;</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>int cvtcs (_SCPCTR receiver, unsigned int rcvr_length, _CVTCS_Control_T *controls, _SCPCTRCN source, unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td>Builtin</td>
<td>void _CVTCS (_SCPCTR receiver, unsigned int rcvr_length, _CVTCS_Control_T *controls, _SCPCTRCN source, unsigned int src_length, int *return_code);</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td>Result</td>
<td>Meaning</td>
</tr>
<tr>
<td>-1</td>
<td>Receiver Overrun</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Source Exhausted</td>
<td></td>
</tr>
<tr>
<td>CVTD</td>
<td>#include &lt;QSYSINC/MIH/MIDTTM&gt;</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>void cvtd (_SCPCTR result_date, _SCPCTRCN source_date, _INST_Templ_T *inst_f);</td>
<td></td>
</tr>
<tr>
<td>Builtin</td>
<td>void _CVTD (_SCPCTR result_date, _SCPCTRCN source_date, _INST_Templ_T *inst_f);</td>
<td></td>
</tr>
<tr>
<td>CVTEFN</td>
<td>#include &lt;QSYSINC/MIH/CVTEFN&gt;</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>int cvtefn (_SCPCTRCN source, unsigned int src_length, char mask[3]); double cvtefn(_SCPCTRCN source, unsigned int src_length, char mask[3]);</td>
<td></td>
</tr>
<tr>
<td>Builtin</td>
<td>void _CVTEFN(_SCPCTR receiver, _DPA_Templ_T *rcvr_descr, _SCPCTRCN source, unsigned int *src_length, _SCPCPR mask);</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>void _CVTEFN(_SCPCTR receiver, _DPA_Templ_T *rcvr_descr, _SCPCTRCN source, unsigned int *src_length, _SCPCPR mask);</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>void _CVTEFN2(_SCPCTR receiver, _DPA_Templ_T *rcvr_descr, _SCPCTRCN source, unsigned int *src_length);</td>
<td></td>
</tr>
<tr>
<td>CVTHC</td>
<td>#include &lt;QSYSINC/MIH/CVTHC&gt;</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>void cvthc (_SCPCTR receiver, _SCPCTRCN source, int size);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Converts source from character to SNA format under control of controls, and places the results in receiver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The date specified in source_date is converted to another calendar external or internal presentation and placed in result_date.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scans a character source for a valid decimal number in display format, removes the display character, and places the result in receiver.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Each 4-bit hex digit of source is converted to an 8-bit character and placed in receiver.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1 (Page 5 of 18). Machine Interface Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVTMC</td>
<td>#include &lt;QSYSINC/MIH/CVTMC&gt;</td>
<td>Converts source from MRJE compressed format to character format under control of controls, and places the results in receiver.</td>
</tr>
<tr>
<td></td>
<td>int cvtmc (_SFCPTR receiver, unsigned int rcvr_length, _CVTMC_Control_T *controls, _SFCPTCRN source, unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _CVTMC (_SFCPTR receiver, unsigned int rcvr_length, _CVTMC_Control_T *controls, _SFCPTCRN source, unsigned int src_length, int +return_code);</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Return Code</strong></td>
<td><strong>Result</strong></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td>CVTSC</td>
<td>#include &lt;QSYSINC/MIH/CVTSC&gt;</td>
<td>Converts source from SNA format to character format under control of controls, and places the results in receiver.</td>
</tr>
<tr>
<td></td>
<td>int cvtsc (_SFCPTR receiver, unsigned int rcvr_length, _CVTSC_Control_T *controls, _SFCPTCRN source, unsigned int src_length);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _CVTSC (_SFCPTR receiver, unsigned int rcvr_length, _CVTSC_Control_T *controls, _SFCPTCRN source, unsigned int src_length, int +return_code);</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Return Code</strong></td>
<td><strong>Result</strong></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Receiver Overrun</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Source Exhausted</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Escape Code Encountered</td>
</tr>
<tr>
<td>CVTT</td>
<td>#include &lt;QSYSINC/MIH/MIDTTM&gt;</td>
<td>The time specified in source_time is converted to another external or internal presentation and placed in result_time.</td>
</tr>
<tr>
<td></td>
<td>void cvtt (_SFCPTR result_time, _SFCPTCRN source_time, _INST_Template_T* inst_t);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _CVTT (_SFCPTR result_time, _SFCPTCRN source_time, _INST_Template_T* inst_t);</td>
<td></td>
</tr>
<tr>
<td>CVTTS</td>
<td>#include &lt;QSYSINC/MIH/MIDTTM&gt;</td>
<td>The timestamp specified in source_timestamp is converted to another external or internal presentation and placed in result_timestamp.</td>
</tr>
<tr>
<td></td>
<td>void cvtts (_SFCPTR result_timestamp, _SFCPTCRN source_timestamp, _INST_Template_T* inst_t);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _CVTTS (_SFCPTR result_timestamp, _SFCPTCRN source_timestamp, _INST_Template_T* inst_t);</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| CPYBLA | `#include <QSYSINC/MIH/CPYBLA>`  
`Function`  
```c
void cpybla (_SCPTR receiver,
           _SCPTRCN source,
           int size);
```
| Copies source to receiver (without pad). |
| CPYBLAP | `#include <QSYSINC/MIH/CPYBLAP>`  
`Function`  
```c
void cpyblap (_SCPTR receiver,
              int rcvr_size,
              _SCPTRCN source,
              int src_size,
              char pad);
```
| Copies source to receiver with pad. |
| CPYHEXNN | `#include <QSYSINC/MIH/CPYHEXNN>`  
`Function`  
```c
char cpyhexnn (char *dest, char source);
```
| The numeric hex digit value (rightmost 4 bits) of the byte referred to by source, is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte of dest. |
| CPYHEXNZ | `#include <QSYSINC/MIH/CPYHEXNZ>`  
`Function`  
```c
char cpyhexnz (char *dest, char source);
```
| The numeric hex digit value (rightmost 4 bits) of the byte referred to by source, is copied to the zone hex digit value (leftmost 4 bits) of the leftmost byte of dest. |
| CPYHEXZN | `#include <QSYSINC/MIH/CPYHEXZN>`  
`Function`  
```c
char cpyhexzn (char *dest, char source);
```
| The zone hex digit value (leftmost 4 bits) of the byte referred to by source, is copied to the numeric hex digit (rightmost 4 bits) of the leftmost byte referred to by dest. |
| CPYHEXZZ | `#include <QSYSINC/MIH/CPYHEXZZ>`  
`Function`  
```c
char cpyhexzz (char *dest, char source);
```
| The zone hex digit value (leftmost 4 bits) of the byte referred to by source, is copied to the zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by dest. |
| CPYNV | `#include <QSYSINC/MIH/CPYNV>`  
`Function`  
```c
void cpynv (_NUM_Desc_T rcvr_descr,
            _SCPTR receiver,
            _NUM_Desc_T src descr,
            _SCPTRCN source);
```
Builtin  
```c
void _LCPYNV (_SCPTR receiver,
              _DPA_Template_I *rcvr_attributes,
              _SCPTRCN source,
              _DPA_Template_I *src_attributes);
```
or  
```c
void _CPYNV (_NUM_Desc_T rcvr_descr,
            _SCPTR receiver,
            _NUM_Desc_T src descr,
            const _SCPTR source);
```
<p>| Copies the numeric value of source to the numeric value of receiver, with appropriate conversions. |</p>
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
</table>
| DCPDATA     | #include <QSYSINC/MIH/DCPDATA>  
Function
   int dcpdata (_SPCPtr result,  
   int result_length,  
   _SPCPtrRCN source);  
Builtin
   void _DCPDATA (_DCPD_Template_T *dcpd_t); | Decompresses user data of a specified length. |
| DECD        | #include <QSYSINC/MIH/MIDTTM>  
Function
   void dec (_SPCPtr result_date,  
   _SPCPtrRCN source_date,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t);  
Builtin
   void _DECD (_SPCPtr result_date,  
   _SPCPtrRCN source_date,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t); | The date specified by source_date is decremented by the date duration specified by duration. The resulting date is placed in result_date. |
| DECT        | #include <QSYSINC/MIH/MIDTTM>  
Function
   void dec (_SPCPtr result_time,  
   _SPCPtrRCN source_time,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t)  
Builtin
   void _DECT (_SPCPtr result_time,  
   _SPCPtrRCN source_time,  
   _SPCPtrRCN duration,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t); | The time specified by source_time is decremented by the time duration specified by duration. The resulting time is placed in result_time. |
| DECTS       | #include <QSYSINC/MIH/MIDTTM>  
Function
   void decs (_SPCPtr result_timestamp,  
   _SPCPtrRCN source_timestamp,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t);  
Builtin
   void _DECTS (_SPCPtr result_timestamp,  
   _SPCPtrRCN source_timestamp,  
   _SPCPtrRCN duration,  
   _INST_Template_T3 *inst_t); | The timestamp specified by source_timestamp is decremented by the date, time or timestamp duration specified by duration. The resulting timestamp is placed in result_timestamp. |
| DESMTX      | #include <QSYSINC/MIH/DESMTX>  
Builtin
   void _DESMTX (_Mutex_T *  
   _Mutex_Destroy_Opt_T *); | Destroys a mutex. |
| DEQ         | #include <QSYSINC/MIH/DEQ>  
Function
   void _DEQ (_DEQ_Msg_Prefix_T *msg_prefix,  
   _SPCPtr message,  
   _SYSPtr queue);  
Builtin
   void _DEQ_WAIT (_DEQ_Msg_Prefix_T *msg_prefix,  
   _SPCPtr message,  
   _SYSPtr *queue); | Retrieves a queue message based on the queue type specified during the queue's creation. Process is placed in a wait state. |
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEQI</td>
<td><code>#include &lt;QSYSINC/MIH/DEQI&gt;</code>&lt;br&gt;<code>Function</code>&lt;br&gt;<code>int deqi (DEQ_Msg_Prefix_T *msg_prefix,&lt;br&gt;_SPCPTR message,&lt;br&gt;_SYSPTR queue);</code>&lt;br&gt;<code>Builtin</code>&lt;br&gt;<code>int _DEQ (DEQ_Msg_Prefix_T *msg_prefix,&lt;br&gt;_SPCPTR message,&lt;br&gt;_SYSPTR *queue);</code>&lt;br&gt;<code>Return Value</code>&lt;br&gt;<code>Value Meaning</code>&lt;br&gt;<code>1 Message Dequeued</code>&lt;br&gt;<code>0 Message Not Dequeued</code></td>
<td>Retrieves a queue message based on queue type (FIFO, LIFO, or keyed). Process is not placed in a wait state.</td>
</tr>
<tr>
<td>EDIT</td>
<td><code>#include &lt;QSYSINC/MIH/EDIT&gt;</code>&lt;br&gt;<code>Function</code>&lt;br&gt;<code>void edit (_SPCPTR receiver,&lt;br&gt;unsigned int rcvr_length,&lt;br&gt;_SPCPRCN source,&lt;br&gt;_NUM_Descr_T src_descr,&lt;br&gt;_SPCPR mask,&lt;br&gt;unsigned int mask_length);</code>&lt;br&gt;<code>Builtin</code>&lt;br&gt;<code>void _LBEDIT (_SPCPR receiver,&lt;br&gt;unsigned int *rcvr_length,&lt;br&gt;_SPCPRCN source,&lt;br&gt;_DPA_Template_T *src_descr,&lt;br&gt;_SPCPR mask,&lt;br&gt;unsigned int *mask_length);</code>&lt;br&gt;<code>Value Meaning</code>&lt;br&gt;<code>1 Message Dequeued</code>&lt;br&gt;<code>0 Message Not Dequeued</code></td>
<td>The value of source is transformed under control of mask and the result is placed in receiver.</td>
</tr>
<tr>
<td>EDIT with Packed Decimal Source</td>
<td><code>#include &lt;QSYSINC/MIH/EDIT&gt;</code>&lt;br&gt;<code>Function</code>&lt;br&gt;<code>void edit_packed (_SPCPR receiver,&lt;br&gt;unsigned int recr_length,&lt;br&gt;_SPCPRCN source,&lt;br&gt;unsigned int src_length,&lt;br&gt;_SPCPR mask,&lt;br&gt;unsigned int mask_length);</code>&lt;br&gt;<code>Builtin</code>&lt;br&gt;<code>void _EDITPD (_SPCPR receiver,&lt;br&gt;unsigned int *rcvr_length,&lt;br&gt;_SPCPRCN source,&lt;br&gt;unsigned int *src_length,&lt;br&gt;_SPCPR mask,&lt;br&gt;unsigned int *mask_length);</code></td>
<td>The value of the packed decimal source is transformed under control of mask, and the result is placed in receiver.</td>
</tr>
<tr>
<td>ENQ</td>
<td><code>#include &lt;QSYSINC/MIH/ENQ&gt;</code>&lt;br&gt;<code>Function</code>&lt;br&gt;<code>void enq (_SYSPTR queue,&lt;br&gt;_ENQ_Msg_Prefix_T *msg_prefix,&lt;br&gt;_SPCPR message);</code>&lt;br&gt;<code>Builtin</code>&lt;br&gt;<code>void _ENQ (_SYSPTR *system_ptr,&lt;br&gt;_ENQ_Msg_Prefix_T *msg_prefix,&lt;br&gt;_SPCPR message);</code></td>
<td>A message is enqueued according to the queue type attribute specified during the queue's creation.</td>
</tr>
<tr>
<td>ENSOBJ</td>
<td><code>#include &lt;QSYSINC/MIH/ENSOBJ&gt;</code>&lt;br&gt;<code>Function</code>&lt;br&gt;<code>void ensobj (_SYSPTR object);</code>&lt;br&gt;<code>Builtin</code>&lt;br&gt;<code>void _ENSOBJ (_SYSPTR *object);</code></td>
<td>The object specified is protected from volatile storage loss.</td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| EXTREXP     | #include <QSYSINC/MIH/EXTREXP>  
Function: int extexp (double source); | Extracts the exponent portion of the floating point scalar and returns the value as a 4-byte integer. |
| FNDINXEN    | #include <QSYSINC/MIH/FNDINXEN>  
Function: int_SCPTR findxen (_SCPSTR receiver, _SYSPTR index_obj, _IIX_Opt_List_T *option_list, _SCPSTR search_arg);  
Builtin: void _FNDINXEN (_SCPSTR receiver, _SYSPTR *index_obj, _IIX_Opt_List_T *option_list, _SCPSTR search_arg); | Searches index_obj according to the search criteria specified in option_list and search argument search_arg. |
| FNDRINVN    | #include <QSYSINC/MIH/FNDRINVN>  
Function: int fndrinvn (_INV_Template_T *inv_t, int Src_optn, _SCPSTRN srcn_arg, char compare);  
Builtin: void _FNDRINVN (int *rel_inv_number, _FNDR_Search_Template_T *srch_t);  
or  
void _FNDRINVN2 (int *rel_inv_number, _INV_Template_T *inv_t, _FNDR_Search_Template_T *srch_t); | Searches a specified range of invocations until an invocation is found which satisfies the search criteria. |
| INCD        | #include <QSYSINC/MIH/MIDTMM>  
Function: void incd (_SCPSTR result_date, _SCPSTRN source_date, _SCPSTRN duration, _INST_Template_T *inst_t);  
Builtin: void _INCD (_SCPSTR result_date, _SCPSTRN source_date, _SCPSTRN duration, _INST_Template_T *inst_t); | The date specified by source_date is incremented by the date duration specified by duration. The resulting date is placed in result_date. |
| INCT        | #include <QSYSINC/MIH/MIDTMM>  
Function: void intct (_SCPSTR result_time, _SCPSTRN source_time, _SCPSTRN duration, _INST_Template_T *inst_t);  
Builtin: void _INCT (_SCPSTR result_time, _SCPSTRN source_time, _SCPSTRN duration, _INST_Template_T *inst_t); | The time specified by source_time is incremented by the time duration specified by duration. The resulting time is placed in result_time. |
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCTS</td>
<td>#include &lt;QSYSINC/MIH/MIDTTM&gt;</td>
<td>The timestamp specified by <code>source_timestamp</code> is incremented by the date, time or timestamp duration specified by <code>duration</code>. The resulting timestamp is placed in <code>result_timestamp</code>.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void incts (_SCPTR result_timestamp,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTRN source_timestamp,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTRN duration,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_INST_Template_T *inst_t);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _INCTS (_SCPTR result_timestamp,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTRN source_timestamp,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTRN duration,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_INST_Template_T *inst_t);</td>
<td></td>
</tr>
<tr>
<td>INSINXEN</td>
<td>#include &lt;QSYSINC/MIH/INSINXEN&gt;</td>
<td>Inserts one or more new entries into the <code>index_obj</code> according to the criteria specified on <code>option_list</code>.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void insinxen (_SYSPTR index_obj,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTR new_entry,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_IIX_Opt_List_T</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*option_list);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _INSINXEN (_SYSPTR *index_obj,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SCPTR new_entry,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_IIX_Opt_List_T</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*option_list);</td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>#include &lt;QSYSINC/MIH/LOCK&gt;</td>
<td>Requests the lock(s) for the system object(s) identified be allocated to the issuing process.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void lock (_SYSPTR object,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MI_Time wait_time,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void Lock (_LOCK_Template_T *lock_request);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _LOCK (_LOCK_Template_T *lock_request);</td>
<td></td>
</tr>
<tr>
<td>LOCKMTX</td>
<td>#include &lt;QSYSINC/MIH/LOCKMTX&gt;</td>
<td>Locks a mutex.</td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _LOCKMTX (_Mutex_T *,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_Mutex_lock_T *);</td>
<td></td>
</tr>
<tr>
<td>LOCKSL</td>
<td>#include &lt;QSYSINC/MIH/LOCKSL&gt;</td>
<td>Grants the space location(s) to the issuing process according to the lock request.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void locksl (_SCPTR location,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void locksl2 (_SCPTR location,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MI_Time wait_time,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _LOCKSL (_SCPTR *location,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char *lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _LOCKSL2 (_LOCKSL_Template_T *</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*lock_t);</td>
<td></td>
</tr>
<tr>
<td>MATACTAT</td>
<td>#include &lt;QSYSINC/MIH/MATACTAT&gt;</td>
<td>Materializes the information selected by <code>attr_selection</code> for the program activation specified by <code>act_mark</code> and stores it in <code>receiver</code>.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matactat (_MTACT_Template_T *receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int act_mark,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char attr_selection);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATACTAT (_MTACT_Template_T *receiver,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned int *act_mark,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char *attr_selection);</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| MATAGAT     | `#include <QSYSINC/MIH/MATAGAT>` | `Function`  
void matagat (_MAGAT_Template_T *receiver,  
_SYSPTR access_group);  
Builtin  
void _MATAGAT (_MAGAT_Template_T *receiver,  
_SYSPTR *access_group); | Materializes the attributes of access_group into receiver. |
| MATAGPAT    | `#include <QSYSINC/MIH/MATAGPAT>` | `Function`  
void matagpat (_MAGP_Template_T *receiver,  
unsigned int act_grp_mark,  
char attr_selection);  
Builtin  
void _MATAGPAT (_MAGP_Template_T *receiver,  
unsigned int *act_grp_mark,  
char *attr_selection); | Materializes the information selected by attr_selection for activation group act_grp_mark and stores it in receiver. |
| MATAOL      | `#include <QSYSINC/MIH/MATAOL>` | `Function`  
void mataol (_MAOL_Template_T *receiver,  
_ANYPTR pointer);  
Builtin  
void _MATAOL (_MAOL_Template_T *receiver,  
_ANYPTR *pointer); | Materializes the current allocated locks on a system object or space location into receiver. |
| MATINVAT    | `#include <QSYSINC/MIH/MATINVAT>` | `Function`  
void matinvat (_SPCPPTR receiver,  
_INV_Template_T *inv_t,  
int attr_id,  
int rcvr_length);  
Builtin  
void _MATINVAT1 (_SPCPPTR receiver,  
_Select_Template_T *select_t);  
or  
void _MATINVAT2 (_SPCPPTR receiver,  
_INV_Template_T *inv_t,  
_Select_Template_T *select_t); | Materializes the attributes of the specified invocation into receiver. |
| MATINXAT    | `#include <QSYSINC/MIH/MATINXAT>` | `Function`  
void matinxat (_IXI_Template_T *receiver,  
_SYSPTR index_obj);  
Builtin  
void _MATINXAT (_IXI_Template_T *receiver,  
_SYSPTR +index_obj); | Materializes the creation attribute and current operational statistics of index_obj into receiver. |
| MATMATR     | `#include <QSYSINC/MIH/MATMATR>` | `Function`  
void matmatr (_MMTR_Template_T *receiver,  
short attr);  
Builtin  
void _MATMATR1 (_MMTR_Template_T *receiver,  
short *attr); | Materializes the machines attributes specified by attr into receiver. |
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
</table>
| MATMDATA    | #include <QSYSINC/MIH/MATMDATA>  
Function  
void matmd (MDATA_Template_T *receiver,  
short options);  
Builtin  
void _MATMDATA (MDATA_Template *receiver,  
short options); | Materializes the requested machine data into receiver. |
| MATMTX      | #include <QSYSINC/MIH/MATMTX>  
Builtin  
void _MATMTX (_Mutex_Mat_T *,  
_Mutex_T *,  
_Mutex_Mat_Opt_T *); | Materializes a mutex. |
| MATOBJLK    | #include <QSYSINC/MIH/MATOBJLK>  
Function  
void matobjlk (_MOBJL_Template_T *receiver,  
_ANYPTR object);  
Builtin  
void _MATOBJLK (_MOBJL_Template_T *receiver,  
_ANYPTR object); | The current lock status of the object or space location is materialized into receiver. |
| MATPG       | #include <QSYSINC/MIH/MATPG>  
Function  
void matpg (_MATPG_Template_T *receiver,  
_SYSPTR program);  
Builtin  
void _MATPG (_MATPG_Template_T *receiver,  
_SYSPTR *program); | The program is materialized into receiver. |
| MATPRMTX    | #include <QSYSINC/MIH/MATPRMTX>  
Builtin  
void _MATPRMTX (_Mutex_Matr_T *,  
_SYSPTR *,  
_Mutex_Matr_Opt_T *); | Materials process mutex locks. |
| MATPTR      | #include <QSYSINC/MIH/MATPTR>  
Function  
void matptr (_MPTR_Template_T *receiver,  
_ANYPTR pointer);  
Builtin  
void _MATPTR (_MPTR_Template_T *receiver,  
_ANYPTR *pointer); | The materialized form of pointer is returned in receiver. |
| MATPTRL     | #include <QSYSINC/MIH/MATPTRL>  
Function  
void matptrl (_MPTL_Template_T *receiver,  
_SPCPTRCN source,  
int length);  
Builtin  
void _MATPTRL (_MPTL_Template_T *receiver,  
_SPCPTRCN source,  
int length); | Finds the pointers in a subset of a space and produces a bit mapping of their relative locations. |
| MATPRAGP    | #include <QSYSINC/MIH/MATPRAGP>  
Function  
void matprag (_MPRAG_Template_T *receiver);  
Builtin  
void _MATPRAGP (_MPRAG_Template_T *receiver); | A list of the activation groups which exist in the current process are returned in receiver. |
<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATPRATR</td>
<td>#include &lt;QSYSINC/MIH/MATPRATR&gt;</td>
<td>Materialize attributes of control_spc into receiver according to the specified options.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matpratr (_MPRT_Template_T *receiver, _SYSPTR control_spc, char *options);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATPRATR ( _MPRT_Template_T *receiver, char *options);</td>
<td></td>
</tr>
<tr>
<td>MATPRATR</td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATPRATR2 ( _MPRT_Template_T *receiver, _SYSPTR *control_spc, char *options);</td>
<td></td>
</tr>
<tr>
<td>MATPRATR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATPRLK</td>
<td>#include &lt;QSYSINC/MIH/MATPRLK&gt;</td>
<td>The lock status of the process identified by pcs is materialized into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matprl (_MPRL_Template_T *receiver, _SYSPTR pcs);</td>
<td></td>
</tr>
<tr>
<td>MATPRLK</td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATPRLK1 ( _MPRL_Template_T *receiver);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td>MATPRLK</td>
<td>void _MATPRLK2 ( _MPRL_Template_T *receiver, _SYSPTR *pcs);</td>
<td></td>
</tr>
<tr>
<td>MATQAT</td>
<td>#include &lt;QSYSINC/MIH/MATQAT&gt;</td>
<td>The attributes of the queue are materialized into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matqat ( _MQAT_Template_T *receiver, _SYSPTR queue);</td>
<td></td>
</tr>
<tr>
<td>MATQAT</td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATQAT ( _MQAT_Template_T *receiver, _SYSPTR *queue);</td>
<td></td>
</tr>
<tr>
<td>MATQMSG</td>
<td>#include &lt;QSYSINC/MIH/MATQMSG&gt;</td>
<td>Materializes selected messages in queue into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matqmsg ( _MQMS_TEMPLATE_T *receiver, _SYSPTR queue, char *selection, int max_key, int max_msg, _SPCPTTR key);</td>
<td></td>
</tr>
<tr>
<td>MATQMSG</td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATQMSG ( _MQMS_Template_T *receiver, _SYSPTR *queue, _MQMS_Select_T *selection);</td>
<td></td>
</tr>
<tr>
<td>MATRMD</td>
<td>#include &lt;QSYSINC/MIH/MATRMD&gt;</td>
<td>The data items requested by control are materialized into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matrd ( _MATRMD_Template_T *receiver, _SPCPTTR control);</td>
<td></td>
</tr>
<tr>
<td>MATRMD</td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATRMD ( _MATRMD_Template_T *receiver, _SPCPTTR control);</td>
<td></td>
</tr>
<tr>
<td>MATS</td>
<td>#include &lt;QSYSINC/MIH/MATS&gt;</td>
<td>The current attributes of the space object specified by space_object are materialized into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void mats ( _SPC_Template_T *receiver, _SYSPTR space_object);</td>
<td></td>
</tr>
<tr>
<td>MATS</td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATS ( _SPC_Template_T *receiver, _SYSPTR *space_object);</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MATSELLK</td>
<td>#include &quot;&lt;QSYSINC/MIH/MATSELLK&gt;&quot;</td>
<td>Materializes the locks held by the process issuing this instruction for the object or space location specified.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matsellk (MSLL_Template_T *receiver, _ANYPTR pointer);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATSELLK (MSLL_Template_T *receiver, _ANYPTR *pointer);</td>
<td></td>
</tr>
<tr>
<td>MATSOBJ</td>
<td>#include &quot;&lt;QSYSINC/MIH/MATSOBJ&gt;&quot;</td>
<td>Materializes the identity and size of a system object addressed by object into receiver.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void matsobj (MSOB_Template_T *receiver, _SYSPTR object);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATSOBJ (MSOB_Template_T *receiver, _SYSPTR *object);</td>
<td></td>
</tr>
<tr>
<td>MATTOD</td>
<td>#include &quot;&lt;QSYSINC/MIH/MATTOD&gt;&quot;</td>
<td>Materializes the time_of_day clock.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void mattod (MI_Time time_of_day);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MATTOD (MI_Time time_of_day);</td>
<td></td>
</tr>
<tr>
<td>MITIME</td>
<td>#include &quot;&lt;QSYSINC/MIH/MICOMMON&gt;&quot;</td>
<td>Sets receiver to the AS/400 system value for time.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_MI_TIME *mitime (MI_Time *receiver, int hours, int minutes, int seconds, int hundredths);</td>
<td></td>
</tr>
<tr>
<td>MODASA</td>
<td>#include &quot;&lt;QSYSINC/MIH/MODASA&gt;&quot;</td>
<td>Extend the current allocation of automatic storage by size bytes.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTR modasa (unsigned int size);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_SPCPTR _MODASA (unsigned int size);</td>
<td></td>
</tr>
<tr>
<td>MODINX</td>
<td>#include &quot;&lt;QSYSINC/MIH/MODINX&gt;&quot;</td>
<td>Modifies the selected attributes of index_obj.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void modinx (_SYSPTR index_obj, char mod_option);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MODINX (_SYSPTR *index_obj, unsigned int *mod_option);</td>
<td></td>
</tr>
<tr>
<td>MODS</td>
<td>#include &quot;&lt;QSYSINC/MIH/MODS&gt;&quot;</td>
<td>Resize the space associated with space_object to size bytes.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void mods (_SYSPTR space_object, int size);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MODS1 (_SYSPTR *space_object, int *size);</td>
<td></td>
</tr>
<tr>
<td>MODS2</td>
<td>#include &quot;&lt;QSYSINC/MIH/MODS2&gt;&quot;</td>
<td>The attributes of the space associated with space_object are modified with the attribute values specified in space_t.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void mods2 (_SYSPTR space_object, _SPC_MOD_T *space_t);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _MODS2 (_SYSPTR *space_object, _SPC_MOD_T *space_t);</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RSLVSP</td>
<td>#include &lt;QSYSINC/MIH/RSLVSP&gt;</td>
<td>Locates an object identified by the object identifier (type, name, and library) and returns the object’s addressability in a system pointer.</td>
</tr>
<tr>
<td></td>
<td>_SYSPTR rslvsp (_OBJ_TYPE_T obj_type, _OBJ_NAME obj_name, _LIB_NAME lib_name, _REQ_AUTH auth);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin void _RSLVSP1 (_SYSPTR *ptr_to_object); or void _RSLVSP2 (_SYSPTR *ptr_to_object, _RSLV_Template_T *objid_and_authreq); or void _RSLVSP3 (_SYSPTR *ptr_to_object, _SYSPTR *library); or void _RSLVSP4 (_SYSPTR *ptr_to_object, _RSLV_Template_T *objid_and_authreq, _SYSPTR *library); or void _RSLVSP5 (_SYSPTR *ptr_to_object, _REQ_AUTH *auth); or void _RSLVSP6 (_SYSPTR *ptr_to_object, _RSLV_Template_T *objid_and_authreq, _REQ_AUTH *auth); or void _RSLVSP7 (_SYSPTR *ptr_to_object, _SYSPTR *library, _REQ_AUTH *auth); or void _RSLVSP8 (_SYSPTR *ptr_to_object, _RSLV_Template_T *objid_and_authreq, _SYSPTR *library, _REQ_AUTH *auth);</td>
<td></td>
</tr>
<tr>
<td>RETCA</td>
<td>#include &lt;QSYSINC/MIH/RETCA&gt;</td>
<td>Retrieves from the machine and returns the 4-byte value containing the selected computational attributes.</td>
</tr>
<tr>
<td></td>
<td>unsigned int retca (unsigned int mask);</td>
<td></td>
</tr>
<tr>
<td>RMVINXEN</td>
<td>#include &lt;QSYSINC/MIH/RMVINXEN&gt;</td>
<td>The index entries identified by option_list and search_arg are removed from index_obj and returned in receiver.</td>
</tr>
<tr>
<td></td>
<td>_SPCPTR rmvinxen (_SPC_PTR receiver, _SPC_PTR index_obj, _IIX_OptList_T *option_list, _SPC_PTR search_arg);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin void _RMVINXEN1 (_SPC_PTR *index_obj, _IIX_OptList_T *option_list, _SPC_PTR search_arg); or void _RMVINXEN2 (_SPC_PTR receiver, _SPC_PTR *index_obj, _IIX_OptList_T *option_list, _SPC_PTR search_arg);</td>
<td></td>
</tr>
<tr>
<td>SCANWC</td>
<td>#include &lt;QSYSINC/MIH/SCANWC&gt;</td>
<td>The source string is scanned for occurrences of a character value satisfying the criteria in control and options.</td>
</tr>
<tr>
<td></td>
<td>int scanwc (char *source, _SCW_Control_T *controls, char options);</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>C Function Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SCANX</td>
<td>#include &lt;QSYSINC/MIH/SCANX&gt;</td>
<td>The source string is scanned for occurrences of a character value satisfying the criteria in control and options.</td>
</tr>
<tr>
<td>SETACST</td>
<td>#include &lt;QSYSINC/MIH/SETACST&gt;</td>
<td>Specifies the access state that the issuing process has for a set of objects or subobject elements in the execution interval following the execution of this instruction.</td>
</tr>
<tr>
<td>SETBTS</td>
<td>#include &lt;QSYSINC/MIH/SETBTS&gt;</td>
<td>Set the bit in bit_string corresponding to bit_offset.</td>
</tr>
<tr>
<td>SETCA</td>
<td>#include &lt;QSYSINC/MIH/SETCA&gt;</td>
<td>Sets the computation attributes selected and stores them into the machine.</td>
</tr>
<tr>
<td>SETSPFP</td>
<td>#include &lt;QSYSINC/MIH/SETSPFP&gt;</td>
<td>Returns a system pointer to the system object addressed by pointer.</td>
</tr>
<tr>
<td>SETPPFP</td>
<td>#include &lt;QSYSINC/MIH/SETPPFP&gt;</td>
<td>Returns addressability to a space object from pointer.</td>
</tr>
<tr>
<td>SETSPO</td>
<td>#include &lt;QSYSINC/MIH/SETSPO&gt;</td>
<td>The value of the offset is assigned to the offset portion of pointer.</td>
</tr>
<tr>
<td>STSPPO</td>
<td>#include &lt;QSYSINC/MIH/STSPPO&gt;</td>
<td>Returns the offset value of pointer.</td>
</tr>
</tbody>
</table>
Table 1 (Page 17 of 18). Machine Interface Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESTAU</td>
<td>#include <code>&lt;QSYSINC/MIH/TESTAU&gt;</code></td>
<td>Verifies that the object authorities and/or ownership rights specified by <code>req_auth</code> are currently available to the process.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>short testau (_SYSPTR obj, short offset, short req_auth);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int _TESTAU1 (_ANYPTR *pointer, short *req_auth);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int _TESTAU2 (short *ovail_auth, _ANYPTR *pointer, short *req_auth);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Authorized</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Not Authorized</td>
</tr>
<tr>
<td>TRIML</td>
<td>#include <code>&lt;QSYSINC/MIH/TRIML&gt;</code></td>
<td>Returns the length of string after <code>trim_char</code> has been trimmed from the end.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int triml (char *string, char trim_char);</td>
<td></td>
</tr>
<tr>
<td>TSTBTS</td>
<td>#include <code>&lt;QSYSINC/MIH/TSTBTS&gt;</code></td>
<td>Test the bit in <code>bit_string</code> corresponding to <code>bit_offset</code>.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int tstbs (_SPC PTR bit_string, unsigned int bit_offset);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int _TSTBTS (_SPC PTR bit_string, unsigned int bit_offset);</td>
<td></td>
</tr>
<tr>
<td>UNLOCK</td>
<td>#include <code>&lt;QSYSINC/MIH/UNLOCK&gt;</code></td>
<td>Releases the object lock(s) specified.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void unlock (_SYSPTR object, char unlock_option, char lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void Unlock (_LOCK_Template_T *unlock_request);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _UNLOCK (_LOCK_Template_T *unlock_request);</td>
<td></td>
</tr>
<tr>
<td>UNLOCKSL</td>
<td>#include <code>&lt;QSYSINC/MIH/UNLOCKSL&gt;</code></td>
<td>Releases the lock(s) for the specified space location(s)</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void unlockl (_SPC PTR spc_loc, char lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _UNLOCKSL1 (_SPC PTR <em>spc_loc, char</em> lock_state);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _UNLOCKSL2 (_LOCKSL_Template_T **lock_t);</td>
<td></td>
</tr>
<tr>
<td>WAITTIME</td>
<td>#include <code>&lt;QSYSINC/MIH/WAITTIME&gt;</code></td>
<td>Causes the process to wait for the time interval specified.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void waittime (_MI_Time *wait_interval, short options);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Builtin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void _WAITTIME (_WAIT_Template_T *wait_t);</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1 (Page 18 of 18). Machine Interface Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C Function Prototype</th>
<th>Description</th>
</tr>
</thead>
</table>
| **XFRLOCK** | `#include <QSYSINC/MIH/XFRLOCK>`  
void xfrlock (_SYSPTR process,  
_SYSPTR object,  
char lock_state_to_transfer);  

void _XFRLOCK (_SYSPTR *receiver,  
_LOCK_Template_T *xfer_t); | The receiving process is allocated the locks designated in `xfer_t`. |
| **XlateWT** | `#include <QSYSINC/MIH/XLATEWT>`  
void xlatewt (char *receiver,  
const char *source,  
const char *table);  

void _XLETTEB (_SPCPTR source,  
_SPCPTRN table,  
unsigned int length); | Translates the source characters under control of the translate table. |
For additional information about topics related to ILE C/C++ MI Library Reference refer to the following IBM publications:

- **ILE Concepts**, SC41-4606, explains concepts and terminology pertaining to the Integrated Language Environment (ILE) architecture of the OS/400 licensed program. Topics covered include creating modules, binding, running programs, debugging programs, and handling exceptions.

- **Machine Interface Functional Reference**, SC41-4810 describes the machine interface instruction set. Describes the functions that can be performed by each instruction and also the necessary information to code each instruction.

- **System API Reference**, SC41-4801, provides information for the experienced programmer on how to use the application programming interfaces (APIs) to such OS/400 functions as:
  - Dynamic Screen Manager
  - Files (database, spooled, hierarchical)
  - Message handling
  - National language support
  - Network management
  - Objects
  - Problem management
  - Registration facility
  - Security
  - Software products
  - Source debug
  - UNIX-type
  - User-defined communications
  - User interface
  - Work management

Includes original program model (OPM), Integrated Language Environment (ILE), and UNIX-type APIs.
# Index

## A
- atiexit 15

## C
- cdd 17
- cirbts 21
- cmppspad 23
- cmpptra 25
- cmpptrt 28
- cprdata 31
- cpybla 69
- cpyblap 71
- cpyhexnn 73
- cpyhexnz 75
- cpyhexzn 77
- cpyhexzz 79
- cpynv 81
- ctd 33
- ctd 36
- cvtbc 40
- cvtcb 42
- cvtch 44
- cvtcm 46
- cvtcs 48
- cvtd 50
- cvtefn 54
- cvthc 56
- cvtmc 58
- cvtsc 60
- cvtt 62
- cvtts 65

## D
- dcpdata 83
- decd 85
- dect 89
- dects 92
- deq 96
- deqi 98

## E
- edit 100
- edit packed decimal 102
- enq 104
- ensobj 106
- extrexp 108

## F
- fndinxen 110
- fndrinvn 113

## I
- incd 116
- inct 120
- incts 123
- insinxen 127

## L
- lock 130
- locksl 133
- locksl2 136

## M
- Machine Interface Support Include File 155
- matactat 139
- matagat 141
- matagpat 145
- mataol 148
- matinvat 150
- matinxat 153
- matmatr 155
- matmdata 157
- matobjlk 159
- matpg 161
- matpragp 168
- matpratr 171
- matprik 173
- matptr 163
- matptrl 165
- matqat 176
- matqmsg 178
- matrmd 182
- mats 184
- matsellk 186
- matsobj 188
- mattod 190
Index

mitime 192
modasa 195
modinx 197
mods 200
mods2 203

R
retca 208
rmvinxen 211
rslvsp 206

S
scanwc 214
setacst 216
setbts 219
setca 221
setspfp 225
setsppfp 228
setsppo 231
stsppo 233

T
testau 234
triml 236
tstbts 238

U
unlock 240
unlocksl 243

W
waittime 246

X
xflock 248
xlatewt 252