CRSP® RESEARCH DATA PRODUCTS

PROGRAMMER'S GUIDE

CRSP US STOCK & US INDEX DATABASES
CRSP/COMPUSTAT MERGED DATABASE

WHO WE ARE:

Founded at the University of Chicago, CRSP has transformed the world of finance. More than 65 years ago, CRSP developed the first market database that allowed investors to measure historic rates of return for U.S. stocks, broadening the appeal of equity ownership. More than 15 years ago, CRSP launched investable indexes that have become trusted by investors globally, including managers of the world's largest mutual fund.

In February 2026, the acquisition of CRSP brought the CRSP Market Indexes—benchmarks for over $3 trillion in U.S. equities—into the Morningstar Indexes family. Additionally, the CRSP Research Data Products, renowned for their academic rigor, historical depth, and accuracy, will further enhance Morningstar's equity research and data capabilities. This integration unites two trusted sources of market insight, reinforcing a shared commitment to transparency, quality, and investor-focused solutions.

Table of Contents

Chapter 1: Overview 5

Overview of CRSPAccess Databases 5
API Library 6
Notational Conventions 7

Chapter 2: Item-Based Access in C 8

Introduction 8
CRSP C API Data Objects 8
Supporting Information 11
Generic Data Types 11
Accessing CRSP Databases 12
CRSP US Stock Database 12
CRSP US Index Database 14
CRSP/Compustat Merged Database 15
CRSP C API Data Types 27
Container Objects 27
Supporting Types 29

Chapter 3: Item-Based Access in Fortran 32

CRSP Fortran 95 API Data Objects 32
Supporting Information 35
Generic Data Types 35
Accessing CRSP Databases 37
CRSP US Stock Database 37
CRSP US Index Database 42
CRSP/Compustat Merged Database 45
CRSP Fortran 95 API Functions 59
CRSP Fortran 95 API Data Types 68
Container Objects 69
Supporting Types 71
CRSP_VARSTRING_T Type 73

Chapter 4: Item-Based Samples 76

Building and Executing Programs 76
Visual Studio 2010 - C Compiler Instructions 85

Chapter 5: Legacy Set Access in C 96

CRSPAccess C Data Structures 96
Data Organization for C Programming 96
Data Objects 96

Document last modified February 12, 2026

Set Structures and Usage 98
C Sample Programs 115
CRSPAccess C Library 116
Chapter 6: Legacy Set Access in Fortran 204
FORTRAN-95 Data Structures 204
Data Organization for FORTRAN-95 Programming 204
Data Objects 204
Set Structures and Usage 206
FORTRAN-95 Stock Sample Programs and Subroutines 226
CRSPAccess FORTRAN-95 Library 228

CHAPTER 1: OVERVIEW

The supplied suite of CRSP utilities allows full-featured access to CRSP databases and is intended to cover a variety of the most typical queries and uses of CRSP data. The features of CRSP tools can often save end-users the whole effort of programming their own reporting utilities. In other cases the user-program flow can be significantly simplified by the use of output files produced from CRSP tools in a number of widely accepted formats.

OVERVIEW OF CRSPACCESS DATABASES

A CRSPAccess database is a customized financial database system supporting time-series, event, and header data for various financial data structures. A single CRSPAccess Database is a set of defined configuration and module files in a directory. Configuration files track the location of data in the module files.

The basic levels of a CRSPAccess database are:

1. Database (`CRSPDB`) is the directory containing the database files. A CRSPDB is identified by the database path.
2. Set Type is a predefined type of financial data; stock or indexes. Each set type has its own defined set of data structures, specialized access functions, and keys. CRSPAccess databases support stock (`STK`) and index (`IND`) set types. A CRSPDB can support multiple set types.
3. Set Identifier (`SETID`) is a defined subset of a set type. `SETID`s of the same set type use the same access functions, structures, and keys, but have different characteristics within those structures. For example, daily stock sets use the same data structure as monthly stock sets, but time series are associated with different cal- endars. Multiple SETIDs of the same set type can be present in one `CRSPDB`.
4. Modules are the groupings of data found in the data files in a `CRSPDB`. Multiple data items can be present in a module. Data are retrieved at a module level, and access functions retrieve data items for keys based on selected modules. A module corresponds to a single physical data file.
5. Objects are the fundamental data types defined for each set type. There are three fundamental object types: time series (`CRSP_TIMESERIES`), event arrays (`CRSP_ARRAY`), and headers (`CRSP_ROW`). Objects con- header information such as counts, ranges, or associated calendars, plus arrays of data for zero or more observations. Some set types allow arrays of objects of one type. In this case, the number of available objects is determined by the `SETID`, and each of the objects in the list has independent counts, ranges, or associated calendars.
6. Arrays are attached to each object. The array contains the set of observations and is the basic level of pro- gramming access. An observation can be a simple data type, such as an integer for an array of volumes, or a complex structure such as for a name history. When there is an array of objects, there is a corresponding array of arrays with the data.

Configuration Files contain information about supported sets and modules in the `CRSPDB`, a list of keys, addresses of data for each key in different data modules, a set of shared calendars, a set of secondary indexes, and a list of free space within the module files. Module files contain the data for groups of objects for keys.

MISSING VALUES

Missing values are relevant for time-series with scalar data type elements. Scalar data types are predefined data types used in C and Fortran. Examples include integer, floating point, real, logical, double precision. Missing values are not meaningful for arrays, C-LANGUAGE structures, and Fortran-95 TYPES; values used in these cases are merely place-holders.

For all time-series, missing values are stored at index zero. Index values 1 through MAX contain meaningful values of the time-series. These may be compared against the missing value at index 0 to determine whether they are miss- ing.

A C application programming interface supports access to defined set structures, such as stock security or index data. A FORTRAN-95 programming interface is also built into the system. The FORTRAN-95 access utilizes direct access by module and by key.

API LIBRARY

For situations when a user-program requires direct access to CRSP datasets, a CRSP API library is provided. The CRSP API comprises the platform-specific object library and a set of precompiled modules and include files. Additionally, a set of sample program sources is supplied along with respective make files and test data to build and test the samples.

ITEM-BASED ACCESS

The CRSP API introduces item-based access which generally supersedes the older set-based method of programming access to CRSP databases.

Conceptually, the item-based access allows uniform access to the whole array of CRSP datasets by presenting them as collections of items that are instances of a generic item object. Each specific CRSP dataset is represented as an instance of a generic access-handle object.

In item-access context users are shielded from the internal mechanics of access to the specific structures of CRSP datasets. This adds more uniformity to your code, especially in cases of accessing a mixed set of CRSP data, thus simplifying code development and maintenance.

In addition to its dynamic and extensible nature, item-access also introduces a set of derived items that are defined as parameterized functional combinations of underlying regular items. These items significantly improve code consistency and simplicity by internally performing all of the necessary calculations that previously had to be coded explicitly in user programs.

LEGACY SET-BASED ACCESS

The supplied CRSP API contains the needed underlying functions for set-based access. However, mixing the legacy set-based access and item-access contexts in the same program is not recommended. Any new user programs that access CRSP data should be implemented in the item-access context. CRSP supplies a set of sample programs that demonstrate the flow of programs using item-access.

NOTATIONAL CONVENTIONS

- All names that occur within CRSP's FORTRAN-95 and C sample programs and include files are printed using a `constant-width, Courier` font. These names include variable names, parameter names, subroutine names, subprogram names, function names, library names, and keywords. For example, CUSIP refers to the CUSIP Agency identifier, while `CUSIP` refers to the variable that the programs use to store this identifier. CRSP's variable mnemonics, used as names and in the descriptions, are displayed capitalized using a `constant-width` font. C and FORTRAN-95 are displayed in `lower case`, excepting constants, which are displayed in `UPPER CASE`.
- All names that refer to the CRSP data utilities, sample programs or include file titles are printed using an italic sans serif font.
- Names with a similar format are sometimes referenced collectively, using three x's where the names differ. For example, the FORTRAN-95 variables `BEGVOL, BEGRET, BEGPRC`, etc. are sometimes referred to as `BEGxxx`.
- In the variable definitions section, the variables i and j are sometimes used in referencing a variable in a FOR- TRAN-95 or C array. In this case, i refers to a possible range of valid data in this array for this company, where the valid range is determined by the number of header variables. For example, in FORTRAN-95, the names date is referred to as stk % names_arr % names(i) % namedt. Here i is an integer between 1 and stk % names_arr % num, which represents the number of name structures that exist for any specified issue in the CRSP US Stock Database.
- In C, all CRSP-defined data types have names in all capitals beginning with CRSP_.

CHAPTER 2: ITEM-BASED ACCESS IN C

INTRODUCTION

The supplied suite of CRSP utilities allows full-featured access to CRSP databases and is intended to cover a variety of the most typical queries and uses of CRSP data. The features of CRSP tools can often save end-users the whole effort of programming their own reporting utilities. In other cases the user-program flow can be significantly simplified by the use of output files produced from CRSP tools in a number of widely accepted formats.

For situations when a user-program requires direct access to CRSP datasets, a CRSP API library is provided. The CRSP API comprises the platform-specific object library and a set of precompiled modules and include-files. Additionally, a set of sample program sources is supplied along with respective make files and test data to build and test the samples.

CRSP C API DATA OBJECTS

Access to CRSP databases is achieved through two principal objects: the access handle – of type `CRSP_ITM_HNDL`, and the item – of type `CRSP_ITM`.

CRSP_ITM_HNDL

The item-access handle object `type(CRSP_ITM_HNDL)` encapsulates the information required to establish and maintain a single item-access session to a given CRSP database. Additional access sessions (either to the same or to another CRSP database), concurrent in the same program, require a separate access handle object. All of the item objects available in the active session are grouped within the respective access handle.

The main properties of the access handle object are listed on the following table:

NAME	C TYPE	DESCRIPTION
keytype	character(LEN=CRSP_NAME_LEN)	Determines the keys used to select data in read functions. Supported keytypes for the application are included in the reference data. A default will be set.
keyset_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether keyset items are labeled by the keyset number (NUM), the keyset tag (TAG), or the expanded list of all items comprising the keyset (EXP). The default display is TAG.
fiscal_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether fiscal-based time series items are reported on a calendar basis (C) or a fiscal basis (F). The default it C.
curr_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether monetary values are reported in the currency reported by Compustat (REP) or in US Dollars (USD). The default currency display code is REP.
grp_fill_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether group item lists are filled so that every selected item is included for every selected keyset (Y or N). The default is Yes (Y).
dataset	CRSP_ITM_SET	Pointer to descriptor of currently attached CRSP dataset; includes root info for the CRSP dataset.

In a user-program the access handle objects are normally declared and allocated directly then passed to C itm-API functions as a parameter. The function `crsp_itm_init` initializes the contents of the access handle and connects it to the specified CRSP database.

CRSP_ITM

The item object `type(CRSP_ITM)` represents a generic container for a single data item defined in a given CRSP database. It unifies the data types defined for each of the supported CRSP databases and allows uniform access to the associated CRSP data containers from your programs.

The main properties of the item object are listed in the following table:

NAME	C TYPE	DESCRIPTION
itm_name	character(LEN=CRSP_NAME_LEN)	name of the item from a CRSP dataset.
keyset	integer	number of the keyset defined in a CRSP dataset.
itm_info	CRSP_ITM_INFO	item metadata; includes description, default keyset, and stored data type.
obj	CRSP_ITM_OBJ	describes the underlying CRSP data-object.
arr	CRSP_ITM_OBJARR	describes the C container associated with the defined CRSP data-object.
itmkeyset	CRSP_ITM_KEYSET	describes the details of the keyset (when non-zero and loaded), including its number, name, tag, and array of composing items of same CRSP_ITM type.
itmcal	CRSP_ITM_CAL	for calendar-bound items, describes the details of the attached calendar, including its id, keyset ,frequency, and attached calendar object of CRSP_CAL type. When requested, the calendar may be 'shifted', based on the currently loaded company's FYE to attribute properly the item's period data.

Item objects are normally declared as C pointers and then attached to the actual defined item objects by calling the `crsp_itm_find` function for the given access handle and the specified item name and keyset.

CRSP_ITM_OBJ

Item data is accessed from the data-object `itm->obj` and associated to a C container `itm->arr`. The item data container object type (CRSP_ITM_OBJ) describes an instance of a CRSP data-object (time-series, array, row) that is defined for the specific item. Only a single data-object can be defined for a given item, which is identified by the `objtype` property.

Properties of the item data-object are listed in the following table:

NAME	C TYPE	DESCRIPTION
objtype	integer	type of the defined and allocated object: CRSP_TS_OTID: CRSP time-series CRSP_ARRAY_OTID: CRSP array CRSP_ROW_OTID: CRSP row
ts	CRSP_TS	pointer to allocated CRSP time-series data-object.
arr	CRSP_ARRAY	pointer to allocated CRSP array data-object.
row	CRSP_ROW	pointer to allocated CRSP row data-object.
is_empty	logical	indicates whether the allocated CRSP data-object contains no data.

The item data-object normally has an associated C container, which is either C array or scalar of the data type corresponding to the actual stored data, as identified by `arrtype` property. Details of the CRSP container objects types are listed in the reference section CRSP Container Objects.

CRSP_ITM_OBJARR

The item data array, type `CRSP_ITM_OBJARR` describes the associated C container object. The C container is allocated based on the object's type (`objtype`) and contained data type (`arrtype`). The respective scalar member has suffix _val to its name, and _arr for the array type. Time-series and array data are stored in array type, while row data is kept in scalar type:

`Itm->arr->arrtype:`

`CRSP_TS_OTID: itm->arr-><arrtype_name>_arr` - time-series object data array

`CRSP_ARARY_OTID: itm->arr-><arrtype_name>_arr` - array object data array

`CRSP_ROW_OTID: itm->arr-><arrtype_name>_val` - row object data scalar.

NOTE: Throughout the implementation of the CRSP C API, the C array indexing is **0-based**, thus the first element of an array is

`data_arr(0)`.

Properties of the item data array for the item data types that are common to all of the supported CRSP datasets are listed in the following table:

NAME	C TYPE	DESCRIPTION
arrtype	integer	data type of the defined and allocated C container. Common data types: • CRSP_INTEGER_TID: integer • CRSP_FLOAT_TID: real • CRSP_DOUBLE_TID: double precision • CRSP_CHAR_TID: character(1) • CRSP_CHARACTER_TID: CRSP_VARSTRING type
int_val / arr	integer / dimension (:)	pointer to allocated scalar / array of integer type
flt_val / arr	real / dimension (:)	pointer to allocated scalar / array of real type
dbl_val / arr	double precision / dimension (:)	pointer to allocated scalar / array of double precision type
char_val / arr	character(LEN=1) / dimension (:)	pointer to allocated scalar / array of single-character type
vstr_val / arr	CRSP_VARSTRING / dimension (:)	pointer to allocated scalar / array of variable-length string type
structured types specific to CRSP datasets	Refer to the description of data types for the specific CRSP dataset.	

In a user-program the item container data is usually accessed directly as defined by item's data type, e.g.:

```
print *, sale_itm->arr->dbl_arr(i)
```

The item data container is normally accessed in association with its item data-object.

NOTE: If an incorrect data container is referenced, an access violation error should occur on the referenced null-pointer. In such situations the recommended action is to verify that the appropriate containers are being accessed for the selected items.

Data for items of CRSP array type is accessed in the valid `[0..num-1]` index range, as defined in the corresponding `arr` data-object. For example:

```
itm->arr->dbl_arr(i), i=0..itm->obj->arr->num-1
```

Data for items of CRSP time-series type is accessed in the valid `[beg,end]` index range, as defined in the corresponding ts data-object, e.g.:

```
itm->arr->dbl_arr(i), i=itm->obj->ts->beg..itm->obj->ts->end
```

Data for items of CRSP row type is not indexed and is accessed directly from the value as defined by the corresponding scalar/ structured type, e.g.:

```
itm->arr->master_val->ccmid
```

To verify if an element of an array item contains a missing value, call the function `crsp_itm_is_miss_arrval`.

SUPPORTING INFORMATION

Various supporting information about CRSP databases, items, keysets and other item-access objects is stored in the following derived types:

C TYPE NAME	DESCRIPTION	ACCESS VIA TYPE NAME	USAGE
CRSP_ITM_INFO	Item information; includes item's full name, description, display format, data type and size information. Also includes the default keyset number associated with this item.	CRSP_ITM	itm->itm_info
CRSP_ITM_KEYSET	Keyset descriptor; includes keyset information and the array of items composing the keyset.	CRSP_ITM	itm->itmkeyset
CRSP_ITM_CAL	Calendar descriptor; includes calendar's id, associated keyset number, base calendar name, and calendar's frequency, also the base calendar object. Additionally, for fiscal calendars indicates whether the calendar has been shifted based on the currently loaded company's FYE.	CRSP_ITM	itm->itmcal
CRSP_KEYSET	Keyset information; includes keyset's number, name, tag, and description. Indicates whether the keyset has been loaded and associated with any of the requested items.	CRSP_ITM_KEYSET	itmkeyset->keyset_info
CRSP_ITM_SET	CRSP data set descriptor; includes the set's path, name, id, and database root information.	CRSP_ITM_HNDL	hndl->dataset
CRSP_ROOT_INFO	CRSP data set root information; includes internal service information about the currently loaded database such as creation/modification date, product code and name, and descriptors of available calendars. Mainly intended for internal use.	CRSP_ITM_SET	dataset->root_info

NOTE: While selected supported information is populated on initiating of the connection to a CRSP data set (on return from call to `crsp_itm_init`), the listed supported information becomes available only on opening of the CRSP data set (on return from call to `crsp_itm_open`).

The relevant details of the derived types shown above are listed in the *Supporting Types* on page 25.

GENERIC DATA TYPES

All CRSP databases contain data items of both simple C data types and of database-specific structured data types. Moreover, each composing field of the structured data type can instead be requested as an individual data item of the simple C data type.

The vast majority of the data items defined in CRSP datasets are of CRSP time-series container object type, with the stored values commonly of generic C data types. A limited set of items is stored in CRSP array and CRSP row container objects; these items are mostly of structured data types and are listed in the following sections regarding particular CRSP database products.

The following table lists the supported generic data types and ways to access data from the item-associated container:

ITEM OBJECT TYPE	C TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ITM
time-series	CRSP_TS		itm->obj->ts
	int	int(4)	Itm->arr->int_arr(i)
	float	float(4)	Itm->arr->flt_arr(i)
	double	double(8)	Itm->arr->dbl_arr(i)
	char(LEN=1)	char (1)	Itm->arr->char_arr(i)

ITEM OBJECT TYPE	C TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ ITM
array	CRSP_ARRAY		itm->obj->arr
	int	int(4)	Itm->arr->int_arr(i)
	float	float(4)	Itm->arr->flt_arr(i)
	double	double(8)	Itm->arr->dbl_arr(i)
	char(LEN=1)	char (1)	Itm->arr->char_arr(i)
row	CRSP_ROW		itm->obj->row
	int	int(4)	Itm->arr->int_val
	float	float(4)	Itm->arr->flt_val
	double	double(8)	Itm->arr->dbl_val
	char(LEN=1)	char (1)	Itm->arr->char_val

ACCESSING CRSP DATABASES

The following sections describe the details of accessing CRSP databases supported by the API. Supported databases are the CRSP US Stock Database, the CRSP US Index Database, and the CRSP/Compustat Merged Database. Each section presents database connection information, available access keys, as well how to access a database's data groups and items from your programs.

CRSP US STOCK DATABASE

To connect to the specific CRSP Stock database instance the path to its database root should be specified. When installed on your system, CRSP Stock data set will be assigned an environment variable pointing to the CRSP Stock database root.

Additionally, an application ID should be specified on the call to `crsp_itm_init` to indicate the item-universe to be loaded for the session and describes the available items and item groups, e.g.:

```
sts = crsp_itm_init (hndl,'CRSP_DSTK',app_id,'stk1')
```

User-programs should access the CRSP Stock data set with the `app_id` as listed in the following table:

STK ROOT/APP ID	C TYPE	DESCRIPTION
CRSP_DSTK		CRSP Daily Stock data set
CRSP_DSTKITM_ID	integer	CRSP Daily Stock data items and groups
CRSP_MSTK		CRSP Monthly Stock data set
CRSP_MSTKITM_ ID	integer	CRSP Monthly Stock data items and groups

The details on included items and item groups can be found starting on page 13.

ACCESS KEYS

CRSP Stock data set contains various data on companies and securities. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to STK data.

Default access key is loaded automatically on opening the access session to the CRSP STK data set

Additionally, a set of alternative access keys (and associated key items) is defined to facilitate access to the data by CRSP PERMNO, CUSIP, and other keys.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_itm_get_key` can also be used to retrieve the value of the access key items for

the currently read record.

To switch to access by an alternative key, a user calls `crsp_itm_load_key` to set the access key index, followed by calls to `crsp_itm_set_key` to set the value of the key items used on subsequent reading of the database.

The defined STK access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	C TYPE	DESCRIPTION	NOTES
PERMNO		CRSP historical PERMNO	default
KYPERMNO	Integer	CRSP company issue's PERMNO	primary key item
PERMCO		CRSP historical PERMCO	
KYPERMCO	Integer	CRSP company's PERMCO	primary key item
CUSIP		CRSP Stock CUSIP	
KYCUSIP	char(CRSP_CUSIP_LEN)	CRSP Stock issue's CUSIP	primary key item
HCUSIP		CRSP Stock Historical CUSIP	
KYHCUSIP	char(CRSP_CUSIP_LEN)	CRSP issue's Historical CUSIP	primary key item
Ticker		CRSP Stock ticker	
KYTICKER	char(CRSP_STK_TIC_LEN)	CRSP issue's ticker	primary key item
SICCD		CRSP Stock SIC code	
KYSIC	Integer	CRSP Stock security's SIC	primary key item

DATA TYPES

Generally, individual CRSP Stock database data items are of common simple C data types and stored data can be accessed through `itm->arr` and corresponding scalar or array member.

Additionally, selected CRSP supplemental STK data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by `itm_name`, but recommended programming access is through the `itm_name` of the structure. To access the structured type and its fields, load the structured type `itm_name` during initialization, create a `CRSP_ITM` pointer matching the `itm_name`, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_itm_load(hndl,'HEADER',match_flag)

sts = crsp_itm_find(hndl,'HEADER',0,header_itm)

permno = header_itm->arr->header_val->permno

...
```

CRSP US INDEX DATABASE

To connect to the specific CRSP Index database instance the path to its database root should be specified. When installed on your system, CRSP Index data sets will be assigned an environment variable pointing to the CRSP Index database root.

Additionally, an application ID should be specified on the call to `crsp_itm_init` to indicate the item-universe to be loaded for the session and describes the available items and item groups, e.g.:

```
sts = crsp_itm_init (hndl,'CRSP_DSTK',app_id,'ind1')
```

User-programs should access the CRSP Index data sets with the `app_id` as listed in the following table:

STK ROOT/APP ID	C TYPE	DESCRIPTION
CRSP_DSTK		CRSP Daily Stock and Index data sets
CRSP_ DINDITEMS_ID	integer	CRSP Daily Index series data items and groups
CRSP_ DINDGITEMS_ID	integer	CRSP Daily Index group data items and groups
CRSP_MSTK		CRSP Monthly Stock and Index data sets
CRSP_ MINDITEMS_ID	integer	CRSP Monthly Index series data items and groups
CRSP_ MINDGITEMS_ID	integer	CRSP Monthly Index group data items and groups

ACCESS KEYS

CRSP Index data sets include various data on market indexes. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to IND data.

Default access key is loaded automatically on opening the access session to the CRSP IND data set.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_itm_get_key` can also be used to retrieve the value of the access key items for the currently read record.

The defined IND access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	C TYPE	DESCRIPTION	NOTES
indno		CRSP Index's INDNO	default
KYINDNO	integer	CRSP index's INDNO	primary key item

DATA TYPES

Generally, individual CRSP Index database data items are of common simple C data types and stored data can be accessed through `itm->arr` and corresponding scalar or array member.

Additionally, selected CRSP supplemental IND data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by itm_name, but recommended programming access is through the itm_name of the structure. To access the structured type and its fields, load the structured type itm_name during initialization, create a CRSP_ITM pointer matching the itm_name, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_itm_load(hndl,'INDHDR',match_flag)

sts = crsp_itm_find(hndl,'INDHDR',0,indhdr_itm)

indno = indhdr_itm->arr->indhdr_val->indno

...
```

CRSP/COMPUSTAT MERGED DATABASE

To connect to the specific CRSP CCM database instance the path to its database root should be specified. When installed on your system, CRSP CCM data set will be assigned an environment variable pointing to the CRSP CCM database root.

Additionally, an application ID should be specified on the call to `crsp_itm_init` to indicate the item-universe to be loaded for the session and describes the available items and item groups, eg:

```
sts = crsp_itm_init (hndl,'CRSP_CCM',app_id,'ccm1')
```

User-programs should access the CRSP CCM data set with the `app_id` as listed in the following table:

CCM ROOT/APP ID	C TYPE	DESCRIPTION
CRSP_CCM		CCM/CRSP Compustat data set
CRSP_CCMITEMS_ID	integer	Compustat Xpressfeed data items and groups

The details on included items and item groups can be found starting on page 17.

ACCESS KEYS

CRSP Compustat Xpressfeed includes various data on companies, securities, and indexes. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to the CCM data.

Default access key for CRSP CCM is loaded automatically on opening the access session to the CRSP CCM data set

Additionally, a set of alternative access keys (and associated key items) is defined to facilitate access to the data by CRSP PERMNO, CUSIP and other keys.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_itm_get_key` can also be used to retrieve the value of the access key items for the currently read record.

To switch to access by an alternative key, a user calls `crsp_itm_load_key` to set the access key index, followed by calls to `crsp_itm_set_key` to set the value of the key items used on subsequent reading of the database.

The defined CCM access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	C TYPE	DESCRIPTION	NOTES
gvkey		Compustat GVKEY and IID	default
KYGVKEY	integer	Compustat company's GVKEY	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
gvkeyx		Compustat permanent identifier for indexes	
KYGVKEYX	integer	Compustat index's GVKEYX	primary key item
ccmid		Compustat permanent identifier - either GVKEY or GVKEYX	
KYCCMID	integer	CRSP CCMID (GVKEY or GVKEYX as reported in MASTER item)	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
permco		CRSP historical PERMCO Link	
KYPERMCO	integer	CRSP company's PERMCO	primary key item
permno		CRSP historical PERMNO Link	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item
cusip		Compustat CUSIP	
KYCUSIP	char(CRSP_CCM_CUSIP_LEN)	Compustat issue's CUSIP	primary key item
ticker		Compustat reported Issue Trading Symbol selects GVKEY and security	

CCM ACCESS KEY/KEY ITEMS	C TYPE	DESCRIPTION	NOTES
KYTICKER	char(CRSP_CCM_TIC_LEN)	Compustat issue's ticker	primary key item
sic		Compustat -reported SIC code. Security or Company	
KYSIC	integer	Compustat security's SIC	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
apermno		Link-Used PERMNO	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item
apermco		Link-Used PERMCO	
KYPERMCO	integer	CRSP company issue's PERMCO	primary key item
ppermno		CRSP PERMNO when security is marked as primary	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item

DATA TYPES

Generally, individual Compustat Xpressfeed data items are of common simple C data types and stored data can be accessed through `itm%arr` and corresponding scalar or array member.

Also additional character data types were introduced to store specific classes of Xpressfeed items, as listed in the following table:

ITEM OBJECT TYPE	C TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ITM	NOTES
time-series	CRSP_TS		Itm->obj->ts	
	CRSP_CCM_FTNT	char(CRSP_CCM_FTNT_LEN)	itm->arr->ftnt_arr(i)->ftnt	Used for CCM footnote items, mainly time-series
	CRSP_CCM_TEXTITEM	char(CRSP_CCM_TEXTITEM_LEN)	itm->arr->text_arr(i)->text	Used for various CCM character string items, mainly time-series
array	CRSP_ARRAY		itm->obj->arr	
	CRSP_CCM_FTNT	char(CRSP_CCM_FTNT_LEN)	itm->arr->ftnt_arr(i)->ftnt	
	CRSP_CCM_TEXTITEM	char(CRSP_CCM_TEXTITEM_LEN)	itm->arr->text_arr(i)->text	
row	CRSP_ROW		itm->obj->row	
	CRSP_CCM_FTNT	char(CRSP_CCM_FTNT_LEN)	itm->arr->ftnt_val->ftnt	
	CRSP_CCM_TEXTITEM	char(CRSP_CCM_TEXTITEM_LEN)	itm->arr->text_val->text	

Additionally, selected Compustat Xpressfeed primary data groups and CRSP supplemental data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by `itm_name`, but recommended programming access is through the itm_name of the structure. To access the structured type and its fields, load the structured type `itm_name` during initialization, create a `CRSP_ITM` pointer matching the `itm_name`, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_itm_load(hndl,'MASTER',match_flag)
```

```
sts = crsp_itm_find(hndl,'MASTER',0,mstr_itm)

ccmid = mstr_itm->arr->master_val->ccmid

...
```

CRSP C API ITEM HANDLING FUNCTIONS

This section contains an alphabetical list of the functions defined in the CRSP C API. Each definition presents the following information about a function:

- Its prototype
- A list of arguments
- A list of return values
- Side effects
- Preconditions

CRSP_ITM_CLOSE

`crsp_itm_close` frees all item lists and item indexes, clears all calendar and key lists, closes the database, frees the handle set, and re-initializes the item access handle itself.

PROTOTYPE:	`int crsp_itm_close(CRSP_ITM_HNDL **hndl)`
ARGUMENTS:	`CRSP_ITM_HNDL *hndl`: Access handle to close.
RETURN VALUES:	CRSP_SUCCESS: If the database is successfully closed and all handle data are free CRSP_FAIL: If there is an error in the parameters, inconsistent handle, error closing databases.
SIDE EFFECTS:	If successful, the handle data are emptied: The database will be closed and the structure cleared. All internal storage allocated for this instance will be freed
PRECONDITIONS:	The item handle must be previously opened with function `crsp_itm_init`.

Example:

```
if (crsp_itm_close(&hndl) == CRSP_FAIL)
{
    fprintf (stderr, err_msg);
    exit (EXIT_FAILURE);
}
```

CRSP_ITM_FIND

`crsp_itm_find` attaches a pointer to a `CRSP_ITM` item that was previously loaded. The `CRSP_ITM` structure describes the data item and contains the underlying time series, array, or row data.

PROTOTYPE:	`int crsp_itm_find(CRSP_ITM_HNDL *itmhndl,char *itm_name, int keyset, CRSP_ITM **foundptr);`

ARGUMENTS:	`CRSP_ITM_HNDL_T * hndl`:Access handle containing the needed set structure information and the current item list. `char *itm_name`:String containing the `itm_name` to find. `Int keyset`:Keyset to find `CRSP_ITM *itm_foundptr`:User-declared pointer that will point to the data item found.
RETURN VALUES:	CRSP_SUCCESS: If successfully found the requested item in the given keyset. CRSP NOT FOUND: If the `itm_name` and keyset combination are not available CRSP_FAIL: If error in parameters, handle not initialized, or error searching for the item.
SIDE EFFECTS:	If successful, the `itm_foundptr` will point to a `CRSP_ITM` item with data and information for the desired item and keyset. Otherwise the null() will be assigned to `itm_foundptr`.
PRECONDITIONS:	The item handle set must be initialized, loaded with a list of items, and opened.

Example:

```
if (crsp_itm_find(hndl,"HEADER",0,&stkhdr_itm) == CRSP_FAIL || !stkhdr_itm)
{
    fprintf (stderr,"Error - invalid item/keyset specified(DSTK:1)\n");
    exit(EXIT_FAILURE);
}
```

CRSP_ITM_GET_KEY

`crsp_itm_get_key` retrieves key information for data loaded by a function `crsp_itm_read` call. An output key item list is prepared when the key is initialized, and loaded by function `crsp_itm_read`. This function finds the `key_itm_name` in the list and copies the value into the user-specified location.

PROTOTYPE:	`int crsp_itm_get_key(CRSP_ITM_HNDL *hndl, char *key_itm, void *keyval);`
ARGUMENTS:	`CRSP_ITM_HNDL_T *hndl`:Access handle containing the needed set structure information and the current item list `Char *key_itm_name`:String containing an `itm_name` of a loaded key to be retrieved. `Void *keyval`: Variable to accept the value of the key item. Data type must agree with the item's type and size.
RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_FAIL: If error in parameters, handle not open, key item.
SIDE EFFECTS:	If successful, the `keyval` is loaded based on the item and key value type.
PRECONDITIONS:	The item handle must be initialized and opened. The item key array must be initialized based on a keytype with the function `crsp_itm_open` or function `crsp_itm_init_key` functions. The `key_itm_name` must be a valid item for that keytype, and the `keyval` data must agree with the type of that item.

Example:

```
if ( crsp_itm_get_key(hndl, "KYPERMNO", keyval) == CRSP_FAIL )
    {
        crsp_errprintf(2, 50, CRSP_FATAL_PRINT,
                CRSP_ERROUT_STDERR," hndl permno, crsp_itm_get_key");
        exit(EXIT_FAILURE);
```

```
}
```

CRSP_ITM_INIT

`crsp_itm_init` prepares a handle for item access operation for one database and one application id. The handle will be initialized and the database set type and set id identified, allowing loading of reference data and allocation of a set structure.

PROTOTYPE:	`int crsp_itm_init(CRSP_ITM_HNDL **hndl, char *dbpath, int app_id, char *hndl_name);`
ARGUMENTS:	`CRSP_ITM_HNDL *hndl:` Access handle that will be used to manage the database information and item lists. `Char *dbpath:` Path to database containing the data to load and the applicable reference data. `Int app_id:` Identifier of a defined application organizing data items into groups for access. Available `app_ids` can be found in the reference array, function `crsp_itm_` app. Common `app_ids` have defined constants: • `CRSP_CCMITEMS_ID` – generic CCM usage application • `CRSP_DSTKITM_ID` – generic Daily Stock usage application • `CRSP_MSTKITM_ID` – generic Monthly Stock usage application • `CRSP_DINDITEMS_ID` – generic Daily Ind Stock usage application • `CRSP_MINDITEMS_ID` – generic Monthly Ind Stock usage application • `CRSP_SIZITM_ID` – generic SIZ 1925-E usage application `Char *hndl_name:` Name to assign to the handle.
RETURN VALUES:	CRSP_SUCCESS: If initialized successfully CRSP_FAIL: If there is an error in the parameter, database cannot be opened, reference data unavailable, incompatibility between database and `app_id`.
SIDE EFFECTS:	If successful, the handle data are loaded: • The handle fields are initialized, including all lists and arrays. • The `ca_ref` structure is loaded with the reference data in the database. If an old database with no reference data, it will use a global reference file with a standard name based on the `app_id` in the `CRSP_LIB` directory. • `Itm_grp` and `itm_avail` arrays in the handle are loaded with available tables and items • `Set_list` element is allocated using the database path and `setid`. The database is opened with a 0 wanted, which loads reference data but allocates no module space. The root information is loaded to the set's `CRSP_ROOT_INFO` structure.
PRECONDITIONS:	The item handle object must be declared and not attached to another access instance. The `app_id` must exist in the reference data of the database opened.

Example:

```
if (crsp_itm_init(&hndl,dbpath,CRSP_SIZITM_ID,"siz1") == CRSP_FAIL)
{
    fprintf (stderr,err_msg);
    exit (EXIT_FAILURE);
}
```

CRSP_ITM_IS_MISS_ARRVAL

`crsp_itm_is_miss_arrval` checks if the requested element in a data object attached to the item contains a missing value. `is_miss` is set to TRUE when a missing value is detected. Only items of simple (non-structured) types are accepted, while the item's underlying data-object can be of structured data-type, in which case the structure offset is used to extract the item value.

PROTOTYPE:	`int crsp_itm_is_miss_arrval(CRSP_ITM *itm,int ind, int *is_miss);`

ARGUMENTS:	`CRSP_ITM *itm`: Pointer to the data item `Int ind`: Index of the data array element to check `Int *is_miss`: Pointer to the resulting flag value
RETURN VALUES:	CRSP_SUCCESS: If successful, the returned value is initialized and set. CRSP_FAIL: If error in parameters, bad item or element index is out-of-range (ignored in case of `CRSP_ROW` object)
SIDE EFFECTS:	If the requested value contains a missing value, the `is_miss` is set to TRUE. Otherwise FALSE is assigned.
PRECONDITIONS:	The item has to have a valid bound data-object. Structured items are not allowed. Field items of structures are allowed.

Example:

```
 if (crsp_itm_is_miss_arrval(itm,ind,&is_miss) != CRSP_FAIL && !is_miss)
{
    fprintf (ofp,"%18.4f",itm->arr.dbl_arr[ind]);
}
else
{
    fprintf (ofp,"%18s","N/A");
}
```

CRSP_ITM_LOAD

`crsp_itm_load` prepares items described by a full list and loads them to an item table structure in an item handle. It splits the full list into the global section and the list section and uses the function `crsp_itm_expand_elem` on each list element in the list section. This will recursively expand the list elements to fill the structure and apply global qualifiers during the process.

PROTOTYPE:	`int crsp_itm_load(CRSP_ITM_HNDL *hndl, char *itm_str, int match_flag);`
ARGUMENTS:	`CRSP_ITM_HNDL_T * hndl:` Access handle containing the needed set structure information and the current item list. `Char *full_list`: String describing all items to add, used on standard item notation. `Int match_flag`: Flag setting the behavior when an item if found but not the keyset. Values are: • `CRSP_MATCH_REQUIRED` (=0): if any indicated item and keyset is not found no further items will be added, and CRSP_ NOT_ FOUND returned. • `CRSP_MATCH_FILL` (=1): a dummy item will be created for any item if the item exists but the keyset does not exist for that item in the current database. • `CRSP_MATCH_IGNORE` (=2): items will not be added if the keyset is not found, but the return remains CRSP_SUCCESS.
RETURN VALUES:	CRSP_SUCCESS: If successful, and all indicated items loaded according to `match_flag` CRSP_FAIL: Error in parameters, bad list, handle not initialized, or reference data not available.
SIDE EFFECTS:	If successful, the `CRSP_ITM_GRP` is loaded with all indicated items. A `CRSP_ITM` is allocated for each item/keyset pair not already loaded. Object pointers are not set by this function.
PRECONDITIONS:	The item handle set must be loaded. The item table must be initialized with an available `app_id`. The first set in the set list must agree with the `app_id`.

Example:

```
if ((status=crsp_itm_load(hndl,"STKHDR_ALL;DSTK_TS", CRSP_MATCH_IGNORE)) == CRSP_FAIL
```

```
        || status == CRSP_NOT_FOUND)
  {
      fprintf (stderr,err_msg);
      exit (EXIT_FAILURE);
  }
```

CRSP_ITM_LOAD_KEY

`crsp_itm_load_key` defines the keytype that will be used for subsequent reads.

PROTOTYPE:	`int crsp_itm_load_key(CRSP_ITM_HNDL *hndl, char *ketytype);`
ARGUMENTS:	`CRSP_ITM_HNDL *hndl:` Access handle containing the needed set structure information and the current item list. `Char *keytype:` Name of the key to initialize. Values are: • `gvkey`: Compustat company key (default) • `gvkeyx`: Compustat index key • `ccmid`: GVKEY or GVKEYX • `permno`: CRSP PERMNO found in any links • `permco`: CRSP PERMCO found in any links • `apermno`: CRSP-centric composite records by PERMNO • `ppermco`: CRSP-centric composite records by PERMNO, primary links only • `sic`: Compustat company SIC code • `ticker`: Compustat security ticker symbol • `cusip`: Security CUSIP
RETURN VALUES:	CRSP_SUCCESS: If successful CRSP_FAIL: Error in parameters, handle not initialized, or keytype not found.
SIDE EFFECTS:	If successful, the handle is prepared to handle reads.
PRECONDITIONS:	The item handle must be initialized. Keytype must be known for the `app_id`.

Example:

```
if (crsp_itm_load_key(hndl,"PERMNO") == CRSP_FAIL)
  {
      fprintf (stderr,err_msg);
      exit (EXIT_FAILURE);
  }
```

CRSP_ITM_OPEN

`crsp_itm_open` registers selected items in a handle by expanding structures and keysets, preparing keys, determining modules needed to access items, opens the needed modules, and binds data in the item lists to the data structure locations. It also builds a master index of all items available in the handle.

PROTOTYPE:	`int crsp_itm_open(CRSP_ITM_HNDL *hndl);`
ARGUMENTS:	`CRSP_ITM_HNDL *hndl:` Access handle containing the needed set structure information and the current item list.
RETURN VALUES:	CRSP_SUCCESS: If opens successfully and binds the data CRSP_FAIL: If error in parameters, inconsistent handle, error opening databases or binding items.

SIDE EFFECTS:	If successful, the handle is ready for access: • All items in the loaded list will have object pointer set to the data location in the set data structure. • If the handle `grp_fill_cd` is 'Y', then the item lists are filled to ensure full tables. Filling creates items to ensure that every `itm_name` and keyset present in a group each combination is present even if not specified. Filling also arranges the lists so if multiple keysets, each is sorted in the same order as the first keyset seen.
PRECONDITIONS:	The item handle must be previously initialized with function `crsp_itm_init`. It generally follows one or more instances of item load function calls.

Example:

```
if (crsp_itm_open(hndl) == CRSP_FAIL)
 {
     fprintf (stderr,err_msg);
     exit (EXIT_FAILURE);
 }
```

CRSP_ITM_READ

`crsp_itm_read` loads data from handle based on item keys specified in prior function `crsp_itm_key` calls and the keyflag argument. Depending on the level of the entity class, the operation may include reading data from the database into structures and/or specifying data already loaded. This allows a direct or positional read based on keyflag.

If the value of the access handle property `fiscal_disp_cd` is "C", any fiscal-based time series are shifted to a calendar basis as part of the read operation.

PROTOTYPE:	`int crsp_itm_load(CRSP_ITM_HNDL *hndl, char *itm_str, int match_flag);`

ARGUMENTS:	`CRSP_ITM_HNDL * hndl: Access handle containing the needed set structure information and the current item list.`
	`Char *itm_str: Code determining how the key is interpreted.`
	• CRSP_EXACT: look for a specific value,
	• CRSP_BACK or CRSP_FORWARD : direct selection when partial matches are allowed, or a positional qualifier to base selection on the position relative to the last key accessed.
	• CRSP_NEXT : read next key in sequence
	• CRSP_PREV : read previous key in sequence
	• CRSP_SAME: read same key, possibly with different information
	• CRSP_FIRST : read first key in the database
	• CRSP_LAST : read last key in the database
	`Int match_flag`: User provided variable to load with the level of the read. It will be loaded with a 0 if the load results in reading new master data. It will be loaded with a number greater than 0 if the load impacts detail or global data, but no master data are affected.

RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_EOF: If positional read reaches the end of the file CRSP_NOT_FOUND: If key not found on exact read. If a detail input key is not provided and no items of that entity class are selected, the return is CRSP_SUCCESS as long as the primary key matches. CRSP_FAIL: If error in parameters, handle not opened, error in read operations.
SIDE EFFECTS:	If successful, the wanted data for the key are loaded into the handle set structure which allows item objects to point to the loaded data. The key found for each level is loaded into the outkey item list. If the handle `fiscal_disp_cd` is set to calendar-based and items are fiscal-based, shifted calendars are created and time series are converted to calendar basis. The status argument is loaded based on whether the primary key changed. Handle `primkey` field and `readlvl` are set. `readlvl` is set to the rank of the first entity class changed. If the primary key changed, `getlvl` is set to 0.
PRECONDITIONS:	The item handle must be initialized and opened. The item key must be initialized based on the key type, key element, and the entity class. If not a positional qualifier, the item key `inpkey` list must be loaded.

Example:

```
if ((ret = crsp_itm_read(hndl,CRSP_EXACT, &status)) == CRSP_FAIL)
{
    fprintf (stderr,err_msg);
    got_db_error=1;
    break;
}
```

CRSP_ITM_SET_KEY

`crsp_itm_set_key` loads key information that will be used to load data in a function `crsp_itm_read` call. The key is setup during the function `crsp_itm_open` based on the active keytype. The value passed to this function is entered into the handle attached to the input key item.

PROTOTYPE:	`int crsp_itm_set_key(CRSP_ITM_HNDL *hndl, char *key_itm, void *keyval);`
ARGUMENTS:	`CRSP_ITM_HNDL *hndl:` Access handle containing the needed set structure information and the current item list. `Char *key_itm`: String containing an `itm_name` of an input key item to be loaded. `Void *keyval`: Data to be loaded into the key item. Data must agree with the key item's type.
RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_FAIL: If error in parameters, handle not open, key item.
SIDE EFFECTS:	If successful, the `keyval` is copied into the data location for the input key item element in the handle.
PRECONDITIONS:	The item handle must be initialized and opened. The item key array must be initialized based on a keytype with the function `crsp_itm_open` or function `crsp_itm_init_key` functions. The `key_itm_name` must be a valid item for that keytype, and the `keyval` data must agree with the type of that item.

Example:

```
if (crsp_itm_set_key(hndl,"KYPERMNO",&permno) == CRSP_FAIL)
      {
         fprintf(stderr,err_msg);
         exit(EXIT_FAILURE);
```

```
}
```

CRSP_REF_ELEM_LOOKUP

`crsp_ref_elem_lookup` uses CA reference data that is available from within the item handle, to return a pointer to the desired object within the CRSPAccess database.

PROTOTYPE:	`void *crsp_ref_elem_lookup(CRSP_CA_REF *ref, char *elem_name, int keyset);`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the CA_REF data array bound within the item handle `char *elem_name`: String containing a name of valid group element to be loaded. `int keyset`:Desired item keyset, default is 0
RETURN VALUES:	NULL: Desired object could not be found. Else: Pointer to the data object (`CRSP_TS, CRSP_ARR, CRSP_ROW`).
PRECONDITIONS:	Item handle must be initialized and open. Group element name and keyset must be valid, and the database must have been opened with a wanted value that includes the module that contains the element.

Example:

```
CRSP_ROW *stkhdr_row = 0;
stkhdr_row = crsp_ref_elem_lookup(stkhdr_itm->hndl->ca_ref, "HEADER", 0);
```

CRSP_REF_ELEM_TYPE_LOOKUP

Using the CA reference data available from within the item handle, `crsp_ref_elem_type_lookup` gives the data type of the specified element.

PROTOTYPE:	`int crsp_ref_elem_type_lookup(CRSP_CA_REF *ref, char *elem_name, int *type)`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the `CA_REF` data array bound within the item handle `char *elem_name`: String containing a name of a valid group element `int *type`: Pointer to an integer which will receive the data type
RETURN VALUES:	CRSP_SUCCESS or CRSP_FAIL
SIDE EFFECTS:	The type pointer will be modified regardless of success or failure
PRECONDITIONS:	Item handle must be initialized and open. Element name must be valid.

Example:

```
stkhdr_row = crsp_ref_elem_lookup(stkhdr_itm->hndl->ca_ref, "HEADER", 0);
```

CRSP_REF_GET_ELEM_VAL_CHAR

Using the CA reference data available from within the item handle, `crsp_ref_get_elem_val_char` returns the value of an observation of a structure element that is a string type.

PROTOTYPE:	`char * crsp_ref_get_elem_val_char(CRSP_CA_REF *ref, char *elem_name, char *struct_name, void *data, int index);`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the CA_REF data array bound within the item handle `char *elem_name`: String containing a name of a valid group element `char *struct_name`: String containing the name of a valid data structure `void *data`: Pointer to the valid data container holding the data structure `int index`: The zero-based index of the element to retrieve
RETURN VALUES:	Character pointer to the string value of the data

PRECONDITIONS:	Item handle must be initialized and open. Element name and structure name must be valid. Data pointer and index must be valid.

Example:

```
	/* Get Character values */
	strcpy(siz_cusip, crsp_ref_get_elem_val_char(stkhdr_itm->hndl->ca_ref, "CUSIP",
"HEADER", stkhdr_row, 0));
	crsp_util_strtrim(siz_cusip);
```

CRSP_REF_GET_ELEM_VAL_DOUBLE

Using the CA reference data available from within the item handle, `crsp_ref_get_elem_val_double` returns the value of an observation of a structure element that is a double type.

PROTOTYPE:	`double crsp_ref_get_elem_val_double(CRSP_CA_REF *ref, char *elem_name, char *struct_name, void *data, int index);`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the CA_REF data array bound within the item handle `char *elem_name`: String containing a name of a valid group element `char *struct_name`: String containing the name of a valid data structure `void *data`: Pointer to the valid data container holding the data structure `int index`: The zero-based index of the element to retrieve
RETURN VALUES:	The value of the requested data element.
PRECONDITIONS:	Item handle must be initialized and open. Element name and structure name must be valid. Data pointer and index must be valid.

Example:

```
	/* Get double values */
	Siz_delprc = crsp_ref_get_elem_val_double(stkhdr_itm->hndl->ca_ref, "DLPRC",
"DELIST", stkhdr_row, 0);
```

CRSP_REF_GET_ELEM_VAL_FLOAT

Using the CA reference data available from within the item handle, `crsp_ref_get_elem_val_float` returns the value of an observation of a structure element that is a float type.

PROTOTYPE:	`float crsp_ref_get_elem_val_float(CRSP_CA_REF *ref, char *elem_name, char *struct_name, void *data, int index);`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the CA_REF data array bound within the item handle `char *elem_name`: String containing a name of a valid group element `char *struct_name`: String containing the name of a valid data structure `void *data`: Pointer to the valid data container holding the data structure `int index`: The zero-based index of the element to retrieve
RETURN VALUES:	The value of the requested data element.
PRECONDITIONS:	Item handle must be initialized and open. Element name and structure name must be valid. Data pointer and index must be valid.

Example:

```
/* Get integer values */
siz_permno = crsp_ref_get_elem_val_int(stkhdr_itm->hndl->ca_ref, "PERMNO", "HEADER",
stkhdr_row, 0);
```

CRSP_REF_GET_ELEM_VAL_INT

Using the CA reference data available from within the item handle, `crsp_ref_get_elem_val_int` returns the value of an observation of a structure element that is an integer type.

PROTOTYPE:	`int crsp_ref_get_elem_val_int(CRSP_CA_REF *ref, char *elem_name, char *struct_name, void *data, int index);`
ARGUMENTS:	`CRSP_CA_REF *ref`: Pointer to the CA_REF data array bound within the item handle `char *elem_name`: String containing a name of a valid group element `char *struct_name`: String containing the name of a valid data structure `void *data`: Pointer to the valid data container holding the data structure `int index`: The zero-based index of the element to retrieve
RETURN VALUES:	The value of the requested data element.
PRECONDITIONS:	Item handle must be initialized and open. Element name and structure name must be valid. Data pointer and index must be valid.

Example:

```
/* Get integer values */
siz_permno = crsp_ref_get_elem_val_int(stkhdr_itm->hndl->ca_ref, "PERMNO", "HEADER",
stkhdr_row, 0);
```

REFERENCE INFORMATION

CRSP C API DATA TYPES

All derived types used in the CRSP C API are defined in the module crsp_itm_types. They are included in user programs automatically through the use of crsp_itm_lib module.

Note: This document lists only selected properties of the defined types that are relevant in the user-scope of item-based access. The full individual definitions of the specific C derived types can be referenced from the respective include source files. These files are already included in the crsp_itm_lib module and an explicit include statement is not necessary to use the defined types in your programs. The supplied CRSP C API include files are listed in the following table:

PLATFORM/ LOCATION	FILE	DESCRIPTION
Windows 32-bit **Windows 64-bit** %CRSP_INCLUDE% **SunOS sparc** **Linux 32-bit** **Linux 64-bit** $CRSP_INCLUDE	crsp_itm_ccm_ types.inc	CRSP CCM/Compustat specific data types
	crsp_itm_stk_ types.inc	CRSP Stock specific data types
	crsp_itm_ind_ types.inc	CRSP Index specific data types
	crsp_itm_gen_ types.inc	CRSP generic data types used in all supported data sets
	crsp_itm_types.inc	Data types used in context of item-access
	crsp_itm_params. inc	Declarations of constant parameters used.

To use the CRSP C API library in your program simply include a 'use' statement for the top-level module crsp_itm_lib. All of the required underlying modules will be included automatically. The supplied CRSP C API module files are listed in the following table:

PLATFORM/ LOCATION	FILE	DESCRIPTION
Windows 32-bit Windows 64-bit %CRSP_INCLUDE%\ mod SunOS sparc SunOS i86pc $CRSP_INCLUDE/mod Linux 32-bit Linux 64-bit $CRSP_INCLUDE/mod $CRSP_INCLUDE/ mod_g95	crsp_itm_lib.mod	CRSP C itm-API user-level module
	crsp_itm_utils.mod	Implementations of CRSP itm-API interfaces
	crsp_itm_types. mod	C derived types used in context of CRSP C itm-API
	crsp_itm_xfer.mod	Internal functions and types for CRSP C/C exchange layer

CONTAINER OBJECTS

CRSP container objects are used to uniformly define the storage for various CRSP data types. Generally, the container's data is stored in the associated C array, except in the case of the CRSP_ROW container, where the storage is allocated for an C scalar of the specified data type. The associated storage array is externally allocated with 0-based array bounds.

The CRSP time-series object has an associated calendar of the CRSP_CAL object type which is aligned with the time-series data array, attributing the date to the values stored in the time-series array.

CRSP calendar data is stored in the CRSP_CAL container object, which defines the loaded calendar and also stores the actual calendar data of the defined type. In the context of the CRSP C API, the calendars associated to the time-series items are of day date-type and are accessed with caldt array.

Each container (except CRSP_ROW) has a defined availability range, with missing values set beyond the defined range. The missing value is specific to the data type of the stored data and is located at the pre-defined array index position.

Properties of the CRSP container object types are listed in the following tables:

CRSP_TS

CRSP time-series container object

NAME	C TYPE	DESCRIPTION
objtype	int	Object type id (CRSP_TS_OTID)
arrtype	int	Type id of the data stored in the container
subtype	int	Subtype id of the data stored in the container
maxarr	int	Maximum bound for the storage array (index is 0-based)
beg	int	Lower index of the available stored data
end	int	Upper index of the available stored data
caltype	int	Calendar type of the associated calendar object
cal	CRSP_CAL	Pointer to associated calendar object
miss_val_at = 0		Array index of the missing value for the stored data type

CRSP_ARRAY

CRSP array container object

NAME	C TYPE	DESCRIPTION
objtype	int	Object type id (CRSP_ARRAY_OTID)
arrtype	int	Type id of the data stored in the container
subtype	int	Subtype id of the data stored in the container
maxarr	int	Maximum bound for the storage array (index is 0-based)
num	int	Upper index of the available stored data (index is 0-based)
miss_val_at = maxarr - 1		Array index of the missing value for the stored data type

CRSP_ROW

CRSP row container object

NAME	C TYPE	DESCRIPTION
objtype	int	Object type id (CRSP_ROW_OTID)
arrtype	int	Type id of the data stored in the container
subtype	int	Subtype id of the data stored in the container

CRSP_CAL

CRSP calendar container object

NAME	C TYPE	DESCRIPTION
objtype	int	Object type id (CRSP_CAL_OTID)
calid	int	Id code of the loaded calendar: • CRSP_CALID_DAILY • CRSP_CALID_MONTHLY • CRSP_CALID_ANNUAL • CRSP_CALID_QUARTERLY • CRSP_CALID_SEMIANNUAL • CRSP_CALID_WEEKLY
Loadflag	int	Binary Flag on loaded array parts

NAME	C TYPE	DESCRIPTION
maxarr	int	Maximum bound of the date storage array
gmtoffset	int	Minutes offset from GMT
timezone	int	Code for time zone GMT
relflag	int	If beg and end absolute or relative
beg	int	Valid range subset begin
end	int	Valid range subset end
ndays	int	Number of periods in calendar
name	character(LEN=CRSP_NAMESIZE)	Calendar name
calcd	char (LEN=CRSP_CALCD_LEN)	Calender Code
freqcd	char (LEN=CRSP_CHSR_STRSIZE)	Frequency Code

SUPPORTING TYPES

The CRSP C itm-API supporting types provide additional information about data items and other associated objects in the context of item-based access. An item object is usually associated to a keyset and calendar (in case of time-series items). The details of the keyset (when non-zero) and calendar are given in the `CRSP_ITM_KEYSET` and `CRSP_ITM_CAL` derived types.

Additionally, the details of the current CRSP data set (such as set name, product name, version, etc.) are provided in the `CRSP_ITM_SET` and `CRSP_ROOT_INFO` derived types.

The relevant fields of the supporting types are listed in the following tables:

CRSP_ITM_INFO

Item detail information

NAME	C TYPE	DESCRIPTION
itm_name	char(LEN=CRSP_NAME_LEN)	Item mnemonic name
dflt_keyset	int	Default keyset
full_name	char(LEN=CRSP_NAMESIZE)	Full non-mnenonic name
itm_type	char(LEN=CRSP_TYPE_LEN)	Type of data item
derv_flg	char(LEN=CRSP_TYPE_LEN)	Item is stored/derived
unit_type	char(LEN=CRSP_CODE_LEN)	Type of units (money, ratio)
unit_mult	double precision	Multiplier to get actual value
cat_type	char(LEN=CRSP_CODE_LEN)	Category (BS, IS, CF, MKT)
src_type	char(LEN=CRSP_CODE_LEN)	Source (filing, market)
freq_type	char(LEN=CRSP_TYPE_LEN)	Reporting frequency type
disp_fmt	char(LEN=CRSP_ITEMNAME_LEN)	Display format specifier
disp_len	int	Field width for formatted output
disp_precn	int	Number of decimal places in output
ca_data_size	int	Internal length
ca_arrtype	int	Internal structure it belongs
ca_subtype	int	Internal data sub type
subno_type	int	Type of variant id
epsflag	int	Epsilon type/digits for diffs
cepsflag	int	Epsilon type for character(LEN=diffs)
epsilon	double precision	Absolute epsilon for diffs

NAME	C TYPE	DESCRIPTION
desc	char(LEN=CRSP_DESC_LEN)	Default description for field

CRSP_ITM_KEYSET

Keyset descriptor

NAME	C TYPE	DESCRIPTION
keyset	int	Keyset number
is_loaded	int	True when items where requested with this keyset
keyset_info	CRSP_KEYSET	Information about the keyset
items_arr	CRSP_ARRAY	CRSP array container definition for keyset composing items
items	CRSP_ITM	Array of the items composing the keyset

CRSP_ITM_CAL

Calendar descriptor

NAME	C TYPE	DESCRIPTION
app_id	int	Application ID
cal_seq_num	int	Calendar sequence number
fiscal_disp_cd	char(LEN=CRSP_TYPE_LEN)	Fiscal Display code
freqcd	char(LEN=CRSP_CHAR_STRSIZE)	Frequency code of the calendar
Itm_name	char (LEN=CRSP_NAME_LEN)	Item Name
cal_derv_flg	char (LEN=CRSP_CODE_LEN)	Calendar derived flag
desc	char (LEN=CRSP_DESC_LEN	Description of the Calendar

CRSP_KEYSET

Keyset information

NAME	C TYPE	DESCRIPTION
keyset	int	Keyset number
keyset_tag	char(LEN=CRSP_NAME_LEN)	Keyset tag name
desc	char(LEN=CRSP_DESC_LEN)	Keyset description

CRSP_ITM_SET

Data set descriptor

NAME	C TYPE	DESCRIPTION
set_name	char(LEN=CRSP_NAME_LEN)	Keyset number
path	char(LEN=CRSP_PATHSIZE)	Keyset tag name
root_info	CRSP_ROOT_INFO	Database root information

CRSP_ROOT_INFO

Database root information

NAME	C TYPE	DESCRIPTION
product_ name	char(LEN=CRSP_PROD_ NAMESIZE)	Database name
product_code	char(LEN=CRSP_CODE_NAMESIZE)	Database code

NAME	C TYPE	DESCRIPTION
version	int	Version number of db
crt_date	char(LEN=CRSP_DATE_SIZE)	Dates are Dow Mon DD HH:MM:SS YYYY
mod_date	char(LEN=CRSP_DATE_SIZE)	Last modification date of db
cut_date	char(LEN=CRSP_DATE_SIZE)	Cut date of db
binary_type	char(LEN=CRSP_CHAR_STRSIZE)	L (IEEE little-endian) or B (big)
code_version	char(LEN=CRSP_OS_NAMESIZE)	CA version

CHAPTER 3: ITEM-BASED ACCESS IN FORTRAN

CRSP FORTRAN 95 API DATA OBJECTS

Access to CRSP databases is achieved through two principal objects: the access handle – of type CRSP_ITM_HNDL_T, and the item – of type CRSP_ITM_T. These two important object types can be seen in steps 2 and 3 of the sample program flow on page 69.

CRSP_ITM_HNDL_T

The item-access handle object `type(CRSP_ITM_HNDL_T)` encapsulates the information required to establish and maintain a single item-access session to a given CRSP database. Additional access sessions (either to the same or to another CRSP database), concurrent in the same program, require a separate access handle object. All of the item objects available in the active session are grouped within the respective access handle.

The main properties of the access handle object are listed on the following table:

NAME	FORTRAN 95 TYPE	DESCRIPTION
keytype	character(LEN=CRSP_NAME_LEN)	Determines the keys used to select data in read functions. Supported keytypes for the application are included in the reference data. A default will be set.
keyset_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether keyset items are labeled by the keyset number (NUM), the keyset tag (TAG), or the expanded list of all items comprising the keyset (EXP). The default display is TAG.
fiscal_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether fiscal-based time series items are reported on a calendar basis (C) or a fiscal basis (F). The default it C.
curr_disp_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether monetary values are reported in the currency reported by Compustat (REP) or in US Dollars (USD). The default currency display code is REP.
grp_fill_cd	character(LEN=CRSP_TYPE_LEN)	Determines whether group item lists are filled so that every selected item is included for every selected keyset (Y or N). The default is Yes (Y).
dataset	CRSP_ITM_SET_T	Pointer to descriptor of currently attached CRSP dataset; includes root info for the CRSP dataset.

In a user-program the access handle objects are normally declared and allocated directly then passed to Fortran 95 itm-API functions as a parameter. The function `crsp_f_itm_init` initializes the contents of the access handle and connects it to the specified CRSP database.

CRSP_ITM_T

The item object `type(CRSP_ITM_T)` represents a generic container for a single data item defined in a given CRSP database. It unifies the data types defined for each of the supported CRSP databases and allows uniform access to the associated CRSP data containers from your programs.

The main properties of the item object are listed in the following table:

NAME	FORTRAN 95 TYPE	DESCRIPTION
itm_name	character(LEN=CRSP_NAME_LEN)	name of the item from a CRSP dataset.
keyset	integer	number of the keyset defined in a CRSP dataset.
itm_info	CRSP_ITM_INFO_T	item metadata; includes description, default keyset, and stored data type.
obj	CRSP_ITM_OBJ_T	describes the underlying CRSP data-object.
arr	CRSP_ITM_OBJARR_T	describes the Fortran 95 container associated with the defined CRSP data-object.
itmkeyset	CRSP_ITM_KEYSET_T	describes the details of the keyset (when non-zero and loaded), including its number, name, tag, and array of composing items of same CRSP_ITM_T type.

NAME	FORTRAN 95 TYPE	DESCRIPTION
itmcal	CRSP_ITM_CAL_T	for calendar-bound items, describes the details of the attached calendar, including its id, keyset ,frequency, and attached calendar object of CRSP_CAL_T type. When requested, the calendar may be 'shifted', based on the currently loaded company's FYE to attribute properly the item's period data.

Item objects are normally declared as Fortran 95 pointers and then attached to the actual defined item objects by calling the `crsp_f_itm_find` function for the given access handle and the specified item name and keyset.

CRSP_ITM_OBJ_T

Item data is accessed from the data-object `itm%obj` and associated to a Fortran 95 container `itm%arr`. The item data container object `type(CRSP_ITM_OBJ_T)` describes an instance of a CRSP data-object (time-series, array, row) that is defined for the specific item. Only a single data-object can be defined for a given item, which is identified by the `objtype` property.

Properties of the item data-object are listed in the following table:

NAME	FORTRAN 95 TYPE	DESCRIPTION
objtype	integer	type of the defined and allocated object: • CRSP_TS_OTID: CRSP time-series • CRSP_ARRAY_OTID: CRSP array • CRSP_ROW_OTID: CRSP row
ts	CRSP_TS_T	pointer to allocated CRSP time-series data-object.
arr	CRSP_ARRAY_T	pointer to allocated CRSP array data-object.
row	CRSP_ROW_T	pointer to allocated CRSP row data-object.
is_empty	logical	indicates whether the allocated CRSP data-object contains no data.

The item data-object normally has an associated Fortran 95 container, which is either Fortran 95 array or scalar of the data type corresponding to the actual stored data, as identified by `arrtype` property. Details of the CRSP container objects types are listed in the reference section CRSP Container Objects.

CRSP_ITM_OBJARR_T

The item data array, type `CRSP_ITM_OBJARR_T` describes the associated Fortran 95 container object. The Fortran 95 container is allocated based on the object's type (`objtype`) and contained data type (`arrtype`). The respective scalar member has suffix _val to its name, and _arr for the array type. Time-series and array data are stored in array type, while row data is kept in scalar type:

`itm%arr%arrtype`:

`CRSP_TS_OTID: itm%arr%<arrtype_name>_arr` - time-series object data array

`CRSP_ARARY_OTID: itm%arr%<arrtype_name>_arr` - array object data array

`CRSP_ROW_OTID: itm%arr%<arrtype_name>_val` - row object data scalar.

NOTE: Throughout the implementation of the CRSP Fortran 95 API, the Fortran 95 array indexing is 0-based, thus the first element of an array is `data_arr(0)`.

Properties of the item data array for the item data types that are common to all of the supported CRSP datasets are listed in the following table:

NAME	FORTRAN 95 TYPE	DESCRIPTION
arrtype	integer	data type of the defined and allocated Fortran 95 container. Common data types: • CRSP_INTEGER_TID: integer • CRSP_FLOAT_TID: real • CRSP_DOUBLE_TID: double precision • CRSP_CHAR_TID: character(1) • CRSP_CHARACTER_TID: CRSP_VARSTRING_T type
int_val / arr	integer / dimension (:)	pointer to allocated scalar / array of integer type
flt_val / arr	real / dimension (:)	pointer to allocated scalar / array of real type
dbl_val / arr	double precision / dimension (:)	pointer to allocated scalar / array of double precision type
char_val / arr	character(LEN=1) / dimension (:)	pointer to allocated scalar / array of single-character type
vstr_val / arr	CRSP_VARSTRING_T / dimension (:)	pointer to allocated scalar / array of variable-length string type
structured types specific to CRSP datasets	Refer to the description of data types for the specific CRSP dataset.	

In a user-program the item container data is usually accessed directly as defined by item's data type, eg:

```
print *, sale_itm%arr%dbl_arr(i)
```

The item data container is normally accessed in association with its item data-object.

NOTE: If an incorrect data container is referenced, an access violation error should occur on the referenced null-pointer. In such situations the recommended action is to verify that the appropriate containers are being accessed for the selected items.

Data for items of CRSP array type is accessed in the valid `[0..num-1]` index range, as defined in the corresponding `arr` data-object. For example:

```
itm%arr%dbl_arr(i), i=0..itm%obj%arr%num-1
```

Data for items of CRSP time-series type is accessed in the valid `[beg,end]` index range, as defined in the corresponding ts data-object, e.g.:

```
itm%arr%dbl_arr(i), i=itm%obj%ts%beg..itm%obj%ts%end
```

Data for items of CRSP row type is not indexed and is accessed directly from the value as defined by the corresponding scalar/structured type, e.g.:

```
itm%arr%master_val%ccmid
```

To verify if an element of an array item contains a missing value, call the function `crsp_f_itm_is_miss_arrval`.

SUPPORTING INFORMATION

Various supporting information about CRSP databases, items, keysets and other item-access objects is stored in the following derived types:

FORTRAN 95 TYPE NAME	DESCRIPTION	ACCESS VIA TYPE NAME	USAGE
CRSP_ITM_INFO_T	Item information; includes item's full name, description, display format, data type and size information. Also includes the default keyset number associated with this item.	CRSP_ITM_T	itm%itm_info

FORTRAN 95 TYPE NAME	DESCRIPTION	ACCESS VIA TYPE NAME	USAGE
CRSP_ITM_KEYSET_T	Keyset descriptor; includes keyset information and the array of items composing the keyset.	CRSP_ITM_T	itm%itmkeyset
CRSP_ITM_CAL_T	Calendar descriptor; includes calendar's id, associated keyset number, base calendar name, and calendar's frequency, also the base calendar object. Additionally, for fiscal calendars indicates whether the calendar has been shifted based on the currently loaded company's FYE.	CRSP_ITM_T	itm%itmcal
CRSP_KEYSET_T	Keyset information; includes keyset's number, name, tag, and description. Indicates whether the keyset has been loaded and associated with any of the requested items.	CRSP_ITM_ KEYSET_T	itmkeyset%keyset_info
CRSP_ITM_SET_T	CRSP data set descriptor; includes the set's path, name, id, and database root information.	CRSP_ITM_HNDL_T	hndl%dataset
CRSP_ROOT_INFO_T	CRSP data set root information; includes internal service information about the currently loaded database such as creation/modification date, product code and name, and descriptors of available calendars. Mainly intended for internal use.	CRSP_ITM_SET_T	dataset%root_info

NOTE: While selected supported information is populated on initiating of the connection to a CRSP data set (on return from call to `crsp_f_itm_init`), the listed supported information becomes available only on opening of the CRSP data set (on return from call to `crsp_f_itm_open`).

The relevant details of the derived types shown above are listed in the *Supporting Types* on page 69.

GENERIC DATA TYPES

All CRSP databases contain data items of both simple Fortran 95 data types and of database-specific structured data types. Moreover, each composing field of the structured data type can instead be requested as an individual data item of the simple Fortran 95 data type.

The vast majority of the data items defined in CRSP datasets are of CRSP time-series container object type, with the stored values commonly of generic Fortran 95 data types. A limited set of items is stored in CRSP array and CRSP row container objects; these items are mostly of structured data types and are listed in the following sections regarding particular CRSP database products.

The following table lists the supported generic data types and ways to access data from the item-associated container:

ITEM OBJECT TYPE	FORTRAN 95 TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ITM_T
time-series	CRSP_TS_T		itm%obj%ts
	integer	int(4)	Itm%arr%int_arr(i)
	real	float(4)	Itm%arr%flt_arr(i)
	double precision	double(8)	Itm%arr%dbl_arr(i)
	character(LEN=1)	char (1)	Itm%arr%char_arr(i)
	CRSP_VARSTRING_T	char(n)	Itm%arr%vstr_arr(i)
array	CRSP_ARRAY_T		itm%obj%arr
	integer	int(4)	Itm%arr%int_arr(i)
	real	float(4)	Itm%arr%flt_arr(i)
	double precision	double(8)	Itm%arr%dbl_arr(i)
	character(LEN=1)	char (1)	Itm%arr%char_arr(i)
	CRSP_VARSTRING_T	char(n)	Itm%arr%vstr_arr(i)

ITEM OBJECT TYPE	FORTRAN 95 TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ITM_T
row	CRSP_ROW_T		itm%obj%row
	integer	int(4)	Itm%arr%int_val
	real	float(4)	Itm%arr%flt_val
	double precision	double(8)	Itm%arr%dbl_val
	character(LEN=1)	char (1)	Itm%arr%char_val
	CRSP_VARSTRING_T	char(n)	Itm%arr%vstr_val

NOTE: The derived type `CRSP_VARSTRING_T` accommodates varying-length character strings and is used in the context of the CRSP Fortran 95 API to store data of individual character items that are composing fields of a structured data item. For example, the CCM structured item COMPANY has a field for company name, which can be referenced indirectly as `company_itm%arr%company_val%conm` as a fixed-length character string. Alternatively, this field can be requested individually as the CONM item and then referenced as `conm_itm%arr%vstr_val` as varying-length string.

See the description of the *CRSP_VARSTRING_T type* on page 71 for usage information.

ACCESSING CRSP DATABASES

The following sections describe the details of accessing CRSP databases supported by the API. Supported databases are the CRSP US Stock Database, the CRSP US Index Database, and the CRSP/Compustat Merged Database. Each section presents database connection information, available access keys, as well how to access a database's data groups and items from your programs.

CRSP US STOCK DATABASE

To connect to the specific CRSP Stock database instance the path to its database root should be specified. When installed on your system, CRSP Stock data set will be assigned an environment variable pointing to the CRSP Stock database root.

Additionally, an application ID should be specified on the call to `crsp_f_itm_init` to indicate the item-universe to be loaded for the session and describes the available items and item groups, eg:

```
sts = crsp_f_itm_init (hndl,'CRSP_DSTK',app_id,'stk1')
```

User-programs should access the CRSP Stock data set with the `app_id` as listed in the following table:

STK ROOT/APP ID	FORTRAN 95 TYPE	DESCRIPTION
CRSP_DSTK		CRSP Daily Stock data set
CRSP_DSTKITM_ID	integer	CRSP Daily Stock data items and groups
CRSP_MSTK		CRSP Monthly Stock data set
CRSP_MSTKITM_ID	integer	CRSP Monthly Stock data items and groups

The details on included items and item groups can be found starting on page 37.

ACCESS KEYS

CRSP Stock data set contains various data on companies and securities. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to STK data.

Default access key is loaded automatically on opening the access session to the CRSP STK data set

Additionally, a set of alternative access keys (and associated key items) is defined to facilitate access to the data by CRSP PERMNO, CUSIP, and other keys.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_f_itm_get_key` can also be used to retrieve the value of the access key items for the currently read record.

To switch to access by an alternative key, a user calls `crsp_f_itm_load_key` to set the access key index, followed by calls to `crsp_itm_set_key` to set the value of the key items used on subsequent reading of the database.

The defined STK access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	FORTRAN 95 TYPE	DESCRIPTION	NOTES
PERMNO		CRSP historical PERMNO	default
KYPERMNO	Integer	CRSP company issue's PERMNO	primary key item
PERMCO		CRSP historical PERMCO	
KYPERMCO	Integer	CRSP company's PERMCO	primary key item
CUSIP		CRSP Stock CUSIP	
KYCUSIP	char(CRSP_CUSIP_LEN)	CRSP Stock issue's CUSIP	primary key item

CCM ACCESS KEY/KEY ITEMS	FORTRAN 95 TYPE	DESCRIPTION	NOTES
HCUSIP		CRSP Stock Historical CUSIP	
KYHCUSIP	char(CRSP_CUSIP_LEN)	CRSP issue's Historical CUSIP	primary key item
Ticker		CRSP Stock ticker	
KYTICKER	char(CRSP_STK_TIC_LEN)	CRSP issue's ticker	primary key item
SICCD		CRSP Stock SIC code	
KYSIC	Integer	CRSP Stock security's SIC	primary key item

DATA TYPES

Generally, individual CRSP Stock database data items are of common simple Fortran 95 data types and stored data can be accessed through `itm%arr` and corresponding scalar or array member.

Additionally, selected CRSP supplemental STK data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by `itm_name`, but recommended programming access is through the `itm_name` of the structure. To access the structured type and its fields, load the structured type `itm_name` during initialization, create a `CRSP_ITM_T` pointer matching the `itm_name`, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_f_itm_load(hndl,'HEADER',match_flag)

sts = crsp_f_itm_find(hndl,'HEADER',0,header_itm)

permno = header_itm%arr%header_val%permno

...
```

STRUCTURED TYPES FOR CRSP US STOCK DATABASE ACCESS

The tables below show the data groups available as STK structured types and their usage through the CRSP_ITM_T type.

STK MNEMONIC		DESCRIPTION	FORTRAN 95 TYPE NAME	OBJECT TYPE	OBJECT ACCESS VIA CRSP_ITM_T
DAILY	MONTHLY				
HEADER	MHEADER	Issue header information	CRSP_STK_HEADER_T	row	header_itm%obj%row
NAMES	MNAMES	Name information snapshot	CRSP_STK_NAME_T	array	names_itm%obj%arr
DISTS	MDISTS	Information for a distribution event	CRSP_STK_DIST_T	array	dists_itm%obj%arr
SHARES	MSHARES	Shares outstanding snapshot	CRSP_STK_SHARE_T	array	shares_itm%obj%arr
DELIST	MDELIST	Delisting information	CRSP_STK_DELIST_T	array	delist_itm%obj%arr
NASDIN	MNASDIN	Snapshot of Nasdaq information	CRSP_STK_NASDIN_T	array	nasdin_itm%obj%arr
PORTF	MPORTF	Portfolio statistic and assignment snapshot	CRSP_STK_PORT_T	array	portf_itm%obj%arr
GROUP	MGROUP	Group statistic and assignment snapshot	CRSP_STK_GROUP_T	array	group_itm%obj%arr

(M)HEADER

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
HEADER	MHEADER	Issue header information	CRSP_STK_HEADER_T		header_itm%arr%header_val
PERMNO	MPERMNO	PERMNO	int(4)	I6	header_val%permno
PERMCO	MPERMCO	PERMCO	int(4)	I6	header_val%permco

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
COMPNO	MCOMPNO	NASDAQ Company Number	int(4)	I8	header_val%compno
ISSUNO	MISSUNO	NASDAQ Issue Number	int(4)	I8	header_val%issuno
HEXCD	MHEXCD	Exchange Code - Header	int(4)	I2	header_val%hexcd
HSHRCD	MHSHRCD	Share Code - Header	int(4)	I3	header_val%hshrcd
HNAMECD	MHNAMECD	Name Code -Header	int(4)	I4	header_val%hnamecd
HSICCD	MHSICCD	Standard Industrial Classification (SIC) Code - Header	int(4)	I2	header_val%hsiccd
BEGDT	MBEGDT	Begin of Stock Data	int(4)	I8	header_val%begdt
ENDDT	MENDDT	End of Stock Data	int(4)	I8	header_val%enddt
HDLSTCD	MHDLSTCD	Delisting Code -Header	int(4)	I3	header_val%dlstcd
CUSIP	MCUSIP	CUSIP - Header	char(16)	A8	header_val%hcusip
HTICK	MHTICK	Ticker Symbol - Header	char(16)	A6	header_val%htick
HNAICS	MHNAICS	North American Industry Classification System (NAICS) - Header	char(8)	A7	header_val%hnaics
HCOMNAM	MHCOMNAM	Company Name - Header	char(36)	A36	header_val%hcomnam
HTSYMBOL	MHTSYMBOL	Trading Ticker Symbol - Header	char(12)	A12	header_val%htsymbol
HCNTRYCD	MHCNTRYCD	Country Code - Header	char(4)	A3	header_val%hcntrycd
HPRIMEXCH	MHPRIMEXCH	Primary Exchange - Header	char(1)	A1	header_val%hprimexch
HSUBEXCH	MHSUBEXCH	Sub-Exchange - Header	char(1)	A1	header_val%hsubexch
HTRDSTAT	MHTRDSTAT	Trading Status - Header	char(1)	A1	header_val%htrdstat
HSECSTAT	MHSECSTAT	Security Status - Header	char(1)	A1	header_val%hsecstat
HSHRTYPE	MHSHRTYPE	Share Type - Header	char(1)	A1	header_val%hshrtype
HISSUERCD	MHISSUERCD	Issuer Code -Header	char(1)	A1	header_val%hissuercd
HINCCD	MHINCCD	Incorporation Code -Header	char(1)	A1	header_val%hinccd
HITS	MHITS	Intermarket Trading System Indicator - Header	char(1)	A1	header_val%hits
HDENOM	MHDENOM	Trading denomination	char(1)	A1	header_arr(i)%hdenom
HELIGCD	MHELIGCD	Eligibility code	char(1)	A1	header_arr(i)%heligcd
HCONVCD	MHCONVCD	Convertible code	char(1)	A1	header_arr(i)%hconvcd
HNAMEFLAG	MHNAMEFLAG	Name flag	char(1)	A1	header_arr(i)%hnameflag
HNAMEDESC	MHNAMEDESC	Name description	char(24)	A15	header_arr(i)%hnamedesc
HRATING	MHRATING	Rating (if applicable) or strike price	float(4)	F9.4	header_arr(i)%hrating
HEXPDT	MHEXPDT	Expiration date	int(4)	I8	header_arr(i)%hexpdt

(M)NAMES

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
NAMES	MNAMES	Name information snapshot	CRSP_STK_NAME_T		names_itm%arr%names_arr
UOT	MUOT	Unit of Trade, End of Period - actual	int(4)	I6	names_arr(i)%uot
NAMEDT	MNAMEDT	Names Information Begin Date	int(4)	I8	names_arr(i)%namedt
NAMEENDDT	MNAMEENDDT	Names Information End Date	int(4)	I8	names_arr(i)%nameenddt
SHRCD	MSHRCD	Share Code	int(4)	I2	names_arr(i)%shrcd

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
NAMECD	MNAMECD	Name Code, End of Period - actual	int(4)	I3	names_arr(i)%namecd
EXCHCD	MEXCHCD	Exchange Code	int(4)	I2	names_arr(i)%exchcd
SICCD	MSICCD	Standard Industrial Classification (SIC) Code	int(4)	I4	names_arr(i)%siccd
NCUSIP	MNCUSIP	CUSIP	char(16)	A8	names_arr(i)%ncusip
TICKER	MTICKER	Ticker Symbol	char(8)	A5	names_arr(i)%ticker
SNAICS	MSNAICS	North American Industry Classification System (NAICS)	char(8)	A7	names_arr(i)%snaics
COMNAM	MCOMNAM	Company Name	char(36)	A32	names_arr(i)%comnam
TSYMBOL	MTSYMBOL	Trading Ticker Symbol	char(12)	A10	names_arr(i)%tsymbol
CNTRYCD	MCNTRYCD	Country Code, End of Period - actual	char(4)	A3	names_arr(i)%cntrycd
PRIMEXCH	MPRIMEXCH	Primary Exchange	char(1)	A1	names_arr(i)%primexch
SUBEXCH	MSUBEXCH	Sub-Exchange	char(1)	A1	names_arr(i)%subexch
TRDSTAT	MTRDSTAT	Trading Status	char(1)	A1	names_arr(i)%trdstat
SECSTAT	MSECSTAT	Security Status	char(1)	A1	names_arr(i)%secstat
SHRTYPE	MSHRTYPE	Share Type, End of Period - actual	char(1)	A1	names_arr(i)%shrtype
ISSUERCD	MISSUERCD	Issuer Code, End of Period - actual	char(1)	A1	names_arr(i)%issuercd
INCCD	MINCCD	Incorporation Code, End of Period - actual	char(1)	A1	names_arr(i)%inccd
ITS	MITS	Intermarket Trading System, End of Period - actual	char(1)	A1	names_arr(i)%its
DENOM	MDENOM	Trading Denomination, End of Period - actual	char(1)	A1	names_arr(i)%denom
ELIGCD	MELIGCD	Eligibility Code, End of Period - actual	char(1)	A1	names_arr(i)%eligcd
CONVCD	MCONVCD	Convertible Code, End of Period - actual	char(1)	A1	names_arr(i)%convcd
NAMEFLAG	MNAMEFLAG	Name Flag, End of Period - actual	char(1)	A1	names_arr(i)%nameflag
SHRCLS	MSHRCLS	Share Class	char(4)	A1	names_arr(i)%shrcls
NAMEDESC	MNAMEDESC	Name Description, End of Period - actual	char(24)	A15	names_arr(i)%namedesc
RATING	MRATING	Interest Rate, End of Period - actual	float(4)	F9.4	names_arr(i)%rating
EXPDT	MEXPDT	Expiration Date, End of Period - actual	int(4)	I8	names_arr(i)%expdt

(M)DISTS

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
DISTS	MDISTS	Information for a distribution event	CRSP_STK_DIST_T		dists_itm%arr%dists_arr
DISTCD	MDISTCD	Distribution Code	int(4)	I4	dists_arr(i)%distcd
DIVAMT	MDIVAMT	Dividend Amount	float(4)	F9.5	dists_arr(i)%divamt
FACPR	MFACPR	Factor to Adjust Price in Period	float(4)	F8.4	dists_arr(i)%facpr
FACSHR	MFACSHR	Factor to Adjust Shares Outstanding	float(4)	F8.4	dists_arr(i)%facshr
DCLRDT	MDCLRDT	Distribution Declaration Date	int(4)	I8	dists_arr(i)%dclrdt
EXDT	MEXDT	Ex-Distribution Date	int(4)	I8	dists_arr(i)%exdt
RCRDDT	MRCRDDT	Record Date	int(4)	I8	dists_arr(i)%rcrddt
PAYDT	MPAYDT	Payment Date	int(4)	I8	dists_arr(i)%paydt
ACPERM	MACPERM	Acquiring PERMNO	int(4)	I5	dists_arr(i)%acperm
ACCOMP	MACCOMP	Acquiring PERMCO	int(4)	I5	dists_arr(i)%accomp

(M)SHARES

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
SHARES	MSHARES	Shares outstanding snapshot	CRSP_STK_SHARE_T		shares_itm%arr%shares_arr
SHROUT	MSHROUT	Shares Outstanding	int(4)	I10	shares_arr(i)%shrout
SHRSDT	MSHRSDT	Shares Outstanding Observation Date	int(4)	I8	shares_arr(i)%shrsdt
SHRSENDDT	MSHRSENDDT	Shares Outstanding Observation End Date	int(4)	I8	shares_arr(i)%shrsenddt
SHRFLG	MSHRFLG	Shares Outstanding Observation Flag	int(4)	I4	shares_arr(i)%shrflg

(M)DELIST

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
DELIST	MDELIST	Delisting information	CRSP_STK_DELIST_T		delist_itm%arr%delist_arr
DLSTDT	MDLSTDT	Delisting Date	int(4)	I8	delist_arr(i)%dlstdt
DLSTCD	MDLSTCD	Delisting Code	int(4)	I3	delist_arr(i)%dlstcd
NWPERM	MNWPERM	Linked PERMNO After Delisting	int(4)	I8	delist_arr(i)%nwperm
NWCOMP	MNWCOMP	Linked PERMCO After Delisting	int(4)	I8	delist_arr(i)%nwcomp
NEXTDT	MNEXTDT	Date of Next Available Information	int(4)	I8	delist_arr(i)%nextdt
DLAMT	MDLAMT	Total Amount Used in Delisting Return	float(4)	F13.5	delist_arr(i)%dlamt
DLRETX	MDLRETX	Delisting Return without Dividends	float(4)	F11.6	delist_arr(i)%dlretx
DLPRC	MDLPRC	Delisting Price	float(4)	F13.5	delist_arr(i)%dlprc
DLPDT	MDLPDT	Delisting Payment Date	int(4)	I8	delist_arr(i)%dlpdt
DLRET	MDLRET	Delisting Return	float(4)	F11.6	delist_arr(i)%dlret

(M)NASDIN

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
NASDIN	MNASDIN	Snapshot of Nasdaq information	CRSP_STK_NASDIN_T		nasdin_itm%arr%nasdin_arr
TRTSCD	MTRTSCD	NASDAQ Status Code, End of Period	int(4)	I2	nasdin_arr(i)%trtscd
TRTSDT	MTRTSDT	Beginning Effective Date of Traits	int(4)	I8	nasdin_arr(i)%trtsdt
TRTSENDDT	MTRTSENDDT	Last Effective Date of Traits	int(4)	I8	nasdin_arr(i)%trtsenddt
NMSIND	MNMSIND	NASDAQ National Market Indicator	int(4)	I2	nasdin_arr(i)%nmsind
MMCNT	MMMCNT	NASDAQ Market Makers Count	int(4)	I4	nasdin_arr(i)%mmcnt
NSDINX	MNSDINX	NASDAQ Index Code	int(4)	I2	nasdin_arr(i)%nsdinx

(M)PORTF

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
PORTF	MPORTF	Portfolio statistic and assignment snapshot	CRSP_STK_PORT_T		port_itm%arr%port_arr
PORT	MPORT	Portfolio Assignment	int(4)	I4	portf_arr(i)%port
STAT	MSTAT	Portfolio Statistic Value	double(8)	F16.5	portf_arr(i)%stat

(M)GROUP

STK MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
GROUP	MGROUP	Group statistic and assignment snapshot	CRSP_STK_GROUP_T		group_itm%arr%group_arr
GRPDT	MGRPDT	Group Beginning Date	int(4)	I8	group_arr(i)%grpdt
GRPENDDT	MGRPENDDT	Group Ending Date	int(4)	I8	group_arr(i)%grpenddt
GRPFLAG	MGRPFLAG	Group Flag	int(4)	I4	group_arr(i)%grpflag
GRPSUBFLAG	MGRPSUBFLAG	Group Subflag	int(4)	I4	group_arr(i)%grpsubflag

CRSP US INDEX DATABASE

To connect to the specific CRSP Index database instance the path to its database root should be specified. When installed on your system, CRSP Index data sets will be assigned an environment variable pointing to the CRSP Index database root.

Additionally, an application ID should be specified on the call to `crsp_f_itm_init` to indicate the item-universe to be loaded for the session and describes the available items and item groups, eg:

```
sts = crsp_f_itm_init (hndl,'CRSP_DSTK',app_id,'ind1')
```

User-programs should access the CRSP Index data sets with the `app_id` as listed in the following table:

STK ROOT/APP ID	FORTRAN 95 TYPE	DESCRIPTION
CRSP_DSTK		CRSP Daily Stock and Index data sets
CRSP_DINDITEMS_ID	integer	CRSP Daily Index series data items and groups
CRSP_DINDGITEMS_ID	integer	CRSP Daily Index group data items and groups
CRSP_MSTK		CRSP Monthly Stock and Index data sets
CRSP_MINDITEMS_ID	integer	CRSP Monthly Index series data items and groups
CRSP_MINDGITEMS_ID	integer	CRSP Monthly Index group data items and groups

ACCESS KEYS

CRSP Index data sets includes various data on market indexes. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to IND data.

Default access key is loaded automatically on opening the access session to the CRSP IND data set.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_f_itm_get_key` can also be used to retrieve the value of the access key items for the currently read record.

The defined IND access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	FORTRAN 95 TYPE	DESCRIPTION	NOTES
indno		CRSP Index's INDNO	default
KYINDNO	integer	CRSP index's INDNO	primary key item

DATA TYPES

Generally, individual CRSP Index database data items are of common simple Fortran 95 data types and stored data can be accessed through `itm%arr` and corresponding scalar or array member.

Additionally, selected CRSP supplemental IND data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by itm_name, but recommended programming access is through the itm_name of the structure. To access the structured type and its fields, load the structured type itm_name during initialization, create a CRSP_ITM_T pointer matching the itm_name, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_f_itm_load(hndl,'INDHDR',match_flag)

sts = crsp_f_itm_find(hndl,'INDHDR',0,indhdr_itm)

indno = indhdr_itm%arr%indhdr_val%indno

...
```

STRUCTURED TYPES FOR CRSP US INDEX DATABASE ACCESS

The tables below show the data groups available as structured types and their usage through the `CRSP_ITM_T` type.

IND MNEMONIC		DESCRIPTION	FORTRAN 95 TYPE NAME	OBJECT TYPE	OBJECT ACCESS VIA CRSP_ITM_T
DAILY	MONTHLY				
INDHDR	MINDHDR	Index header information	CRSP_IND_HEADER_T	row	indhdr_itm%obj%row
REBAL	MREBAL	Index rebalancing period summary	CRSP_IND_REBAL_T	array	rebal_itm%obj%arr
LIST	MLIST	Issue list information	CRSP_IND_LIST_T	array	list_itm%obj%arr

(M)INDHDR

IND MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
INDHDR	MINDHDR	Index header information	CRSP_IND_HEADER_T		indhdr_itm%arr%indhdr_val
INDNO	MINDNO	specific index series	int(4)	I7	indhdr_val%indno
INDCO	MINDCO	major index series group	int(4)	I7	indhdr_val%indco
PRIMFLAG	MPRIMFLAG	0 if master or permno of master	int(4)	I6	indhdr_val%primflag
PORTNUM	MPORTNUM	portfolio number in master if subset	int(4)	I6	indhdr_val%portnum
INDNAME	MINDNAME	index name	char(80)	A79	indhdr_val%indname
GROUPNAME	MGROUPNAME	index group name	char(80)	A79	indhdr_val%groupname
METHCODE	MMETHCODE	code of possible methodology types	int(4)	I6	indhdr_val%method%methcode
PRIMTYPE	MPRIMTYPE	fractile, selected, rulebased, market	int(4)	I6	indhdr_val%method%primtype
SUBTYPE	MSUBTYPE	index type subcode	int(4)	I6	indhdr_val%method%subtype
WGTTYPE	MWGTTYPE	reweighting type flag	int(4)	I6	indhdr_val%method%wgttype
WGTFLAG	MWGTFLAG	reweighting timing flag	int(4)	I6	indhdr_val%method%wgtflag
FLAGCODE	MFLAGCODE	code of possible exception types	int(4)	I6	indhdr_val%flags%flagcode
ADDFLAG	MADDFLAG	handling new issues	int(4)	I6	indhdr_val%flags %addflag
DELFLAG	MDELFLAG	handling issues becoming ineligible	int(4)	I6	indhdr_val%flags%delflag
DELRETFLAG	MDELRETFLAG	measuring return of delisted issues	int(4)	I6	indhdr_val%flags%delretflag
MISSFLAG	MMISSFLAG	handling missing prices	int(4)	I6	indhdr_val%flags%missflag
UUNIVCODE	MUUNIVCODE	code of possible universe/subset types (index universe)	int(4)	I6	indhdr_val%induniv%univcode
UBEGDT	MUBEGDT	beginning date of valid data (index universe)	int(4)	I8	indhdr_val%induniv%begdt
UENDDT	MUENDDT	beginning date of valid data (index universe)	int(4)	I8	indhdr_val%induniv%enddt

IND MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
UWANTEXCH	MUWANTEXCH	valid exchange code (index universe)	int(4)	I6	indhdr_val%induniv%wantexch
UWANTNMS	MUWANTNMS	Nasdaq National Market classification (index universe)	int(4)	I6	indhdr_val%induniv%wantnms
UWANTWI	MUWANTWI	when-issued code (index universe)	int(4)	I6	indhdr_val%induniv%wantwi
UWANTINC	MUWANTINC	valid incorporation (index universe)	int(4)	I6	indhdr_val%induniv%wantinc
USCCODE	MUSCCODE	code of possible restriction types (index universe)	int(4)	I6	indhdr_val%induniv%sccode
UFSTDIG	MUFSTDIG	bitmap of first share code digit (index universe)	int(4)	I6	indhdr_val%induniv%fstdig
USECDIG	MUSECDIG	bitmap of second share code digit (index universe)	int(4)	I6	indhdr_val%induniv%secdig
PUNIVCODE	MPUNIVCODE	code of possible universe/subset types (partition universe)	int(4)	I6	indhdr_val%partuniv%univcode
PBEGDT	MPBEGDT	beginning date of valid data (partition universe)	int(4)	I8	indhdr_val%partuniv%begdt
PENDDT	MPENDDT	beginning date of valid data (partition universe)	int(4)	I8	indhdr_val%partuniv%enddt
PWANTEXCH	MPWANTEXCH	valid exchange code (partition universe)	int(4)	I6	indhdr_val%partuniv%wantexch
PWANTNMS	MPWANTNMS	Nasdaq National Market classification (partition universe)	int(4)	I6	indhdr_val%partuniv%wantnms
PWANTWI	MPWANTWI	when-issued code (partition universe)	int(4)	I6	indhdr_val%partuniv%wantwi
PWANTINC	MPWANTINC	valid incorporation (partition universe)	int(4)	I6	indhdr_val%partuniv%wantinc
PSCCODE	MPSCCODE	code of possible restriction types (partition universe)	int(4)	I6	indhdr_val%partuniv%sccode
PFSTDIG	MPFSTDIG	bitmap of first share code digit (partition universe)	int(4)	I6	indhdr_val%partuniv%fstdig
PSECDIG	MPSECDIG	bitmap of second share code digit (partition universe)	int(4)	I6	indhdr_val%partuniv%secdig
RULECODE	MRULECODE	code of possible assignment rule types	int(4)	I6	indhdr_val%rules%rulecode
BUYFNCT	MBUYFNCT	function code for buy rules	int(4)	I6	indhdr_val%rules%buyfnct
SELLFNCT	MSELLFNCT	function code for sell rules	int(4)	I6	indhdr_val%rulesl%sellfnct
STATFNCT	MSTATFNCT	function code for calculating statistic	int(4)	I6	indhdr_val%rules%statfnct
GROUPFLAG	MGROUPFLAG	how stats are grouped before applying rules	int(4)	I6	indhdr_val%rules%groupflag
ASSIGNCODE	MASSIGNCODE	code of possible assignment types	int(4)	I6	indhdr_val%assign%assigncode
ASPERM	MASPERM	permno of associated index for breakpoints	int(4)	I6	indhdr_val%assign%asperm
ASPORT	MASPORT	portfolio number in associated index	int(4)	I6	indhdr_val%assign%asport
REBALCAL	MREBALCAL	calid of rebalancing calendar	int(4)	I6	indhdr_val%assign%rebalcal
ASSIGNCAL	MASSIGNCAL	calid of assignment calendar	int(4)	I6	indhdr_val%assign%assigncal
CALCCAL	MCALCCAL	calid of calculation range calendar	int(4)	I6	indhdr_val%assign%calccal

(M)REBAL

IND MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
REBAL	MREBAL	Index rebalancing period summary	CRSP_IND_REBAL_T		rebal_itm%arr%rebal_arr
RBBEGDT	MRBBEGDT	rebalancing beginning date	int(4)	I8	rebal_arr(i)%rbbegdt
RBENDDT	MRBENDDT	rebalancing ending date	int(4)	I8	rebal_arr(i)%rbenddt
RUSDCNT	MRUSDCNT	count used as of rebalancing	int(4)	I8	rebal_arr(i)%rusdcnt
MAXCNT	MMAXCNT	maximum count during period	int(4)	I8	rebal_arr(i)%maxcnt
RTOTCNT	MRTOTCNT	available count as of rebalancing	int(4)	I8	rebal_arr(i)%rtotcnt
ENDCNT	MENDCNT	count at end of period	int(4)	I8	rebal_arr(i)%endcnt
MINID	MMINID	identifier at minimum value	int(4)	I8	rebal_arr(i)%minid
MAXID	MMAXID	identifier at maximum value	int(4)	I8	rebal_arr(i)%maxid

IND MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
MINSTAT	MMINSTAT	smallest statistic in period	double(8)	F14.3	rebal_arr(i)%minstat
MAXSTAT	MMAXSTAT	largest statistic in period	double(8)	F14.3	rebal_arr(i)%maxstat
MEDSTAT	MMEDSTAT	median statistic in period	double(8)	F14.3	rebal_arr(i)%medstat
AVGSTAT	MAVGSTAT	average statistic in period	double(8)	F14.3	rebal_arr(i)%avgstat

(M)LIST

IND MNEMONIC		FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DAILY	MONTHLY				
LIST	MLIST	Issue list information	CRSP_IND_LIST_T		list_itm%arr%list_arr
TPERMNO	MTPERMNO	issue identifier	int(4)	I8	list_arr(i)%tpermno
LBEGDT	MLBEGDT	first date included	int(4)	I8	list_arr(i)%lbegdt
LENDDT	MLENDDT	last date included	int(4)	I8	list_arr(i)%lenddt
SUBIND	MSUBIND	code for subcategory of list	int(4)	I6	list_arr(i)%subind
LWEIGHT	MLWEIGHT	weight during range	double(8)	F16.5	list_arr(i)%lweight
TPERMNO	MTPERMNO	issue identifier	int(4)	I8	list_arr(i)%tpermno
LBEGDT	MLBEGDT	first date included	int(4)	I8	list_arr(i)%lbegdt
LENDDT	MLENDDT	last date included	int(4)	I8	list_arr(i)%lenddt
SUBIND	MSUBIND	code for subcategory of list	int(4)	I6	list_arr(i)%subind
LWEIGHT	MLWEIGHT	weight during range	double(8)	F16.5	list_arr(i)%lweight

CRSP/COMPUSTAT MERGED DATABASE

To connect to the specific CRSP CCM database instance the path to its database root should be specified. When installed on your system, CRSP CCM data set will be assigned an environment variable pointing to the CRSP CCM database root.

Additionally, an application ID should be specified on the call to crsp_f_itm_init to indicate the item-universe to be loaded for the session and describes the available items and item groups, eg:

```
sts = crsp_f_itm_init (hndl,'CRSP_CCM',app_id,'ccm1')
```

User-programs should access the CRSP CCM data set with the `app_id` as listed in the following table:

CCM ROOT/APP ID	FORTRAN 95 TYPE	DESCRIPTION
CRSP_CCM		CCM/CRSP Compustat data set
CRSP_CCMITEMS_ID	integer	Compustat Xpressfeed data items and groups

The details on included items and item groups can be found starting on page 47.

ACCESS KEYS

CRSP Compustat Xpressfeed includes various data on companies, securities, and indexes. Access key is composed of access key items the values of which can be retrieved or set from the user-program to control the direct access to the CCM data.

Default access key for CRSP CCM is loaded automatically on opening the access session to the CRSP CCM data set.

Additionally, a set of alternative access keys (and associated key items) is defined to facilitate access to the data by CRSP PERMNO, CUSIP and other keys.

The current key universe can be retrieved by requesting header information and sequentially traversing the whole data set on the selected access key. The function `crsp_f_itm_get_key` can also be used to retrieve the value of the access key items for the currently read record.

To switch to access by an alternative key, a user calls `crsp_f_itm_load_key` to set the access key index, followed by calls to `crsp_itm_set_key` to set the value of the key items used on subsequent reading of the database.

The defined CCM access keys and associated key items are listed in the following table:

CCM ACCESS KEY/KEY ITEMS	FORTRAN 95 TYPE	DESCRIPTION	NOTES
gvkey		Compustat GVKEY and IID	default
KYGVKEY	integer	Compustat company's GVKEY	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
gvkeyx		Compustat permanent identifier for indexes	
KYGVKEYX	integer	Compustat index's GVKEYX	primary key item
ccmid		Compustat permanent identifier - either GVKEY or GVKEYX	
KYCCMID	integer	CRSP CCMID (GVKEY or GVKEYX as reported in MASTER item)	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
permco		CRSP historical PERMCO Link	
KYPERMCO	integer	CRSP company's PERMCO	primary key item
permno		CRSP historical PERMNO Link	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item
cusip		Compustat CUSIP	
KYCUSIP	char(CRSP_CCM_CUSIP_LEN)	Compustat issue's CUSIP	primary key item
ticker		Compustat reported Issue Trading Symbol selects GVKEY and security	
KYTICKER	char(CRSP_CCM_TIC_LEN)	Compustat issue's ticker	primary key item
sic		Compustat -reported SIC code. Security or Company	
KYSIC	integer	Compustat security's SIC	primary key item
KYIID	char(CRSP_CCM_IID_LEN)	Compustat company security's IID	secondary key item
apermno		Link-Used PERMNO	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item
apermco		Link-Used PERMCO	
KYPERMCO	integer	CRSP company issue's PERMCO	primary key item
ppermno		CRSP PERMNO when security is marked as primary	
KYPERMNO	integer	CRSP company issue's PERMNO	primary key item

DATA TYPES

Generally, individual Compustat Xpressfeed data items are of common simple Fortran 95 data types and stored data can be accessed through `itm%arr` and corresponding scalar or array member.

Also additional character data types were introduced to store specific classes of Xpressfeed items, as listed in the following table:

ITEM OBJECT TYPE	FORTRAN 95 TYPE NAME	INTERNAL STORAGE	ACCESS VIA CRSP_ ITM_T	NOTES
time-series	CRSP_TS_T		itm%obj%ts	
	CRSP_CCM_FTNT_T	char(CRSP_CCM_FTNT_LEN)	itm%arr%ftnt_arr(i)%ftnt	Used for CCM footnote items, mainly time-series
	CRSP_CCM_TEXTITEM_T	char(CRSP_CCM_TEXTITEM_LEN)	itm%arr%text_arr(i)%text	Used for various CCM character string items, mainly time-series
array	CRSP_ARRAY_T		itm%obj%arr	
	CRSP_CCM_FTNT_T	char(CRSP_CCM_FTNT_LEN)	itm%arr%ftnt_arr(i)%ftnt	
	CRSP_CCM_TEXTITEM_T	char(CRSP_CCM_TEXTITEM_LEN)	itm%arr%text_arr(i)%text	
row	CRSP_ROW_T		itm%obj%row	
	CRSP_CCM_FTNT_T	char(CRSP_CCM_FTNT_LEN)	itm%arr%ftnt_val%ftnt	
	CRSP_CCM_TEXTITEM_T	char(CRSP_CCM_TEXTITEM_LEN)	itm%arr%text_val%text	

Additionally, selected Compustat Xpressfeed primary data groups and CRSP supplemental data groups can be accessed by the entire group as a defined structured type rather than as a stand-alone item. These data groups and their elements can both be accessed by `itm_name`, but recommended programming access is through the itm_name of the structure. To access the structured type and its fields, load the structured type `itm_name` during initialization, create a `CRSP_ITM_T` pointer matching the `itm_name`, attach it to the data, and access the structured type and its fields through the pointer:

```
sts = crsp_f_itm_load(hndl,'MASTER',match_flag)

sts = crsp_f_itm_find(hndl,'MASTER',0,mstr_itm)

ccmid = mstr_itm%arr%master_val%ccmid

...
```

STRUCTURED TYPES FOR CRSP/COMPUSTAT MERGED DATABASE ACCESS

The tables below show the data groups available as CCM structured types and their usage through the `CRSP_ITM_T` type.

CCM MNEMONIC	DESCRIPTION	FORTRAN 95 TYPE NAME	OBJECT TYPE	OBJECT ACCESS VIA CRSP_ ITM_T
MASTER	CCM company id and range data	CRSP_CCM_MASTER_T	row	master_itm%obj%row
COMPANY	CCM company header information	CRSP_CCM_COMPANY_T	row	company_itm%obj%row
IDX_INDEX	CCM idx_index header information	CRSP_CCM_IDX_INDEX_T	row	idx_index_itm%obj%row
SPIND	S&P index header (pre-GICS)	CRSP_CCM_SPIND_T	row	spind_itm%obj%row
COMPHIST	CCM company header history	CRSP_CCM_COMPHIST_T	array	comphist_itm%obj%arr
CSTHIST	CST header history	CRSP_CST_NAME_T	array	csthist_itm%obj%arr
LINK	CRSP CCM link history	CRSP_CCM_LINK_T	array	link_itm%obj%arr
LINKUSED	CCM company CRSP link used data	CRSP_CCM_LINKUSED_T	array	linkused_itm%obj%arr
LINKRNG	CCM company CRSP link range data	CRSP_CCM_LINKRNG_T	array	linkused_itm%obj%arr
ADJFACT	CCM company adjustment factor history	CRSP_CCM_ADJFACT_T	array	adjfact_itm%obj%arr
HGIC	CCM company GICS code history	CRSP_CCM_HGIC_T	array	hgic_itm%obj%arr
OFFTITL	CCM company officer title data	CRSP_CCM_OFFTITL_T	array	offtitl_itm%obj%arr
CCM_FILEDATE	CCM company filing date data	CRSP_CCM_FILEDATE_T	array	ccm_filedate_itm%obj%arr
CCM_IPCD	CCM industry presentation code data	CRSP_CCM_IPCD_T	array	ccm_ipcd_itm%obj%arr
SECURITY	CCM security header information	CRSP_CCM_SECURITY_T	row	security_itm%obj%row
SECHIST	CCM security header history	CRSP_CCM_SECHIST_T	array	sechist_itm%obj%arr

CCM MNEMONIC	DESCRIPTION	FORTRAN 95 TYPE NAME	OBJECT TYPE	OBJECT ACCESS VIA CRSP_ITM_T
SEC_MTHSPT	CCM security monthly split events	CRSP_CCM_SEC_MTHSPT_T	row	sec_mthspt_itm%obj%row
SEC_MSPT_FN	CCM security monthly split event footnotes	CRSP_CCM_SEC_MTH_FN_T	row	sec_mspt_fn_itm%obj%row
SEC_MDIV_FN	CCM security monthly dividend event footnotes	CRSP_CCM_SEC_MTH_FN_T	row	sec_mdiv_fn_itm%obj%row
SEC_SPIND	CCM security S&P information events	CRSP_CCM_SEC_SPIND_T	row	sec_spind_itm%obj%row
IDXCST_HIS	CCM security historical index constituents	CRSP_CCM_IDXCST_HIS_T	array	idxcst_his_itm%obj%arr
SPIDX_CST	CCM security S&P index constituent events	CRSP_CCM_SPIDX_CST_T	array	spidx_cst_itm%obj%arr
CCM_SEGCUR	CCM opseg currency rate data	CRSP_CST_SEGCUR_T	array	ccm_segcur_itm%obj%arr
CCM_SEGSRC	CCM opseg source data	CRSP_CST_SEGSRC_T	array	ccm_segsrc_itm%obj%arr
CCM_SEGPROD	CCM opseg product data	CRSP_CST_SEGPROD_T	array	ccm_segprod_itm%obj%arr
CCM_SEGCUST	CCM opseg customer data	CRSP_CST_SEGCUST_T	array	ccm_segcust_itm%obj%arr
CCM_SEGDTL	CCM opseg detail data	CRSP_CST_SEGDTL_T	array	ccm_segdtl_itm%obj%arr
CCM_SEGITM	CCM opseg item data	CRSP_CST_SEGITM_T	array	ccm_seg_itm%obj%arr
CCM_SEGNAICS	CCM opseg NAICS data	CRSP_CST_SEGNAICS_T	array	ccm_segnaics_itm%obj%arr
CCM_SEGGEO	CCM opseg geographic data	CRSP_CST_SEGGEO_T	array	ccm_seggeo_itm%obj%arr

MASTER

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
MASTER				master_itm%arr%master_val
BEGQTR	Quarterly date of earliest data (yyyy.q)	int(4)	I6	master_val%begqtr
BEGYR	Annual date of earliest data (yyyymmdd)	int(4)	I4	master_val%begyr
CBEGDT	First date of Compustat data	int(4)	I8	master_val%cbegdt
CCMID	Permanent record identifier for Compustat company or index data, represents GVKEY for company, GVKEYX for index	int(4)	I6	master_val%ccmid
CCMIDTYPE	Type of key for Compustat data. 1 = company data, 2 = index data	int(4)	I2	master_val%ccmidtype
CENDT	Last date of Compustat data	int(4)	I8	master_val%cendt
ENDQTR	Quarterly date of last data (yyyy.q)	int(4)	I6	master_val%endqtr
ENDYR	Annual date of last data (yyyymmdd)	int(4)	I4	master_val%endyr

COMPANY

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
COMPANY				company_itm%arr%company_val
ADD1-4	Address lines 1-4	char(68)	A65	company_val%add#
ADDZIP	Postal code	char(24)	A24	company_val%addzip
BUSDESC	Business description	char(2000)	A2000	company_val%busdesc
CIK	CIK number	char(12)	A10	company_val%cik
CITY	City	char(104)	A104	company_val%city
CONM	Company name	char(256)	A255	company_val%conm
CONML	Company legal name	char(104)	A100	company_val%conml
COSTAT	Postal code	char(24)	A1	company_val%addzip
COUNTY	County code	char(104)	A100	company_val%county
DLDTE	Research company deletion date	int(4)	I8	company_val%dldte

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
DLRSN	Research company reason for deletion	char(12)	A8	company_val%dlrsn
EIN	Employer identification number	char(12)	A10	company_val%ein
FAX	Fax number	char(24)	A18	company_val%fax
FIC	ISO Country code of incorporation	char(16)	A3	company_val%fic
FYRC	Fiscal year end (current)	int(4)	I2	company_val%fyrc
GGROUP	GICS groups	char(12)	A4	company_val%ggroup
GIND	GICS industries	char(12)	A6	company_val%gind
GSECTOR	GICS sectors	char(12)	A2	company_val%gsector
GSUBIND	GICS sub-industries	char(12)	A8	company_val%gsubind
IDBFLAG	International/Domestic/Both indicator	char(12)	A1	company_val%idbflag
INCORP	State/Province of incorporation code	char(12)	A8	company_val%incorp
IPODATE	Company initial public offering date	int(4)	I8	company_val%ipodate
LOC	ISOCountry code/ headquarters	char(4)	A3	company_val%loc
NAICS	North American Industry Classification Code	char(8)	A6	company_val%naics
PHONE	Phone number	char(24)	A18	company_val%phone
PRICAN	Primary Issue Tag - Canada	char(12)	A8	company_val%prican
PRIROW	Primary Issue Tag – rest of world	char(12)	A8	company_val%prirow
PRIUSA	Primary Issue Tag - USA	char(12)	A8	company_val%priusa
SIC	SIC code	int(4)	I4	company_val%sic
SPCINDCD	S&P industry sector code - reference	int(4)	I4	company_val%spcindcd
SPCSECCD	S&P economic sector code - reference	int(4)	I4	company_val%spcseccd
STATE	State/Province	char(12)	A8	company_val%state
STKO	Stock ownership code	int(4)	I1	company_val%stko
WEBURL	Website address	char(68)	A60	company_val%weburl

IDX_INDEX

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
IDX_INDEX				idx_index_itm%arr%idx_index_val
IDX13KEY	13 character key	char(16)	A13	idx_index_val%idx13key
IDXCSTFLG	Index constituent flag	char(4)	A2	idx_index_val%idxcstflg
IDXSTAT	Index Status	char(2)	A1	Idx_index_val%idxstat
INDEXCAT	Index category code	char(12)	A10	idx_index_val%indexcat
INDEXGEO	Index geographical area	char(12)	A10	idx_index_val%indexgeo
INDEXTYPE	Index type	char(12)	A10	idx_index_val%indextype
INDEXVAL	Index value	char(12)	A10	idx_index_val%indexval
SPII	S&P industry index code	int(4)	I4	idx_index_val%spii
SPMI	S&P major index code	int(4)	I4	idx_index_val%spmi
TICI	Issue trading ticker	char(12)	A8	idx_index_val%tici
XCONM	Company Name (Index)	char(256)	A255	idx_index_val%xconm
XINDEXID	Index ID	char(12)	A12	idx_index_val%xindexid
XTIC	Ticker/trading symbol (index)	char(10)	A10	idx_index_val%xtic

SPIND

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SPIND				spind_itm%arr%spind_val
SPIID	S&P Industry ID	int(4)	I4	spind_val%spiid
SPIMID	S&P Major Index ID	int(4)	I4	spind_val%spimid
SPITIC	S&P Index ticker	char(12)	A12	spind_val%spitic
SPIDESC	S&P Index industry description /reference	char(256)	A256	spind_val%spidesc

COMPHIST

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
COMPHIST				comphist_itm%arr%comphist_arr(i)
HCHGENDDT	Comphist description last effective date	int(4)	I8	comphist_arr(i)%hchgenddt
HDLDTE	Historical research company – deletion date	int(4)	I8	comphist_arr(i)%hdldte
HFYRC	Historical fiscal year end month / current	int(4)	I10	comphist_arr(i)%hfyrc
HIPODATE	Historical company official public offering date	int(4)	I10	comphist_arr(i)%hipodate
HSIC	Historical SIC Code	int(4)	I10	comphist_arr(i)%hsic
HSPCINDCD	Historical S&P Industry code	int(4)	I10	comphist_arr(i)%hspcindcd
HSPCSECCD	Historical S&P Economic sector code	int(4)	I10	comphist_arr(i)%hspcseccd
HSTKO	Historical stock ownership code	int(4)	I10	comphist_arr(i)%hstko
HADD1…4	Historical address lines 1-4	char(68)	A68	comphist_arr(i)%haddl#
HADDZIP	Historical postal code	char(68)	A24	comphist_arr(i)%haddzip
HBUSDESC	Historical business description	char(2000)	A2000	comphist_arr(i)%hbusdesc
HCIK	Historical CIK number	char(12)	A12	comphist_arr(i)%hcik
HCITY	Historical city	char(104)	A104	comphist_arr(i)%hcity
HCONM	Historical company name	char(256)	A256	comphist_arr(i)%hconm
HCONML	Historical legal company name	char(104)	A104	comphist_arr(i)%hconml
HCOSTAT	Historical active/inactive status marker	char(4)	A4	comphist_arr(i)%hcostat
HCOUNTY	Historical county code	char(104)	A1044	comphist_arr(i)%hcounty
HDLRSN	Historical research company reason for deletion	char(12)	A12	comphist_arr(i)%hdlrsn
HEIN	Historical employer identification number	char(12)	A12	comphist_arr(i)%hein
HFAX	Historical fax number	char(16)	A16	comphist_arr(i)%hfax
HFIC	Historical ISO country code / incorporation	char(16)	A16	comphist_arr(i)%hfic
HGGROUP	Historical GICS group	char(12)	A12	comphist_arr(i)%hggroup
HGIND	Historical GICS industries	char(12)	A12	comphist_arr(i)%hgind
HGSECTOR	Historical GICS sector	char(12)	A12	comphist_arr(i)%hgsector
HGSUBIND	Historical GICS sub-industries	char(12)	A12	comphist_arr(i)%hgsubind
HIDBFLAG	Historical international, domestic, both indicator	char(12)	A12	comphist_arr(i)%hidbflag
HINCORP	Historical state/province of incorporation code	char(12)	A12	comphist_arr(i)%hincorp
HLOC	Historic ISO country code/ headquarters	char(4)	A4	comphist_arr(i)%hloc
HNAICS	Historical NAICS codes	char(8)	A8	comphist_arr(i)%hnaics
HPHONE	Historical phone number	char(16)	A16	comphist_arr(i)%hphone
HPRICAN	Historical primary issue tag - Cananda	char(12)	A12	comphist_arr(i)%hprican
HPRIROW	Historical primary issue tag – rest of world	char(12)	A12	comphist_arr(i)%hprirow

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
HPRIUSA	Historical primary issue tag - US	char(12)	A12	comphist_arr(i)%hpriusa
HSTATE	Historical state/province	char(12)	A12	comphist_arr(i)%hstate
HWEBURL	Historical website url	char(68)	A68	comphist_arr(i)%hweburl

CSTHIST

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CSTHIST				csthist_itm%arr%csthist_arr(i)
CST_CHGDT	CST History effective date	int(4)	I8	csthist_arr(i)%cst_chgdt
CST_CHGENDDT	CST History last effective date	int(4)	I8	csthist_arr(i)%cst_chgenddt
CST_DNUM	CST History industry code	int(4)	I4	csthist_arr(i)%cst_dnum
CST_FILE	CST History file identification code	int(4)	I4	csthist_arr(i)%cst_file
CST_ZLIST	CST History exchange listing and S&P Index code	int(4)	I4	csthist_arr(i)%cst_zlist
CST_STATE	CST History state identification code	int(4)	I4	csthist_arr(i)%cst_state
CST_COUNTY	CST History county identification code	int(4)	I4	csthist_arr(i)%cst_county
CST_STINC	CST History state incorporation code	int(4)	I4	csthist_arr(i)%cst_stinc
CST_FINC	CST History foreign incorporation code	int(4)	I4	csthist_arr(i)%cst_finc
CST_XREL	CST History industry index relative code	int(4)	I4	csthist_arr(i)%cst_xrel
CST_STK	CST History stock ownership code	int(4)	I4	csthist_arr(i)%cst_stk
CST_DUP	CST History duplicate file code	int(4)	I4	csthist_arr(i)%cst_dup
CST_CCNDX	CST History current Canadian index code	int(4)	I4	csthist_arr(i)%cst_ccndx
CST_GICS	CST History Global Industry Classification Standard Code	int(4)	I4	csthist_arr(i)%cst_gics
CST_IPODT	CST History IPO date	int(4)	I4	csthist_arr(i)%cst_ipodt
CST_FUNDF1	CST History fundamental file identification code 1	int(4)	I4	csthist_arr(i)%cst_fund1
CST_FUNDF2	CST History fundamental file identification code 2	int(4)	I4	csthist_arr(i)%cst_fundf2
CST_FUNDF3	CST History fundamental file identification code 3	int(4)	I4	csthist_arr(i)%cst_fundf3
CST_NAICS	CST History North American Industry Classification	char(8)	A8	csthist_arr(i)%cst_naics
CST_CPSPIN	CST History primary S&P Index marker	char(4)	A4	csthist_arr(i)%cst_cpspin
CST_CSSPIN	CST History subset S&P Index marker	char(4)	A4	csthist_arr(i)%cst_csspin
CST_CSSPII	CST History secondary S&P Index marker	char(4)	A4	csthist_arr(i)%cst_csspii
CST_SUBDBT	CST History current S&P subordinated debt rating	char(8)	A8	csthist_arr(i)%cst_subdbt
CST_CPAPER	CST History current S&P commercial paper rating	char(4)	A4	csthist_arr(i)%cst_cpaper
CST_SDBT	CST History current S&P senior debt rating	char(4)	A4	csthist_arr(i)%cst_sdbt
CST_SDBTIM	CST History current S&P senior debt rating - footnote	char(4)	A4	csthist_arr(i)%cst_sdbtim
CST_CNUM	CST History CUSIP issuer code	char(16)	A16	csthist_arr(i)%cst_cnum
CST_CIC	CST History issuer number	char(4)	A4	csthist_arr(i)%cst_cic
CST_CONAME	CST History company name	char(64)	A64	csthist_arr(i)%cst_coname
CST_INAME	CST History industry name	char(4)	A4	csthist_arr(i)%cst_iname
CST_SMBL	CST History stock ticker symbol	char(16)	A16	csthist_arr(i)%cst_smbl
CST_EIN	CST History employer identification number	char(16)	A16	csthist_arr(i)%cst_ein
CST_INCORP	CST History incorporation ISO country code	char(4)	A4	csthist_arr(i)%cst_incorp

LINK

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
LINK				link_itm%arr%link_arr(i)
LINKDT	Effective date of the link record	int(4)	I8	link_arr(i)%linkdt
LINKENDDT	Last effective date of the link record	int(4)	I8	link_arr(i)%linkenddt
LPERMNO	CRSP PERMNO link during link period	int(4)	I6	link_arr(i)%lpermno
LPERMCO	CRSP PERMCO link during link period	int(4)	I10	link_arr(i)%lpermco
LIID	Security identifier	char(4)	A3	link_arr(i)%liid
LNKTYPE	Link type code	char(4)	A4	link_arr(i)%lnktype
LINKPRIM	Primary security link marker	char(4)	A1	link_arr(i)%linkprim

LINKUSED

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
LINKUSED				linkused_itm%arr%linkused_arr(i)
ULINKDT	Effective date of the link	int(4)	I8	linkused_arr(i)%ulinkdt
ULINKENDDT	Last effective date of the link	int(4)	I8	linkused_arr(i)%ulinkenddt
ULINKID	Linkused row identifier	int(4)	I	linkused_arr(i)%ulinkid
UGVKEY	GVKEY used in the link	int(4)	I6	linkused_arr(i)%ugvkey
UPERMNO	CRSP PERMNO link during link period	int(4)	I6	linkused_arr(i)%upermno
UPERMCO	CRSP PERMCO link during link period	int(4)	I6	linkused_arr(i)%upermco
UIID	Used Security ID	char(4)	A3	linkused_arr(i)%uiid
USEDFLAG	Flag marking whether link is used in building composite record	int(4)	I	linkused_arr(i)%usedflag
ULINKPRIM	Used link primary marker	char(4)	A1	linkused_arr(i)%ulinkprim
ULINKTYPE	Used link type	char(4)	A4	linkused_arr(i)%ulinktype

LINKRNG

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
LINKRNG				linkrng_itm%arr%linkrng_arr(i)
RKEYSET	Keyset applicable to range	int(4)	I8	linkrng_arr(i)%rkeyset
RCALID	Calendar applicable to range	int(4)	I8	linkrng_arr(i)%rcalid
RBEGIND	Beginning time series range of link	int(4)	I8	linkrng_arr(i)%rbegind
RENDIND	Ending time series range of link	int(4)	I8	linkrng_arr(i)%rendind
RPREVIND	Time series range immediately preceding the link	int(4)	I8	linkrng_arr(i)%rprevind
RBEGDT	Beginning calendar range of link	int(4)	I8	linkrng_arr(i)%rbegdt
RENDDT	Ending calendar range of link	int(4)	I8	linkrng_arr(i)%renddt
RPREVDT	Ending calendar range preceding the link	int(4)	I8	linkrng_arr(i)%rprevdt
RFISCAL_DATA_FLG	Type of time series, C-calendar or F-fiscal.	char(8)	A8	linkrng_arr(i)%rfiscal_data_flg
EFFDATE	Effective date- company cumulative factor	int(4)	I10	linkrng_arr(i)%effdate
THRUDATE	Thu date – company cumulative factor	int(4)	I10	linkrng_arr(i)%thrudate

ADJFACT

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
ADJFACT				adjfact_itm%arr%adjfact_arr(i)
ADJEX	Cumulative adjustment factor by Ex-date	double(8)	F18.4	adjfact_arr(i)%adjex
ADJPAY	Cumulative adjustment factor by Pay-date	double(8)	F18.4	adjfact_arr(i)%adjpay

HGIC

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
HGIC				hgic_itm%arr%hgic_arr(i)
INDFROM	Effective from (start) date	int(4)	I8	hgic_arr(i)%indfrom
INDTHRU	Effective through (last) date	int(4)	I8	hgic_arr(i)%indthru
GGROUPH	Industry group name	char(12)	A12	hgic_arr(i)%ggrouph
GINDH	Group industry	char(12)	A12	hgic_arr(i)%gindh
GSECTORH	Group industry sector	char(12)	A12	hgic_arr(i)%gsectorh
GSUBINDH	Group sub-industries	char(12)	A12	hgic_arr(i)%gsubindh

OFFTITL

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
OFFTITL				offtitl_itm%arr%offtitl_arr(i)
OFID	Officer ID	int(4)	I9	offtitl_arr(i)%ofid
OFCD	Officer title	char(16)	A8	offtitl_arr(i)%ofcd
OFNM	Officer Name(s)	char(40)	A39	offtitl_arr(i)%ofnm

CCM_FILEDATE

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_FILEDATE				ccm_filedate_itm%arr%ccm_filedate_arr(i)
FDATADATE	Company filing data date	int(4)	I8	ccm_filedate_arr(i)%fdatadate
FCONSOL	Company consolidation level filedate	char(2)	A2	ccm_filedate_arr(i)%fconsol
FPOPSRC	Population source filedate	char(2)	A2	ccm_filedate_arr(i)%fpopsrc
SRCTYPE	Document source type filedate	char(12)	A12	ccm_filedate_arr(i)%srctype
FILEDATE	Company filing date	int(4)	I8	ccm_filedate_arr(i)%filedate

CCM_IPCD

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_IPCD				ccm_ipcd_itm%arr%ccm_ipcd_arr(i)
IPDATADATE	Industry presentation code data date	int(4)	I8	ccm_ipcd_arr(i)%ipdatadate
IPCONSOL	Level of consolidation (Industry presentation code)	char(2)	A1	ccm_ipcd_arr(i)%ipconsol
IPPOPSRC	Population source (Industry presentation code)	char(2)	A1	ccm_ipcd_arr(i)%ippopsrc
IPCD	Industry presentation code	char(2)	A1	ccm_ipcd_arr(i)%ipcd

SECURITY

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SECURITY				security_itm%arr%security_val
EXCHG	Stock exchange	int(4)	I4	security_val%exchg
DLDTEI	Security inactivation date	int(4)	I8	security_val%dltei
IID_SEQ_ NUM	IID sequence number	int(4)	I8	security_val%iid_seq_num
SBEGDT	First date of Compustat data for issue	int(4)	I8	security_val%sbegdt
SENDDT	Last date of Compustat data for issue	int(4)	I8	security_val%senddt
IID	Issue ID	char(4)	A3	security_val%iid
SCUSIP	CUSIP	char(12)	A12	security_val%cusip
DLRSNI	Security inactivation code	char(12)	A8	security_val%dlrsni
DSCI	Security description	char(32)	A28	security_val%dsci
EPF	Earnings participation flag	char(4)	A1	security_val%epf
EXCNTRY	Stock exchange country code	char(4)	A3	security_val%excntry
ISIN	International security identification number	char(16)	A12	security_val%isin
SSECSTAT	Security status marker	char(4)	A4	security_val%ssecstat
SEDOL	SEDOL	char(8)	A7	security_val%sedol
TIC	Ticker/trading symbol	char(12)	A8	security_val%tic
TPCI	Issue type	char(12)	A8	security_val%tpci

SECHIST

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SECHIST				sechist_itm%arr%sechist_arr(i)
HSCHGDT	Historical security change date	int(4)	I8	sechist_arr(i)%hschgdt
HSCHGENDDT	Historical security change end date	int(4)	I8	sechist_arr(i)%hschgenddt
HEXCHG	Historical stock exchange	int(4)	I10	sechist_arr(i)%hexchg
HDLDTEI	Historical security inactivation date	int(4)	I8	sechist_arr(i)%hdldtei
HIID_SEQ_NUM	Historical issue ID sequence number	int(4)	I10	sechist_arr(i)%hiid_seq_num
HIID	Historical issue ID	char(4)	A3	sechist_arr(i)%hiid
HSCUSIP	Historical CUSIP	char(12)	A12	sechist_arr(i)%hscusip
HDLRSNI	Historical security inactivation code	char(12)	A12	sechist_arr(i)%hdhdrsni
HDSCI	Historical security description	char(32)	A32	sechist_arr(i)%hdsci
HEPF	Historical earnings participation flag	char(4)	A4	sechist_arr(i)%hepf
HEXCNTRY	Historical stock exchange country code	char(4)	A4	sechist_arr(i)%hexcntry
HISIN	Historical international security identification number	char(16)	A16	sechist_arr(i)%hisin
HSSECSTAT	Historical security status marker	char(4)	A4	sechist_arr(i)%hssecstat
HSEDOL	Historical SEDOL	char(8)	A8	sechist(i)%hsedol
HTIC	Historical ticker/trading symbol	char(12)	A12	sechist_arr(i)%htic
HTPCI	Historical issue type	char(12)	A12	sechist_arr(i)%htpci

SEC_MTHSPT

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SEC_MTHSPT				sec_mthspt_itm%arr%sec_mthspt_arr(i)
DATADATEM	Monthly adjustment factor data date	int(4)	I10	sec_mthspt_arr(i)%datadatem
RAWPM	Raw adjustment factor – pay date - monthly	double(8)	F18.4	sec_mthspt_arr(i)%rawpm
RAWXM	Raw adjustment factor – ex date - monthly	double(8)	F18.4	sec_mthspt_arr(i)%rawxm

SEC_MSPT_FN

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SEC_MSPT_FN				sec_mspt_fn_itm%arr%sec_mspt_fn_ arr(i)
DATAITEMMF	Monthly split footnote dataitem	char(8)	A8	sec_mspt_fn_arr(i)%dataitemmf
RAWPM_FN1..FN5	Raw adjustment factor – pay date – monthly – footnotes 1-5	char(4)	A4	sec_mspt_fn_arr(i)%rawpm_fn1..fn5
RAWXM_FN1..FN5	Raw adjustment factor – ex date – monthly – footnotes 1-5	char(4)	A4	sec_mspt_fn_arr(i)%rawxm_fn1..fn5

SEC_MDIV_FN

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SEC_MDIV_FN				sec_mdiv_fn_itm%arr%sec_mdiv_fn_ arr(i)
DIVDATADATEMF	Monthly dividend footnote data date	int(4)	I10	sec_mdiv_fn_arr(i)%divdatadatemf
DIVDATAITEMMF	Monthly dividend footnote data item	char(8)	A8	sec_mdiv_fn_arr(i)%divdataitemmf
DVPSPM_FN1..FN5	Dividend per share – pay date – monthly – footnotes 1-5	char(4)	A4	sec_mdiv_fn_arr(i)%dvpspm_fn1..fn5
DVPSXM_FN1..FN5	Dividend per share – ex date – monthly – footnotes 1-5	char(4)	A4	sec_mdiv_fn_arr(i)%dvpsxm_fn1..fn5

SEC_SPIND

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SEC_SPIND				sec_spind_itm%arr%sec_spind_arr(i)
SPBEGDATE	S&P Index event beginning date	int(4)	I10	sec_spind_arr(i)%spbegdate
SPENDDATE	S&P Index event ending date	int(4)	I10	sec_spind_arr(i)%spenddate
SPHIID	S&P holdings industry index ID	int(4)	I4	sec_spind_arr(i)%sphiid
SPHMID	S&P holdings major index ID	int(4)	I4	sec_spind_arr(i)%sphmid
SPHSEC	S&P holdings sector code	int(4)	I4	sec_spind_arr(i)%sphsec
SPH100	S&P holdings S&P 100 marker	int(4)	I4	sec_spind_arr(i)%sph100
SPHCUSIP	S&P holdings CUSIP	char(12)	A9	sec_spind_arr(i)%sphcusip
SPHNAME	S&P holdings name	char(36)	A31	sec_spind_arr(i)%sphname
SPHTIC	S&P holdings ticker	char(12)	A8	sec_spind_arr(i)%sphtic
SPHVG	S&P holdings value/growth indicator	char(4)	A1	sec_spind_arr(i)%sphvg

IDXCST_HIS

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
IDXCST_HIS				idxcst_his_itm%arr%idxcst_his_arr(i)

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
XFROM	S&P constituent from event date	int(4)	I10	idxcst_his_arr(i)%xfrom
IDX13KEY	S&P 13 character key	char(16)	A13	idxcst_his_arr(i)%idx13key
XGVKETX	S&P constituent event index GVKEY	int(4)	I10	idxcst_his_arr(i)%xgvkeyx

SPIDX_CST

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SPIDX_CST				spidx_cst_itm%arr%spidx_cst_arr(i)
SXBEGDATE	S&P constituent event beginning date	int(4)	I10	spidx_cst_arr(i)%sxbegdate
SXENDDATE	S&P constituent event ending date	int(4)	I10	spidx_cst_arr(i)%sxenddate
SPFLOAT	S&P float shares	int(4)	I4	spidx_cst_arr(i)%spfloat
INDEXID	S&P major index ID	char(12)	A10	spidx_cst_arr(i)%indexid
EXCHGX	S&P constituent exchange	char(8)	A4	spidx_cst_arr(i)%exchgx
TICX	S&P holdings ticker	char(12)	A10	spidx_cst_arr(i)%ticx
CUSIPX	S&P constituent CUSIP	char(12)	A9	spidx_cst_arr(i)%cusipx
CONMX	S&P constituent name	char(44)	A40	spidx_cst_arr(i)%conmx
CONTYPE	S&P constituent type	char(12)	A10	spidx_cst_arr(i)%contype
CONVAL	S&P constituent value	char(12)	A10	spidx_cst_arr(i)%conval

CCM_SEGCUR

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGCUR				ccm_segcur_itm%arr%segcur_arr(i)
SC_DATAYR	Data year	int(4)	I4	segcur_arr(i)%sc_datayr
SC_DATAFYR	Data fiscal year end month	int(4)	I2	segcur_arr(i)%sc_datafyr
SC_CALYR	Data calendar year	int(4)	I4	segcur_arr(i)%sc_calyr
SC_SRCYFYR	Source fiscal year end month	int(4)	I2	segcur_arr(i)%sc_srcfyr
SC_XRATE	Period end exchange rate	double(8)	F16.8	segcur_arr(i)%sc_xrate
SC_XRATE12	12-month moving exchange rate	double(8)	F16.8	segcur_arr(i)%sc_xrate12
SC_SRCCUR	Source currency code	char(4)	A3	segcur_arr(i)%sc_srccur
SC_CURCD	ISO currency code (USD)	char(4)	A3	segcur_arr(i)%sc_curcd

CCM_SEGSRC

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGSRC				ccm_segsrc_itm%arr%segsrc_arr(i)
SS_SRCYR	Source year	int(4)	I4	segsrc_arr(i)%ss_srcyr
SS_SRCFYR	Source fiscal year end month	int(4)	I2	segsrc_arr(i)%ss_srcfyr
SS_CALYR	Data calendar year	int(4)	I4	segsrc_arr(i)%ss_calyr
SS_RCST1	Reserved 1	int(4)	I4	segsrc_arr(i)%ss_rcst1
SS_SSRCE	Source document code	char(4)	A2	segsrc_arr(i)%ss_ssrce
SS_SUCODE	Source update code	char(4)	A2	segsrc_arr(i)%ss_sucode
SS_CURCD	ISO currency code	char(4)	A3	segsrc_arr(i)%ss_curcd

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SS_SRCCUR	Source ISO currency code	char(4)	A3	segsrc_arr(i)%ss_srccur
SS_HNAICS	Segment primary historical NAICS	char(8)	A6	segsrc_arr(i)%ss_hnaics

CCM_SEGPROD

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGPROD				ccm_segprod_itm%arr%segprod_arr(i)
SP_SRCYR	Source year	int(4)	I4	segprod_arr(i)%sp_srcyr
SP_SRCFYR	Source fiscal year end month	int(4)	I2	segprod_arr(i)%sp_srcfyr
SP_PDID	Product identifier	int(4)	I4	segprod_arr(i)%sp_pdid
SP_PSID	Segment link – segment identifier	int(4)	I4	segprod_arr(i)%sp_psid
SP_PSALE	External revenues	float(4)	F10.4	segprod_arr(i)%sp_psale
SP_RCST1	Reserved 1	float(4)	F10.4	segprod_arr(i)%sp_rcst1
SP_PNAICS	Product NAICS code	char(8)	A6	segprod_arr(i)%sp_pnaics
SP_PSTYPE	Segment link- segment type	char(16)	A83	segprod_arr(i)%sp_pstype
SP_PNAME	Product name	char(64)	A64	segprod_arr(i)%sp_pname

CCM_SEGCUST

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGCUST				ccm_segcust_itm%arr%segcust_arr(i)
SC_SRCYR	Source year	int(4)	I4	segcust_arr(i)%sc_srcyr
SC_SRCFYR	Source fiscal year end month	int(4)	I2	segcust_arr(i)%sc_srcfyr
SC_CDID	customer identifier	int(4)	I4	segcust_arr(i)%sc_cdid
SC_CSID	Segment link – segment identifier	int(4)	I4	segcust_arr(i)%sc_csid
SC_CSALE	customer revenues	float(4)	F10.4	segcust_arr(i)%sc_csale
SC_RCST1	Reserved 1	int(4)	I4	segcust_arr(i)%sc_rcst1
SC_CTYPE	Customer type	char(16)	A8	segcust_arr(i)%sc_ctype
SC_CGEOCD	Geographic area code	char(16)	A8	segcust_arr(i)%sc_cgeocd
SC_CGEOAR	Geographic area type	char(16)	A8	segcust_arr(i)%sc_cgeoar
SC_CSTYPE	Segment link – segment type	char(16)	A8	segcust_arr(i)%sc_cstype
SC_CNAME	Customer name data	char(64)	A64	segcust_arr(i)%sc_cname

CCM_SEGDTL

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGDTL				ccm_segdtl_itm%arr%segdtl_arr(i)
SD_SRCYR	Source year	int(4)	I4	segdtl_arr(i)%sd_srcyr
SD_SRCFYR	Source fiscal year end month	int(4)	I2	segdtl_arr(i)%sd_srcfyr
SD_SID	Segment identifier	int(4)	I4	segdtl_arr(i)%sd_sid
SD_RCST1	Reserved 1	int(4)	I4	segdtl_arr(i)%sd_rcst1
SD_STYPE	Segment type	char(16)	A8	segdtl_arr(i)%sd_stype
SD_SOPTP1	Operating segment type 1	char(16)	A8	segdtl_arr(i)%sd_ctype

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
SD_SOPTP2	Operating segment type	char(16)	A8	segdtl_arr(i)%sd_cgeocd
SD_SGEOTP	Geographic segment type	char(16)	A8	segdtl_arr(i)%sd_cgeoar
SD_SNAME	Segment name	char(256)	A64	segdtl_arr(i)%sd_cname

CCM_SEGITM

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGITM				ccm_segitm_itm%arr%segitm_arr(i)
SI_DATYR	Data year	int(4)	I4	segitm_arr(i)%si_datyr
SI_FISCYR	Data fiscal year end month	int(4)	I4	segitm_arr(i)%si_fiscyr
SI_SRCYR	Source year	int(4)	I4	segitm_arr(i)%si_srcyr
SI_SRCFYR	Source fiscal year end month	int(4)	I2	segitm_arr(i)%si_crsfyr
SI_CALYR	Data calendar year	int(4)	I4	segitm_arr(i)%si_calyr
SI_SID	Segment identifier	int(4)	I4	segitm_arr(i)%si_sid
SI_EMP	Employees	int(4)	I9	segitm_arr(i)%si_emp
SI_SALE	Net sales	float(4)	F10.4	segitm_arr(i)%si_sale
SI_OIBD	Operating income before depreciations	float(4)	F10.4	segitm_arr(i)%si_oibd
SI_DP	Depreciation & amortization	float(4)	F10.4	segitm_arr(i)%si_dp
SI_OIAD	Operating income after depreciation	float(4)	F10.4	segitm_arr(i)%si_oiad
SI_CAPX	Capital expenditures	float(4)	F10.4	segitm_arr(i)%si_capx
SI_IAT	Identifiable assets	float(4)	F10.4	segitm_arr(i)%si_iat
SI_EQEARN	Equity in earnings	float(4)	F10.4	segitm_arr(i)%si_eqearn
SI_INVEQ	Investments at equity	float(4)	F10.4	segitm_arr(i)%si_inveq
SI_RD	Research & development	float(4)	F10.4	segitm_arr(i)%si_rd
SI_OBKLG	Order backlog	float(4)	F10.4	segitm_arr(i)%si_obklg
SI_EXPORTS	Export sales	float(4)	F10.4	segitm_arr(i)%si_exports
SI_INTSEG	Inter-segment eliminations	int(4)	I4	segitm_arr(i)%si_intseg
SI_OPINC	Operating profit	float(4)	F10.4	segitm_arr(i)%si_opinc
SI_PI	Pretax income	float(4)	F10.4	segitm_arr(i)%si_pi
SI_IB	Income before extraordinary earnings	float(4)	F10.4	segitm_arr(i)%si_ib
SI_NI	Net income (loss)	float(4)	F10.4	segitm_arr(i)%si_ni
SI_RCST1	Reserved 1	float(4)	F10.4	segitm_arr(i)%si_rcst1
SI_RCST2	Reserved 2	float(4)	F10.4	segitm_arr(i)%si_rcst2
SI_RCST3	Reserved 3	float(4)	F10.4	segitm_arr(i)%si_rcst3
SI_SALEF	Footnote 1 - sales	char(16)	A8	segitm_arr(i)%si_salef
SI_OPINCF	Footnote 2 – operating profit	char(16)	A8	segitm_arr(i)%si_opincf
SI_CAPXF	Footnote 3 – capital expenditures	char(16)	A8	segitm_arr(i)%si_capxf
SI_EQEARNF	Footnote 4 – equity in earnings	char(16)	A8	segitm_arr(i)%si_eqearnf
SI_EMPF	Footnote 5 - employees	char(16)	A8	segitm_arr(i)%si_empf
SI_RDF	Footnote 6 – research & development	char(16)	A8	segitm_arr(i)%si_rdf
SI_STYPE	Segment type	char(16)	A8	segitm_arr(i)%si_stype

CCM_SEGNAICS

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGNAICS				ccm_segnaics_itm%arr%segnaics_arr(i)
SN_SRCYR	Source year	int(4)	I4	segnaics_arr(i)%sn_srcyr
SN_SRCFYR	Source fiscal year end month	int(4)	I2	segnaics_arr(i)%sn_srcfyr
SN_SID	Segment identifier	int(4)	I4	segnaics_arr(i)%sn_sid
SN_RCST1	Reserved 1	int(4)	I4	segnaics_arr(i)%sn_rcst1
SN_STYPE	Segment type	char(16)	A8	segnaics_arr(i)%sn_stype
SN_SNAICS	NAICS code	char(8)	A6	segnaics_arr(i)%sn_snaics
SN_RANK	Ranking	int(4)	I4	segnaics_arr(i)%sn_rank
SN_SIC	Segment SIC code	int(4)	I4	segnaics_arr(i)%sn_sic

CCM_SEGGEO

CCM MNEMONIC	FIELD NAME	INTERNAL STORAGE	DISPLAY FORMAT	FIELD USAGE
CCM_SEGGEO				ccm_seggeo_itm%arr%seggeo_arr(i)
SG_SRCYR	Source year	int(4)	I4	seggeo_arr(i)%sg_srcyr
SG_SRCFYR	Source fiscal year end month	int(4)	I2	seggeo_arr(i)%sg_srcfyr
SG_SID	Segment identifier	int(4)	I4	seggeo_arr(i)%sg_sid
SG_RCST1	Reserved 1	int(4)	I4	seggeo_arr(i)%sg_rcst1
SG_STYPE	Segment type	char(16)	A8	seggeo_arr(i)%sg_stype
SG_SGEOCD	Geographic area code	char(16)	A8	seggeo_arr(i)%sg_sgeocd
SG_SGEOTP	Geographic area type	char(16)	A8	seggeo_arr(i)%sg_sgeotp

CRSP FORTRAN 95 API FUNCTIONS

This section contains an alphabetical list of the functions defined in the CRSP Fortran 95 API. Each definition presents the following information about a function:

- Its prototype
- A list of arguments
- A list of return values
- Side effects
- Preconditions

CRSP_F_ITM_CLOSE

`crsp_f_itm_close` frees all item lists and item indexes, clears all calendar and key lists, closes the database, frees the handle set, and re-initializes the item access handle itself.

PROTOTYPE:	integer function crsp_f_itm_close (hndl)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle to close.
RETURN VALUES:	CRSP_SUCCESS: If the database is successfully closed and all handle data are free CRSP_FAIL: If there is an error in the parameters, inconsistent handle, error closing databases.
SIDE EFFECTS:	If successful, the handle data are emptied: • The database will be closed and the structure cleared. • All internal storage allocated for this instance will be freed
PRECONDITIONS:	The item handle must be previously opened with function crsp_f_itm_init.

Example:

```
if (crsp_f_itm_close(hndl) == CRSP_FAIL) then

   !!--error

         print *, 'Error-- failed to close db:',TRIM(dbpath)

         stop

   endif
```

CRSP_F_ITM_FIND

`crsp_f_itm_find` attaches a pointer to a `CRSP_ITM_T` item that was previously loaded. The `CRSP_ITM_T` structure describes the data item and contains the underlying time series, array, or row data.

PROTOTYPE:	integer function crsp_f_itm_find (hndl, itm_name, keyset,itm_foundptr)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. character(*) :: itm_name – String containing the itm_name to find. integer :: keyset – Keyset to find type(CRSP_ITM_T), pointer :: itm_foundptr – User-declared pointer that will point to the data item found.

RETURN VALUES:	CRSP_SUCCESS: If successfully found the requested item in the given keyset. CRSP NOT FOUND: If the itm_name and keyset combination are not available CRSP_FAIL: If error in parameters, handle not initialized, or error searching for the item.
SIDE EFFECTS:	If successful, the itm_foundptr will point to a CRSP_ITM_T item with data and information for the desired item and keyset. Otherwise the null() will be assigned to itm_foundptr.
PRECONDITIONS:	The item handle set must be initialized, loaded with a list of items, and opened.

Example:

```
if (crsp_f_itm_find(hndl,'HEADER',0,stkhdr_itm) == CRSP_FAIL          &
    .or. crsp_f_itm_find(hndl,'PRC',0,prc_itm) == CRSP_FAIL        &
    .or. crsp_f_itm_find(hndl,'ADJPRC',0,adjprc_itm) == CRSP_FAIL  &
    .or. crsp_f_itm_find(hndl,'ADJSHR',0,adjshr_itm) == CRSP_FAIL  &
    .or. crsp_f_itm_find(hndl,'ADJVOL',0,adjvol_itm) == CRSP_FAIL  &
    .or.  .not. associated(stkhdr_itm)  &
    .or.  .not. associated(prc_itm)     &
    .or.  .not. associated(adjprc_itm)  &
    .or.  .not. associated(adjshr_itm)  &
    .or.  .not. associated(adjvol_itm)) then
    print *,'Error - invalid item/keyset specified'
    stop
endif
```

CRSP_F_ITM_FIND_ITMCAL

`crsp_f_itm_find_itmcal` attaches a pointer to a `CRSP_ITM_CAL_T` item calendar that was previously loaded. The `CRSP_ITM_CAL_T` structure describes a global calendar or a calendar associated with an item and contains the underlying `CRSP_CAL_T` data.

PROTOTYPE:	integer function crsp_f_itm_find_itmcal (hndl, calid, keyset,itmcal_foundptr)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list.
	Integer :: calid – Calendar id of the calendar to find: • CRSP_CALID_DAILY – daily calendar • CRSP_CALID_MONTHLY – monthly calendar • CRSP_CALID_ANNUAL – annual calendar • CRSP_CALID_QUARTERLY – quarterly calendar • CRSP_CALID_SEMIANNUAL – semi-annual calendar • CRSP_CALID_WEEKLY – weekly calendar integer :: keyset – Keyset of the calendar to find. • keyset >= 0 – when item-access configured with fiscal_disp_cd='C', the calendars associated with fiscal items will be "shifted", based on the loaded company's FYE. • keyset=-1 – will request the global "non-shifted" calendar with the specified calid. type(CRSP_ITM_CAL_T), pointer :: itmcal_foundptr – User-declared pointer that will point to the calendar found.

RETURN VALUES:	CRSP_SUCCESS: If successfully found the requested item calendar in the given keyset. CRSP NOT FOUND: If the calid and keyset combination are not available CRSP_FAIL: If error in parameters, handle not initialized, or error searching for the item calendar.
SIDE EFFECTS:	If successful, the itmcal_foundptr will point to a CRSP_ITM_CAL_T item calendar with data and information for the desired calendar and keyset. Otherwise the null() will be assigned to itmcal_foundptr.
PRECONDITIONS:	The item handle set must be initialized, loaded with a list of items, and itmed.

CRSP_F_ITM_FIND_ITMCAL_DT

`crsp_f_itm_find_itmcal_dt` finds the array index of a date entry in `CRSP_ITM_CAL_T` item calendar.

PROTOTYPE:	integer function crsp_f_itm_find_itmcal_dt (itmcal,dt,match_mode,ifound)
ARGUMENTS:	type(CRSP_ITM_CAL_T), pointer :: itmcal – Pointer to the item calendar. integer :: dt – Requested date to be found. Format: yyymmdd. integer :: match_mode – Matching mode: • CRSP_EXACT – exact match requested • CRSP_NEXT – if exact is not found – return next valid • CRSP_PREV – if exact is not found – return previous valid integer ::ifound – Array index of the date found. When date is not found, ifound = -1
RETURN VALUES:	CRSP_SUCCESS: If successfully found the requested date. CRSP NOT FOUND: If the date is not found with the given matching mode. CRSP_FAIL: If error in parameters, item calendar is not set or loaded.
SIDE EFFECTS:	If successful, the ifound will be set to a valid array index for the caldt array attached to calendar object. The index can be used to directly access the corresponding time-series item data values associated with this calendar.
PRECONDITIONS:	The item calendar set must be initialized and loaded.

CRSP_F_ITM_GET_KEY

`crsp_f_itm_get_key` retrieves key information for data loaded by a function `crsp_f_itm_read` call. An output key item list is prepared when the key is initialized, and loaded by function `crsp_f_itm_read`. This function finds the `key_itm_name` in the list and copies ithe value into the user-specified location.

PROTOTYPE:	integer function crsp_f_itm_get_key (hndl, key_itm_name,keyval)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. character(*) :: key_itm_name – String containing an itm_name of a loaded key to be retrieved. {integer OR type(CRSP_VARSTRING_T)} :: keyval – Variable to accept the value of the key item. Data type must agree with the item's type and size.
RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_FAIL: If error in parameters, handle not open, key item.
SIDE EFFECTS:	If successful, the keyval is loaded based on the item and key value type.
PRECONDITIONS:	The item handle must be initialized and opened. The item key array must be initialized based on a keytype with the function crsp_f_itm_open or function crsp_f_itm_init_key functions. The key_itm_name must be a valid item for that keytype, and the keyval data must agree with the type of that item.

CRSP_F_ITM_INIT

`crsp_f_itm_init` prepares a handle for item access operation for one database and one application id. The handle will be initialized and the database set type and set id identified, allowing loading of reference data and allocation of a set structure.

PROTOTYPE:	integer function crsp_f_itm_init(hndl, dbpath,`app_id`, hndl_name)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle that will be used to manage the database information and item lists. character(*) :: dbpath – Path to database containing the data to load and the applicable reference data. integer :: app_id – Identifier of a defined application organizing data items into groups for access. Available app_ids can be found in the reference array, function crsp_f_itm_ app. Common app_ids have defined constants: • CRSP_CCMITEMS_ID – generic CCM usage application • CRSP_DSTKITM_ID – generic Daily Stock usage application • CRSP_MSTKITM_ID – generic Monthly Stock usage application • CRSP_DINDITEMS_ID – generic Daily Ind Stock usage application • CRSP_MINDITEMS_ID – generic Monthly Ind Stock usage application character(*) :: hndl name – Name to assign to the handle
RETURN VALUES:	CRSP_SUCCESS: If initialized successfully CRSP_FAIL: If there is an error in the parameter, database cannot be opened, reference data unavailable, incompatibility between database and `app_id`.
SIDE EFFECTS:	If successful, the handle data are loaded: • The handle fields are initialized, including all lists and arrays. • The ca_ref structure is loaded with the reference data in the database. If an old database with no reference data, it will use a global reference file with a standard name based on the `app_id` in the CRSP_LIB directory. • Itm_grp and itm_avail arrays in the handle are loaded with available tables and items • Set_list element is allocated using the database path and setid. The database is opened with a 0 wanted, which loads reference data but allocates no module space. The root information is loaded to the set's CRSP_ROOT_INFO_T structure.
PRECONDITIONS:	The item handle object must be declared and not attached to another access instance. The app_id must exist in the reference data of the database opened.

Example:

```
if (crsp_f_itm_init(hndl,dbpath,stkappid,'stk1') == CRSP_FAIL) then
   !!--error
   print *,'Error - failed to connect to db:',TRIM(dbpath)
   stop
endif
```

CRSP_F_ITM_IS_MISS_ARRVAL

`crsp_f_itm_is_miss_arrval` checks if the requested element in a data object attached to the item contains a missing value. `is_miss` is set to `.TRUE.` when a missing value is detected. Only items of simple (non-structured) types are accepted, while the item's underlying data-object can be of structured data-type, in which case the structure offset is used to extract the item value.

PROTOTYPE:	integer function crsp_f_itm_is_miss_arrval (itm, ind, is_miss)
ARGUMENTS:	type(CRSP_ITM_T),pointer :: itm – Pointer to the data item integer :: ind – Index of the data array element to check logical :: is_miss – Pointer to the resulting flag value
RETURN VALUES:	CRSP_SUCCESS: If successful, the returned value is initialized and set. CRSP_FAIL: If error in parameters, bad item or element index is out-of-range (ignored in case of CRSP_ROW_T object)
SIDE EFFECTS:	If the requested value contains a missing value, the is_miss is set to .TRUE. Otherwise .FALSE. is assigned.
PRECONDITIONS:	The item has to have a valid bound data-object. Structured items are not allowed. Field items of structures are allowed.

CRSP_F_ITM_LOAD

`crsp_f_itm_load` prepares items described by a full list and loads them to an item table structure in an item handle. It splits the full list into the global section and the list section and uses the function `crsp_f_itm_expand_elem` on each list element in the list section. This will recursively expand the list elements to fill the structure and apply global qualifiers during the process.

PROTOTYPE:	integer function crsp_f_itm_load(hndl, full_list,match_flag)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. character(*) :: full_list – String describing all items to add, used on standard item notation integer :: match_flag – Flag setting the behavior when an item if found but not the keyset. Values are: • CRSP_MATCH_REQUIRED (=0) – if any indicated item and keyset is not found no further items will be added, and CRSP_ NOT_FOUND returned. • CRSP_MATCH_FILL (=1)– a dummy item will be created for any item if the item exists but the keyset does not exist for that item in the current database. • CRSP_MATCH_IGNORE (=2) – items will not be added if the keyset is not found, but the return remains CRSP_SUCCESS.
RETURN VALUES:	CRSP_SUCCESS: If successful, and all indicated items loaded according to match_flag CRSP_FAIL: Error in parameters, bad list, handle not initialized, or reference data not available.
SIDE EFFECTS:	If successful, the CRSP_ITM_GRPis loaded with all indicated items. A CRSP_ITM is allocated for each item/keyset pair not already loaded. Object pointers are not set by this function.
PRECONDITIONS:	The item handle set must be loaded. The item table must be initialized with an available `app_id`. The first set in the set list must agree with the app_id.

CRSP_F_ITM_LOAD_KEY

`crsp_f_itm_load_key` defines the keytype that will be used for subsequent reads.

PROTOTYPE:	integer function crsp_f_itm_load_key(hndl, keytype)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. character(*) :: keytype – Name of the key to initialize. Values are: • gvkey – Compustat company key (default) • gvkeyx – Compustat index key • ccmid – gvkey or gvkeyx • permno – CRSP permno found in any links • permco – CRSP permco found in any links • apermno – CRSP-centric composite records by permno • ppermco – CRSP-centric composite records by permno – primary links only • sic – Compustat company SIC code • ticker – Compustat security ticker symbol • cusip – security CUSIP
RETURN VALUES:	CRSP_SUCCESS: If successful CRSP_FAIL: Error in parameters, handle not initialized, or keytype not found.
SIDE EFFECTS:	If successful, the handle is prepared to handle reads.
PRECONDITIONS:	The item handle must be initialized. Keytype must be known for the app_id.

CRSP_F_ITM_OPEN

`crsp_f_itm_open` registers selected items in a handle by expanding structures and keysets, preparing keys, determining modules needed to access items, opens the needed modules, and binds data in the item lists to the data structure locations. It also builds a master index of all items available in the handle.

PROTOTYPE:	integer function crsp_f_itm_open (hndl)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list.
RETURN VALUES:	CRSP_SUCCESS: If opens successfully and binds the data CRSP_FAIL: If error in parameters, inconsistent handle, error opening databases or binding items.
SIDE EFFECTS:	If successful, the handle is ready for access: • All items in the loaded list will have object pointer set to the data location in the set data structure. • If the handle grp_fill_cd is 'Y', then the item lists are filled to ensure full tables. Filling creates items to ensure that every itm_name and keyset present in a group each combination is present even if not specified. Filling also arranges the lists so if multiple keysets, each is sorted in the same order as the first keyset seen.
PRECONDITIONS:	The item handle must be previously initialized with function crsp_f_itm_init. It generally follows one or more instances of item load function calls

Example:

```
if (crsp_f_itm_open(hndl) == CRSP_FAIL) then
   !!--error
   print *,'Error - failed to open db for access'
   stop
endif
```

CRSP_F_ITM_READ

`crsp_f_itm_read` Loads data from handle based on item keys specified in prior function `crsp_f_itm_key` calls and the `keyflag` argument. Depending on the level of the entity class, the operation may include reading data from the database into structures and/or specifying data already loaded. This allows a direct or positional read based on `keyflag`.

If the value of the access handle property `fiscal_disp_cd` is "C", any fiscal-based time series are shifted to a calendar basis as part of the read operation.

PROTOTYPE:	integer function crsp_f_itm_read (hndl, keyflag, key_status)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. integer :: keyflag – Code determining how the key is interpreted. • CRSP_EXACT – look for a specific value, • CRSP_BACK or CRSP_FORWARD – direct selection when partial matches are allowed, or a positional qualifier to base selection on the position relative to the last key accessed. • CRSP_NEXT (=-99) – read next key in sequence • CRSP_PREV (=-96) – read previous key in sequence • CRSP_SAME (=-98) – read same key, possibly with different information • CRSP_FIRST (=-95) – read first key in the database • CRSP_LAST (=-97) – read last key in the database integer :: key_status – User provided variable to load with the level of the read. It will be loaded with a 0 if the load results in reading new master data. It will be loaded with a number greater than 0 if the load impacts detail or global data, but no master data are affected.
RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_EOF: If positional read reaches the end of the file CRSP_NOT_FOUND: If key not found on exact read. If a detail input key is not provided and no items of that entity class are selected, the return is CRSP_SUCCESS as long as the primary key matches. CRSP_FAIL: If error in parameters, handle not opened, error in read operations.
SIDE EFFECTS:	If successful, the wanted data for the key are loaded into the handle set structure which allows item objects to point to the loaded data. The key found for each level is loaded into the outkey item list. If the handle fiscal_disp_cd is set to calendar-based and items are fiscal-based, shifted calendars are created and time series are converted to calendar basis. The status argument is loaded based on whether the primary key changed. Handle primkey field and readlvl are set. readlvl is set to the rank of the first entity class changed. If the primary key changed, getlvl is set to 0.
PRECONDITIONS:	The item handle must be initialized and opened. The item key must be initialized based on the key type, key element, and the entity class. If not a positional qualifier, the item key inpkey list must be loaded.

Example:

```
sts = crsp_f_itm_read(hndl,CRSP_EXACT, key_sts)
if ( sts == CRSP_FAIL) then
   got_db_error = .true.
   print *,'Error - failed to read db for key:',key
   exit
endif
```

CRSP_F_ITM_SET_KEY

`crsp_f_itm_set_key` loads key information that will be used to load data in a function `crsp_f_itm_read` call. The key is setup during the function `crsp_f_itm_open` based on the active keytype. The value passed to this function is entered into the handle attached to the input key item.

PROTOTYPE:	integer function crsp_f_itm_set_key (hndl, key_itm_name, keyval)
ARGUMENTS:	type(CRSP_ITM_HNDL_T) :: hndl – Access handle containing the needed set structure information and the current item list. character(*) :: key_itm_name – String containing an itm_name of an input key item to be loaded. {integer OR type(CRSP_VARSTRING_T)} :: keyval – Data to be loaded into the key item. Data must agree with the key item's type.
RETURN VALUES:	CRSP_SUCCESS: If data loaded successfully CRSP_FAIL: If error in parameters, handle not open, key item.
SIDE EFFECTS:	If successful, the keyval is copied into the data location for the input key item element in the handle.
PRECONDITIONS:	The item handle must be initialized and opened. The item key array must be initialized based on a keytype with the function crsp_f_itm_open or function crsp_f_itm_init_key functions. The key_itm_name must be a valid item for that keytype, and the keyval data must agree with the type of that item.

Example:

```
if (crsp_f_itm_set_key(hndl,'KYPERMNO',key) == CRSP_FAIL) then
   !!-- error
   print *,'Error - failed to set key:',key
   stop
endif
```

REFERENCE INFORMATION

CRSP FORTRAN 95 API DATA TYPES

All derived types used in the CRSP Fortran 95 API are defined in the module crsp_f_itm_types. They are included in user programs automatically through the use of crsp_f_itm_lib module.

Note: This document lists only selected properties of the defined types that are relevant in the user-scope of item-based access. The full individual definitions of the specific Fortran 95 derived types can be referenced from the respective include source files. These files are already included in the crsp_f_itm_lib module and an explicit include statement is not necessary to use the defined types in your programs. The supplied CRSP Fortran 95 API include files are listed in the following table:

PLATFORM/LOCATION	FILE	DESCRIPTION
Windows 32-bit Windows 64-bit %CRSP_INCLUDE% SunOS sparc SunOS i86pc Linux 32-bit Linux 64-bit $CRSP_INCLUDE	crsp_itm_ccm_types.inc	CRSP CCM/Compustat specific data types
	crsp_itm_stk_types.inc	CRSP Stock specific data types
	crsp_itm_ind_types.inc	CRSP Index specific data types
	crsp_itm_gen_types.inc	CRSP generic data types used in all supported data sets
	crsp_itm_types.inc	Data types used in context of item-access
	crsp_itm_params.inc	Declarations of constant parameters used.

To use the CRSP Fortran 95 API library in your program simply include a 'use' statement for the top-level module crsp_f_itm_lib. All of the required underlying modules will be included automatically. The supplied CRSP Fortran 95 API module files are listed in the following table:

PLATFORM/LOCATION	FILE	DESCRIPTION
Windows 32-bit Windows 64-bit %CRSP_INCLUDE%\mod SunOS sparc SunOS i86pc $CRSP_INCLUDE/mod Linux 32-bit Linux 64-bit $CRSP_INCLUDE/mod $CRSP_INCLUDE/mod_g95	crsp_f_itm_lib.mod	CRSP Fortran 95 itm-API user-level module
	crsp_f_varstring.mod	CRSP implementation of varying-length string (CRSP_VARSTRING_T Fortran 95 derived type)
	crsp_f_itm_utils.mod	Implementations of CRSP itm-API interfaces
	crsp_f_itm_types.mod	Fortran 95 derived types used in context of CRSP Fortran 95 itm-API
	crsp_f_itm_xfer.mod	Internal functions and types for CRSP Fortran 95/C exchange layer

CONTAINER OBJECTS

CRSP container objects are used to uniformly define the storage for various CRSP data types. Generally, the container's data is stored in the associated Fortran 95 array, except in the case of the CRSP_ROW_T container, where the storage is allocated for an Fortran 95 scalar of the specified data type. The associated storage array is externally allocated with 0-based array bounds.

The CRSP time-series object has an associated calendar of the CRSP_CAL_T object type which is aligned with the time-series data array, attributing the date to the values stored in the time-series array.

CRSP calendar data is stored in the CRSP_CAL_T container object, which defines the loaded calendar and also stores the actual calendar data of the defined type. In the context of the CRSP Fortran 95 API, the calendars associated to the time-series items are of day date-type and are accessed with caldt array.

Each container (except CRSP_ROW_T) has a defined availability range, with missing values set beyond the defined range. The missing value is specific to the data type of the stored data and is located at the pre-defined array index position.

Properties of the CRSP container object types are listed in the following tables:

CRSP_TS_T

CRSP time-series container object

NAME	FORTRAN 95 TYPE	DESCRIPTION
objtype	integer	Object type id (CRSP_TS_OTID)
arrtype	integer	Type id of the data stored in the container
subtype	integer	Subtype id of the data stored in the container
maxarr	integer	Maximum bound for the storage array (index is 0-based)
beg	integer	Lower index of the available stored data
end	integer	Upper index of the available stored data
caltype	integer	Calendar type of the associated calendar object
cal	CRSP_CAL_T	Pointer to associated calendar object
miss_val_at = 0		Array index of the missing value for the stored data type

CRSP_ARRAY_T

CRSP array container object

NAME	FORTRAN 95 TYPE	DESCRIPTION
objtype	integer	Object type id (CRSP_ARRAY_OTID)
arrtype	integer	Type id of the data stored in the container
subtype	integer	Subtype id of the data stored in the container
maxarr	integer	Maximum bound for the storage array (index is 0-based)
num	integer	Upper index of the available stored data (index is 0-based)
miss_val_at = maxarr - 1		Array index of the missing value for the stored data type

CRSP_ROW_T

CRSP row container object

NAME	FORTRAN 95 TYPE	DESCRIPTION
objtype	integer	Object type id (CRSP_ROW_OTID)
arrtype	integer	Type id of the data stored in the container
subtype	integer	Subtype id of the data stored in the container

CRSP_CAL_T

CRSP calendar container object

NAME	FORTRAN 95 TYPE	DESCRIPTION
objtype	integer	Object type id (CRSP_CAL_OTID)
calid	integer	Id code of the loaded calendar: • CRSP_CALID_DAILY • CRSP_CALID_MONTHLY • CRSP_CALID_ANNUAL • CRSP_CALID_QUARTERLY • CRSP_CALID_SEMIANNUAL • CRSP_CALID_WEEKLY
maxarr	integer	Maximum bound of the date storage array
gmtoffset	integer	Minutes offset from GMT
timezone	integer	Code for time zone GMT
relflag	integer	If beg and end absolute or relative
beg	integer	Valid range subset begin
end	integer	Valid range subset end
ndays	integer	Number of periods in calendar
name	character(LEN=CRSP_NAMESIZE)	Calendar name
caldt	integer,dimension(:)	Array of the available day dates (yyyymmdd) in the calendar

SUPPORTING TYPES

The CRSP Fortran 95 itm-API supporting types provide additional information about data items and other associated objects in the context of item-based access. An item object is usually associated to a keyset and calendar (in case of time-series items). The details of the keyset (when non-zero) and calendar are given in the `CRSP_ITM_KEYSET_T` and `CRSP_ITM_CAL_T` derived types.

Additionally, the details of the current CRSP data set (such as set name, product name, version, etc.) are provided in the `CRSP_ITM_SET_T` and `CRSP_ROOT_INFO_T` derived types.

The relevant fields of the supporting types are listed in the following tables:

CRSP_ITM_INFO_T

Item detail information

NAME	FORTRAN 95 TYPE	DESCRIPTION
itm_name	character(LEN=CRSP_NAME_LEN)	Item mnemonic name
dflt_keyset	integer	Default keyset
full_name	character(LEN=CRSP_NAMESIZE)	Full non-mnenonic name
itm_type	character(LEN=CRSP_TYPE_LEN)	Type of data item
derv_flg	character(LEN=CRSP_TYPE_LEN)	Item is stored/derived
unit_type	character(LEN=CRSP_CODE_LEN)	Type of units (money, ratio)
unit_mult	double precision	Multiplier to get actual value
cat_type	character(LEN=CRSP_CODE_LEN)	Category (BS, IS, CF, MKT)
src_type	character(LEN=CRSP_CODE_LEN)	Source (filing, market)
freq_type	character(LEN=CRSP_TYPE_LEN)	Reporting frequency type
disp_fmt	character(LEN=CRSP_ITEMNAME_LEN)	Display format specifier
disp_len	integer	Field width for formatted output
disp_precn	integer	Number of decimal places in output
ca_data_size	integer	Internal length
ca_arrtype	integer	Internal structure it belongs
ca_subtype	integer	Internal data sub type
subno_type	integer	Type of variant id
epsflag	integer	Epsilon type/digits for diffs
cepsflag	integer	Epsilon type for character(LEN=diffs)
epsilon	double precision	Absolute epsilon for diffs
desc	character(LEN=CRSP_DESC_LEN)	Default description for field

CRSP_ITM_KEYSET_T

Keyset descriptor

NAME	FORTRAN 95 TYPE	DESCRIPTION
keyset	integer	Keyset number
is_loaded	logical	True when items where requested with this keyset
keyset_info	CRSP_KEYSET_T	Information about the keyset
items_arr	CRSP_ARRAY_T	CRSP array container definition for keyset composing items
items	CRSP_ITM_T,dimension(:)	Array of the items composing the keyset

CRSP_ITM_CAL_T

Calendar descriptor

NAME	FORTRAN 95 TYPE	DESCRIPTION
calid	integer	Calendar ID
keyset	integer	Keyset number keyset = -1 for global base calendars (non-shifted)
is_shifted	loigcal	True if the calendar day dates were shifted based on company's FYE
calcd	character(LEN=CRSP_CALCD_LEN)	Base calendar name
freqcd	character(LEN=CRSP_CHAR_STRSIZE)	Frequency code of the calendar
cal	CRSP_CAL_T, pointer	Pointer to CRSP calendar container object

CRSP_KEYSET_T

Keyset information

NAME	FORTRAN 95 TYPE	DESCRIPTION
keyset	integer	Keyset number
keyset_tag	character(LEN=CRSP_NAME_LEN)	Keyset tag name
desc	character(LEN=CRSP_DESC_LEN)	Keyset description

CRSP_ITM_SET_T

Data set descriptor

NAME	FORTRAN 95 TYPE	DESCRIPTION
set_name	character(LEN=CRSP_NAME_LEN)	Keyset number
path	character(LEN=CRSP_PATHSIZE)	Keyset tag name
root_info	CRSP_ROOT_INFO_T	Database root information

CRSP_ROOT_INFO_T

Database root information

NAME	FORTRAN 95 TYPE	DESCRIPTION
product_name	character(LEN=CRSP_PROD_NAMESIZE)	Database name
product_code	character(LEN=CRSP_CODE_NAMESIZE)	Database code
version	integer	Version number of db
crt_date	character(LEN=CRSP_DATE_SIZE)	Dates are Dow Mon DD HH:MM:SS YYYY
mod_date	character(LEN=CRSP_DATE_SIZE)	Last modification date of db
cut_date	character(LEN=CRSP_DATE_SIZE)	Cut date of db
binary_type	character(LEN=CRSP_CHAR_STRSIZE)	L (IEEE little-endian) or B (big)
code_version	character(LEN=CRSP_OS_NAMESIZE)	CA97 version

CRSP_VARSTRING_T TYPE

The varying-length string Fortran 95 derived type `CRSP_VARSTRING_T` allows for flexible use of expandable character strings and complements the standard fixed-length Fortran 95 character type. The `_VARSTRING_T` string can be constructed from and converted into a regular fixed-length string. The internals of the derived type handle the necessary storage allocation and provide public functions for basic string-related operations.

The `CRSP_VARSTRING_T` derived type is implemented in a separate module, `crsp_f_varstring`. This module is automatically included in the `crsp_f_itm_types` module and is available to programs using the CRSP Fortran 95 API.

FORTRAN 95 MODULE	NAME	DESCRIPTION
crsp_f_varstring	CRSP_VARSTRING_T	varying-length character string

ASSIGNMENT (=)

This interface extends the built-in assignment operator. It allows for construction, assignment, and conversion of a varying-length string, which are handled by internal elemental functions.

INTERFACE:	assignment(=) left = right
ARGUMENTS1:	type(CRSP_VARSTRING_T) :: left type(CRSP_VARSTRING_T) :: right
ARGUMENTS2:	type(CRSP_VARSTRING_T) :: left character(*) :: right
ARGUMENTS3:	character(*) :: left type(CRSP_VARSTRING_T) :: right
RETURN VALUES:	None
SIDE EFFECTS:	Left variable gets assigned the value of the right variable. The left CRSP_VARSTRING_T variable will be re-initialized to accommodate the string value on the right.
PRECONDITIONS:	Available heap memory necessary for dynamic allocation of the internal string storage.
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr1, vstr2 character(LEN=10) :: fixstr = '1235' vstr1 = 'TEST' vstr 2 = fixstr vstr1 = vstr2

LEN

`len` extends the intrinsic `LEN()` function to operate on `CRSP_VARSTRING_T` strings. It returns the current allocated length of the stored string.

INTERFACE:	LEN(vstr)
ARGUMENTS:	type(CRSP_VARSTRING_T) :: vstr
RETURN VALUES:	integer :: len – Length of the stored string. If string is not allocated, returned len=0.
SIDE EFFECTS:	None
PRECONDITIONS:	None
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr integer :: len vstr = 'TEST' len = LEN(vstr)

TRIM

`trim` extends the intrinsic `TRIM()` function to operate on `CRSP_VARSTRING_T` strings. It returns a [right] blank-trimmed stored string as a fixed-length string.

INTERFACE:	TRIM(vstr)
ARGUMENTS:	type(CRSP_VARSTRING_T) :: vstr
RETURN VALUES:	character(*) :: fixstr – Fixed-length string resulting from the stored string with blanks trimmed from the right side.
SIDE EFFECTS:	None
PRECONDITIONS:	None
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr character(LEN=10) :: fixstr fixstr='TEST' !! is blank-padded to allocated length vstr = fixstr print *,'trimmed_str:[',TRIM(vstr),']'

CHAR

Explicitly converts the stored string into fixed-length string. char is often used on string arguments to output statements (such as `PRINT` and `WRITE`).

INTERFACE:	CHAR(vstr)
ARGUMENTS:	type(CRSP_VARSTRING_T) :: vstr
RETURN VALUES:	character(*) :: fixstr – Fixed-length string from the stored string
SIDE EFFECTS:	None
PRECONDITIONS:	None
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr vstr = 'TEST' print *,'vstr:[',CHAR(vstr),']'

CRSP_F_VSTR_FREE

`crsp_f_vstr_free` frees the internally allocated heap storage. It is expected to be called by the user when the `CRSP_VARSTRING_T` variable goes out of its scope of use, so that the allocated memory is released back to the process heap.

PROTOTYPE:	pure subroutine crsp_f_vstr_free (str,stat)
ARGUMENTS:	type(CRSP_VARSTRING_T) :: vstr integer, optional :: stat – Error code to indicate status of the operation: • stat = 0 – SUCCESS • stat = non-zero – FAILED
RETURN VALUES:	None
SIDE EFFECTS:	
PRECONDITIONS:	None
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr integer :: errcd vstr = 'TEST' call crsp_fvstr_free(vstr,stat=errcd) if (errcd /= 0) stop

CRSP_F_VSTR_INIT

`crsp_f_vstr_init` reserves internal storage to hold a string of the specified length and initializes the reserved string with blanks. Normally an explicit call to this function is not required from user programs, as it is called internally by the defined assignment operators.

PROTOTYPE:	pure subroutine crsp_f_vstr_init (str,len, stat)
ARGUMENTS:	type(CRSP_VARSTRING_T) :: vstr – resulting varying-length string integer,intent(in) :: len – Reserved length of the string integer, optional :: stat – Error code to indicate status of the operation: • stat = 0 – SUCCESS • stat = non-zero – FAILED
RETURN VALUES:	None
SIDE EFFECTS:	The error code is set to the stat variable when passed as argument.
PRECONDITIONS:	Available heap memory necessary for dynamic allocation of the internal string storage.
EXAMPLE:	type(CRSP_VARSTRING_T) ::vstr integer :: errcd call crsp_fvstr_init(vstr,10, stat=errcd) if (errcd /= 0) stop

CHAPTER 4: ITEM-BASED SAMPLES

BUILDING AND EXECUTING PROGRAMS

The CRSP API includes a variety of sample programs illustrating item-based access to CRSP databases from Fortran 95 programs. This section describes the sample programs and shows you how to build and execute them on the Windows XP, Sun Solaris, and Linux platforms.

Before creating your own programs, it is a good idea to first build and execute one or more of these sample programs. Besides illustrating CRSP API programming techniques, the sample programs have been tested and debugged by CRSP. If you can successfully run them, then you know your programming environment is correctly configured.

The following table lists the supplied sample programs, organized by database.

FORTRAN ITEM-BASED SAMPLE PROGRAMS

CRSP DATABASE	SAMPLE PROGRAMS	MAKE FILE/ PLATFORM .EXT
CRSP/Compustat Merged Database	ccmitm_fsamp1.f90 –Sequential item-access to CRSP Compustat dataset ccmitm_fsamp2.f90 –Direct item-access to CRSP Compustat dataset by GVKEY list ccmitm_fsamp3.f90 –Direct item-access to CRSP Compustat securities data by GVKEY.IID ccmitm_fsamp4.f90 –Use of CRSP Link for Compustat, item-access by CRSP permno.	Windows / SunOS /Linux Lahey Fortran 95 / Linux G95\ `f95_samp_ccm.mak / .mk/ .mk5 / .mkg5`
CRSP US Stock Database	stkitm_fsamp1.f90 –sequential item-access to CRSP Stock dataset. stkitm_fsamp2.f90 –direct item-access to CRSP Stock dataset by CRSP permno. stkitm_fsamp4.f90 –use of regular and derived data-items for CRSP Stock dataset.	Windows / SunOS /Linux Lahey Fortran 95 / Linux G95 `f95_samp_stkitm.mak / .mk/ .mk5 / .mkg5`
CRSP US Index Database	inditm_fsamp1.f90 –sequential item-access to CRSP Stock & Index Database. inditm_fsamp2.f90 –direct item-access to CRSP Stock Ind dataset by CRSP indno.	Windows / SunOS /Linux Lahey Fortran 95 / Linux G95 `f95_samp_inditm.mak / .mk/ .mk5 / .mkg5`

C ITEM-BASED SAMPLE PROGRAMS

SKTITM_SAMP1.C	Reads CRSP SIZ database sequentially by PERMNO	*stkitm_samp1.c* by default reads all securities in a CRSP Stock database sequentially and creates an output file with one header line per security.
STKITM_SAMP2.C	Reads CRSP Stock database directly using an input file of PERMNOs or file of CUSIPs, as indicated.	*stkitm_samp2.c* reads an input file of PERMNOs or file of CUSIPs and loads data from a CRSP Stock database for each input key. It creates an output file with one header line per record. If an input key is not found in the CRSP Stock database, a message is printed to the screen and no processing is done. This program loads data for selected records. Other sample programs are available that can process a CRSP Stock database sequentially.
STKITM_SAMP4.C	Reads CRSP Stock database directly using an input file of PERMNOs or file of CUSIPs, as indicated. Shows access to derived items.	*stkitm_samp4.c* reads an input file of PERMNOs or file of CUSIPs and loads data from a CRSP Stock database for each input key. It creates an output file with one header line per record. If an input key is not found in the CRSP Stock database, a message is printed to the screen and no processing is done. This program loads data for selected records. Other sample programs are available that can process a CRSP Stock database sequentially.

API ENVIRONMENT

Environment Variables

The CUPL installation process sets a number of environment variables pointing to the locations of modules, include and library files, as well as sample programs. The values of these environment variables are given in the following table, broken down by platform.

FORTRAN

PLATFORM	F95 MODULES (*.MOD)	F95 INCLUDES (*.INC)	F95 LIBRARY	F95 SAMPLES (*.F90)
Windows 32- and 64-bit	%CRSP_INCLUDE%\mod	%CRSP_INCLUDE%	%CRSP_LIB%	%CRSP_SAMPLE%
Sun Solaris - Ultra Sparc - 64-bit	$CRSP_INCLUDE/mod	$CRSP_INCLUDE	$CRSP_LIB	$CRSP_SAMPLE
Red Hat Linux 32- and 64-bit	G95 0.91 complier: $CRSP_INCLUDE/mod_g95	$CRSP_INCLUDE	$CRSP_LIB	$CRSP_SAMPLE

C

PLATFORM	C MODULES (*.MOD)	C INCLUDES (*.INC)	C LIBRARY	C SAMPLES
Windows 32- and 64-bit	%CRSP_INCLUDE%\mod	%CRSP_INCLUDE%	%CRSP_LIB%	%CRSP_SAMPLE%
Sun Solaris - Ultra Sparc - 64-bit	$CRSP_INCLUDE/mod	$CRSP_INCLUDE	$CRSP_LIB	$CRSP_SAMPLE
Red Hat Linux 32- and 64-bit	G95 0.91 complier: $CRSP_INCLUDE/mod_g95	$CRSP_INCLUDE	$CRSP_LIB	$CRSP_SAMPLE

Compiler Options

Platform-specific Fortran 95 compiler options used with Fortran 95 CRSP API are listed in the table below. Refer to the CRSPAccess Release Notes for specific versions of the supported Fortran 95 compilers.

FORTRAN

PLATFORM	FORTRAN 95 COMPILER OPTIONS
Windows 32- and 64-bit	Intel VisualFortran 2011/ParallelStudio XE ifort /Qvec- /I %CRSP_INCLUDE% /I %CRSP_INCLUDE%\mod
Sun Solaris - Ultra Sparc - 64-bit	Sun Fortran-95 8.2: f95 -w -xtarget=generic64 -ext_names=plain -I$CRSP_INCLUDE -M$CRSP_INCLUDE/mod -KPIC
Red Hat Linux 32- and 64-bit	G95 0.91 g95 -w -I$CRSP_INCLUDE -I$CRSP_INCLUDE/mod_g95

C

PLATFORM	C COMPILER OPTIONS
Windows 32- and 64-bit	MS Visual Studio C++ 2008 & 2010
Sun Solaris - Ultra Sparc - 64-bit	Sun C 5.8, part of SunStudio 11
Red Hat Linux 32- and 64-bit	Gcc 4.1.2

LIBRARIES

Platform-specific libraries and options for linking with CRSP API are listed in the table below:

FORTRAN

PLATFORM	FORTRAN 95 COMPILER OPTIONS
Windows 32- and 64-bit	Intel VisualFortran 2011/ParallelStudio XE %CRSP_LIB%\crsp_lib_f95.lib %CRSP_LIB%\crsp_lib.lib
Sun Solaris - Ultra Sparc - 64-bit	Sun Fortran-95 8.2: $CRSP_LIB/crsplib_f95.a $CRSP_LIB/crsplib.a -lm –lnsl
Red Hat Linux 32- and 64-bit	G95 0.91 $CRSP_LIB/crsplib_g95.a $CRSP_LIB/crsplib.a -lm

C

PLATFORM	FORTRAN 95 COMPILER OPTIONS
Windows 32- and 64-bit	Intel VisualFortran 2011/ParallelStudio XE %CRSP_LIB%\crsp_lib_c.lib %CRSP_LIB%\crsp_lib.lib
Sun Solaris - Ultra Sparc - 64-bit	Sun Fortran-95 8.2: $CRSP_LIB/crsplib_c.a $CRSP_LIB/crsplib.a -lm –lnsl
Red Hat Linux 32- and 64-bit	G95 0.91 $CRSP_LIB/crsplib_g95.a $CRSP_LIB/crsplib.a -lm

SUN SOLARIS

CRSP currently supports Sun Sparc Solaris 2.9/5.9 with the Forte Developer 7.0, Fortran 95 7.0, and Sun x86 Solaris 2.9/5.9.

Fortran was compiled and tested using the above compiler. Fortran library functions interface with C functions in the CRSP object library. Ordinary sample Fortran usage links to the object library, but does not require compiling C programs.

CRSP access depends on environment variables set during installation. Environment variables can be used on Unix with the name preceded by the $ symbol. All file names and environment variable names are case sensitive on Unix systems. The env command can be used in a terminal window to find available environment variables.

Important CRSP files or directories can be found with the following names:

`$CRSP_BIN`	Directory containing executable programs and shell scripts files. This directory is in the PATH so programs can be run from any directory. Executable versions of the sample programs can be found in this directory.
`$CRSP_LIB`	Directory containing CRSP object library and internal files.
`$CRSP_LIB/crsplib.a`	CRSP C object library.
`$CRSP_LIB/crsplib_f95.a`	CRSP F95 object library.
`$CRSP_INCLUDE`	Directory containing CRSP FORTRAN header files referred to by INCLUDE statements.
`$CRSP_SAMPLE`	Directory containing CRSP sample programs.
`$CRSP_MSTK`	Directory containing monthly CRSP stock and index databases.
`$CRSP_DSTK`	Directory containing daily CRSP stock and index databases.
`$CRSP_CST`	Directory containing CRSP Link and COMPUSTAT database.
`$CRSP_WORK`	Directory created to hold user-generated files

SUN FORTRAN 95 8.2

Command line

```
> cp $CRSP_SAMPLE/stkitm_fsamp1.f90 .

> chmod 660 stkitm_fsamp1.f90
```

In Sun Sparc Solaris 2.9/5.9:

```
> f95 -w -xarch=v9 -ext_names=plain -I$CRSP_INCLUDE -KPIC -o stkitm_fsamp1 stkitm_fsamp1.f90 $CRSP_
LIB/crsplib_f95.a  $CRSP_LIB/crsplib.a
```

In Sun x86 Solaris 2.9/5.9:

```
> f95 -w -xtarget=generic64 -ext_names=plain -I$CRSP_INCLUDE -KPIC -o stkitm_fsamp1 stkitm_fsamp1.
f90  $CRSP_LIB/crsplib_f95.a $CRSP_LIB/crsplib.a
```

To run the program:

```
> ./stkitm_fsamp1
```

Sample programs can also be compiled and linked using the make utility. The sample program directory `$CRSP_SAMPLE` contains sample make description files for Sun Solaris in `f95_samp.mak`. To use make, copy the relevant description file to your program directory, edit it to support the program(s) of interest and create local executables.

using a Make file:

To compile a specific sample program:

```
> make -f f95_samp.mk stkitm_fsamp1
```

To compile all sample programs:

```
> make -f f95_samp.mk
```

To run the program:

```
> ./stkitm_fsamp1
```

LINUX

CRSP currently supports Linux, Red Hat 7.2 (32-bit) and RHEL5 (64-bit) on Intel x86. FORTRAN was compiled and tested using the Lahey Fortran 95 Version 6.2 (32-bit) and the g95 Version 0.091 32- and 64-bit compilers. Fortran library functions interface with C functions in the CRSP object library. Ordinary sample Fortran usage links to the object library, but does not require compiling C programs.

CRSP access depends on environment variables set during installation. Environment variables can be used on Linux with the name preceded by the $ symbol. All file names and environment variable names are case sensitive on Linux systems. The env command can be used in a terminal window to find available environment variables.

Important CRSP files or directories can be found with the following names:

$CRSP_BIN	Directory containing executable programs and shell scripts files. This directory is in the PATH so programs can be run from any directory. Executable versions of the sample programs can be found in this directory.
$CRSP_LIB	Directory containing CRSP object library and internal files.
$CRSP_LIB/crsplib.a	CRSP object library.
$CRSP_LIB/crsplib_f95.a	CRSP F95 object library.
$CRSP_INCLUDE	Directory containing CRSP Fortran header files referred to by INCLUDE statements.
$CRSP_SAMPLE	Directory containing CRSP sample programs.
$CRSP_MSTK	Directory containing monthly CRSP stock and index databases.
$CRSP_DSTK	Directory containing daily CRSP stock and index databases.
$CRSP_CST	Directory containing CRSP Link and COMPUSTAT database.
$CRSP_WORK	Directory created to hold user-generated files

Following is an example of modifying and running a sample FORTRAN program:

G95 Ver. 0.91 32- and 64-bit

Command line:

```
> cp $CRSP_SAMPLE/stkitm_fsamp1.f90 .

> chmod 660 stkitm_fsamp1.f90

> g95 -o stkitm_fsamp1 -w stkitm_fsamp1.f90 -I$CRSP_INCLUDE $CRSP_LIB/crsplib.a $CRSP_LIB/crsplib_
f95.a `find /usr/local -name libf95.a 2>&1 | grep libf95\.a -lm
```

To run the program:

```
> ./stkitm_fsamp1
```

Using a Make File:

The sample program directory `$CRSP_SAMPLE` contains sample make description files for Linux in `f95_samp.mkg5` for the g95 compiler. To use the make file, copy the relevant description file to your program directory, and edit it to support the program(s) of interest and create local executables.

To compile specific sample program:

```
> make -f f95_samp.mkg5 stkitm_fsamp1
```

To compile all sample programs:

```
> make -f f95_samp.mkg5
```

To run the program:

```
> ./stkitm_fsamp1
```

USING THE CRSP FORTRAN 95 API

When you have ascertained that you can successfully compile and execute the provided sample programs, you are ready to begin creating your own programs. This section illustrates the general flow of CRSP API programs and discusses the data objects you will use when accessing CRSP Databases from Fortran 95.

SAMPLE API PROGRAM FLOW

CRSP Fortran 95 API client applications are structured according to the following steps. The code snippets shown below are excerpted from the `stkitm_fsamp4.f90` sample program included with the API.

1. Include a 'use' statement for the `crsp_f_itm_lib` module. This step makes API functions and constants available from within your program.

```
use crsp_f_itm_lib
```

2. Declare the item-access handle:

```
type(CRSP_ITM_HNDL_T) :: hndl
```

3. Declare pointers to item objects to be used in your program:

```
type(CRSP_ITM_T),pointer :: stkhdr_itm => null(),  &

                            prc_itm => null(),          &

                            adjprc_itm => null(),       &

                            adjshr_itm => null(),       &

                            adjvol_itm => null()
```

4. Connect to a CRSP database for item-access. Specify database root and the available item-set app_id. :

```
if (crsp_f_itm_init(hndl,dbpath,stkappid,'stk1') == CRSP_FAIL) then

        !!--error
        print *,'Error - failed to connect to db:',TRIM(dbpath)
        stop
endif
```

5. Select the wanted items to be loaded into active item set. Multiple calls to crsp_f_itm_load are allowed and have expanding effect on the item selection, while without the selected item duplication.

```
sts=crsp_f_itm_load(hndl,'STKHDR_ALL',CRSP_MATCH_IGNORE)

if ( sts == CRSP_FAIL .or. sts == CRSP_NOT_FOUND) then

        !!--error
        print *,'Error - failed to load the requested data items (DSTK:1)'
        stop
endif
```

```
sts=crsp_f_itm_load(hndl,'DSTK_TS;ADJPRC;ADJSHR;ADJVOL',CRSP_MATCH_IGNORE)

if ( sts == CRSP_FAIL .or. sts == CRSP_NOT_FOUND) then

        !!--error
        print *,'Error - failed to load the requested data items (DSTK:2)'
        stop
endif
```

6. Configure the item-access if needed by setting any of the access configuration codes in the access handle, eg:

```
hndl%fiscal_disp_cd = 'F'
```

See CRSP Fortran 95 API Data Objects on "CRSP Fortran 95 API Data Objects" on page 31 for the details of access handle properties.

7. Open the item-access to the selected CRSP dataset:

```
if (crsp_f_itm_open(hndl) == CRSP_FAIL) then

        !!--error
        print *,'Error - failed to open db for access'
        stop
endif
```

8. Attach the user-declared item pointers to the loaded items. Specify the item name and keyset:

```
if (crsp_f_itm_find(hndl,'HEADER',0,stkhdr_itm) == CRSP_FAIL          &
        .or. crsp_f_itm_find(hndl,'PRC',0,prc_itm) == CRSP_FAIL       &
        .or. crsp_f_itm_find(hndl,'ADJPRC',0,adjprc_itm) == CRSP_FAIL &
        .or. crsp_f_itm_find(hndl,'ADJSHR',0,adjshr_itm) == CRSP_FAIL &
        .or. crsp_f_itm_find(hndl,'ADJVOL',0,adjvol_itm) == CRSP_FAIL &
        .or.  .not. associated(stkhdr_itm)  &
        .or.  .not. associated(prc_itm)     &
        .or.  .not. associated(adjprc_itm)  &
        .or.  .not. associated(adjshr_itm)  &
        .or.  .not. associated(adjvol_itm)) then
        !!--error
        print *,'Error - invalid item/keyset specified'
        stop
endif
```

Note: Only previously loaded items can be found. If an item is not being found, first make sure the requested item has been loaded in the requested keyset explicitly or implicitly through a collective item group.

9. Load the primary access key if different from a defined default.

```
if (crsp_f_itm_load_key(hndl,'permno') == CRSP_FAIL) then

        !!--error
        print *,'Error - failed to load index for key:','permno'
```

```
        stop
endif
```

10. In case of direct access, set the corresponding key item to the target value.

```
if (crsp_f_itm_set_key(hndl,'KYPERMNO',key) == CRSP_FAIL) then

        !!-- error
        print *,'Error - failed to set key:',key
        stop
endif
```

11. Read the database. Specify the key matching mode when exact key value is not found. In case of sequential access set key flag to CRSP_NEXT. For composite primary key the non-zero key_status signals access on detail key:

```
sts = crsp_f_itm_read(hndl,CRSP_EXACT, key_sts)

if ( sts == CRSP_FAIL) then

        got_db_error = .true.
        print *,'Error - failed to read db for key:',key
        exit
endif
```

12. On successful read-status the data is loaded and item data containers are ready for access. The item data can be accessed through the attached item pointers. This step is where your application-specific logic comes into play.

```
do i=prc_itm%obj%ts%beg, prc_itm%obj%ts%end

   write(ofunit,601)                          &

        stkhdr_itm%arr%header_val%permno,   &
        stkhdr_itm%arr%header_val%hcomnam, &
        prc_itm%obj%ts%cal%caldt(i),        &
        prc_itm%arr%flt_arr(i),            &
        adjprc_itm%arr%flt_arr(i),         &
        adjshr_itm%arr%int_arr(i),          &
        adjvol_itm%arr%dbl_arr(i)


601       format(I6,1X,A32,1X,I8,1X,F12.5,1X,F12.5,1X,I9,1X,F13.1)

enddo
```

13. Close the item-access and disconnect from the selected CRSP dataset. This also releases the internally allocated storage for this item-handle instance and invalidates any user-declared item pointers attached to the handle.

```
if (crsp_f_itm_close(hndl) == CRSP_FAIL) then

        !!--error
        print *, 'Error-- failed to close db:',TRIM(dbpath)
        stop
endif
```

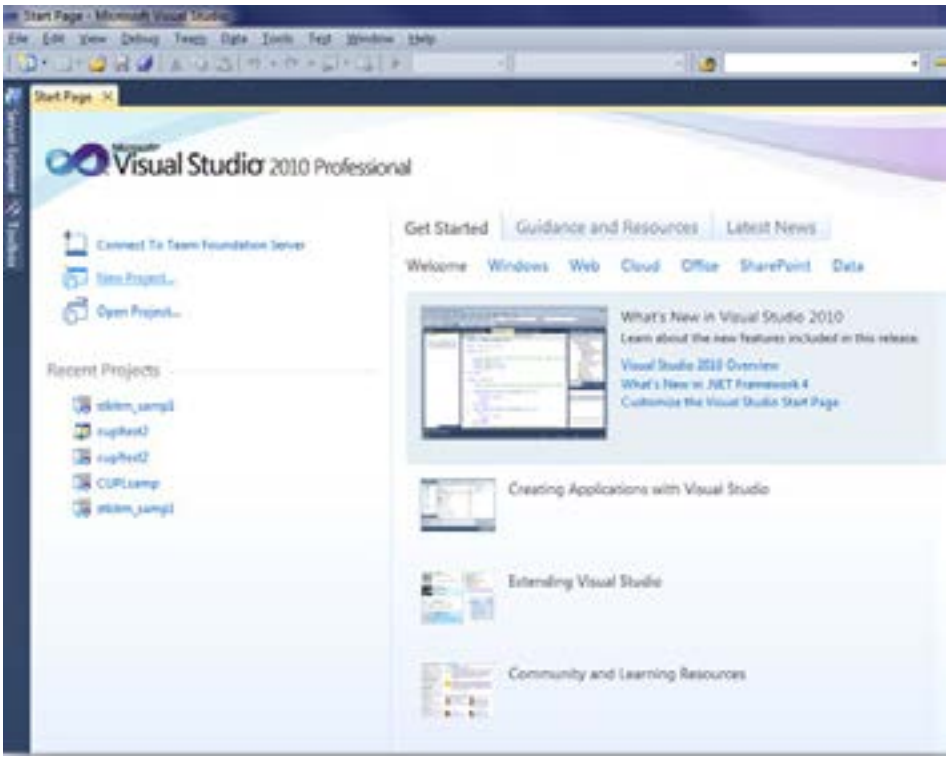
For more detailed examples of item-access to supported CRSP database products, you are encouraged to refer to the supplied set of sample programs.

MICROSOFT WINDOWS

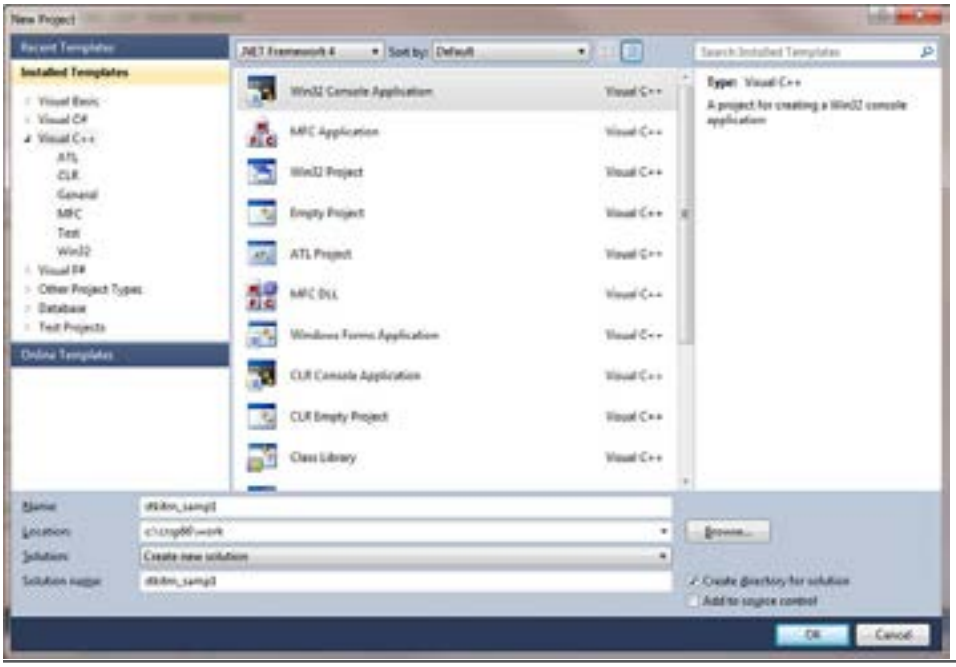
VISUAL STUDIO 2010 - C COMPILER INSTRUCTIONS

CRSP supports compiling C programs in Windows 32-bit and 64-bit environments. The following example compiles a sample C program provided with the CUPL tools using Microsoft Visual Studio 2010. Use 64-bit options in Visual Studio with a 64-bit install of CUPL, 32-bit options with a 32-bit install.

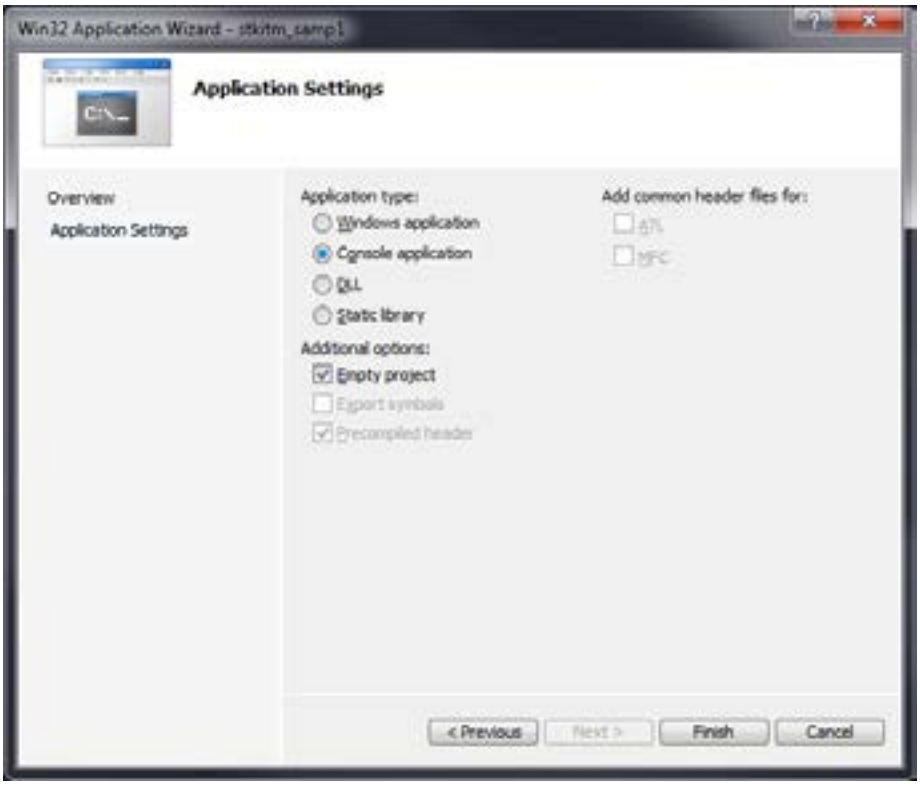
STEP 1: To begin, open Visual Studio 2010. Click on New Project.



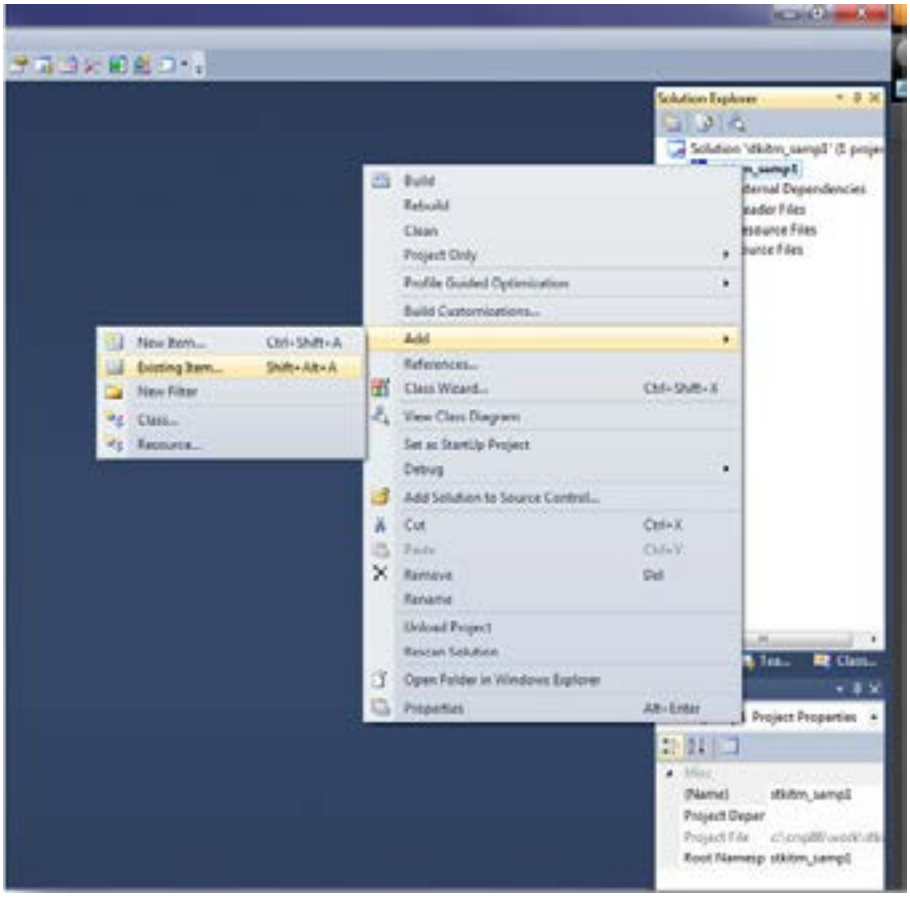
STEP 2: Select Visual C++ and highlight Win32 Console Application. Give the project a name, specify a location, and provide a solution name. Click OK.



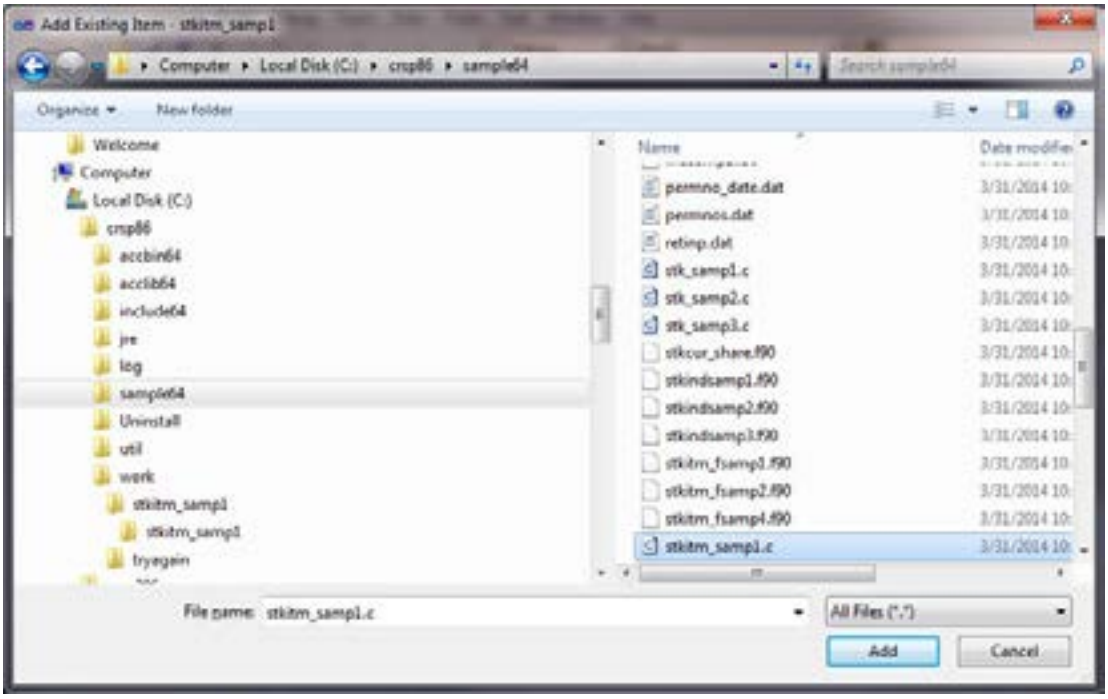
STEP 3: On the Application Settings screen, click on Console application, and check Empty project, then click on Finish.



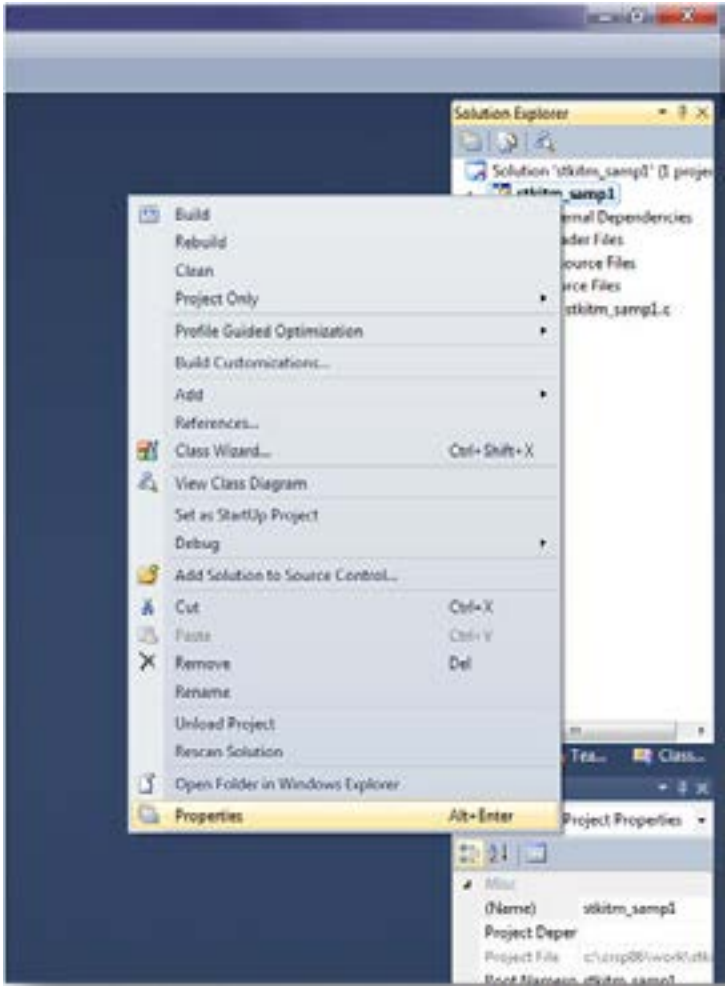
STEP 4: You are ready to add information to the project that you are building. To do so, right click on the project, in this example, `stkitm_samp1` (in bold). On the pop-up screen, select Add > Existing Item.



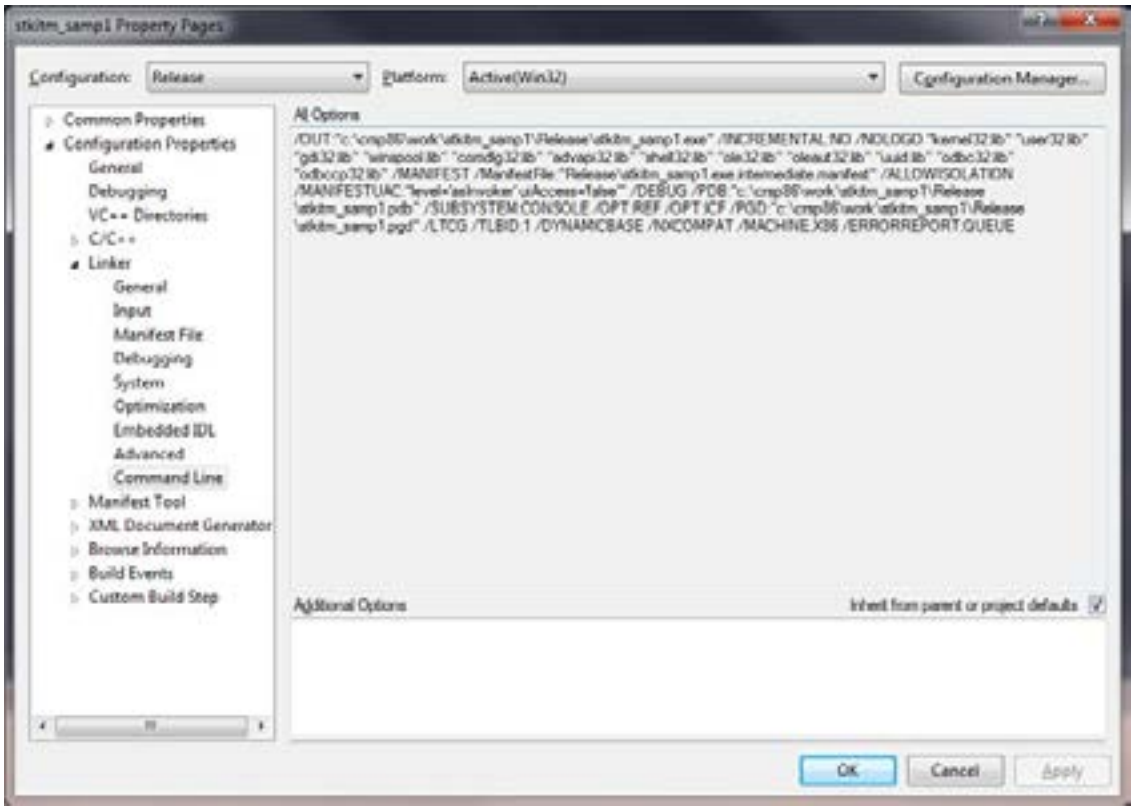
STEP 5: Browse for the program that you would like to add. In the CUPL Version 3.86 – 64-bit tools, sample programs are located in the Sample64 folder. Highlight `stkitm_samp1.c` program and Add.



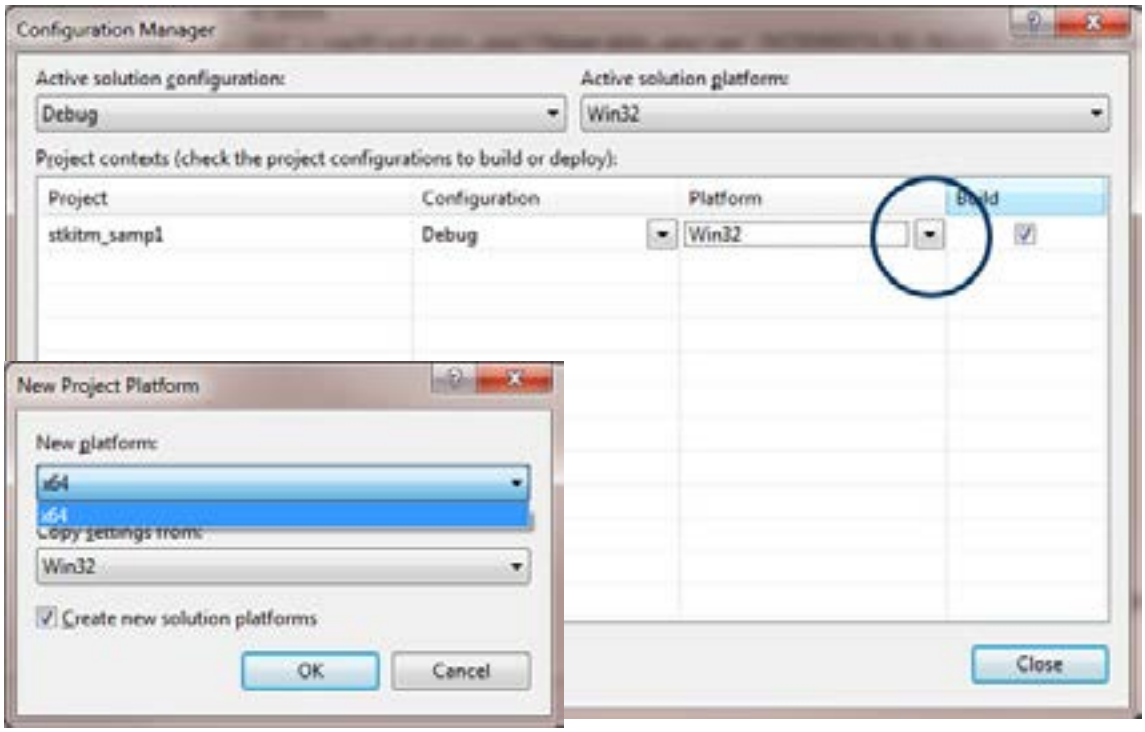
STEP 6: The program will display in the Source folder of the project. Right click on the `stkitm_samp1` project again, and at the very bottom of the window, select Properties.



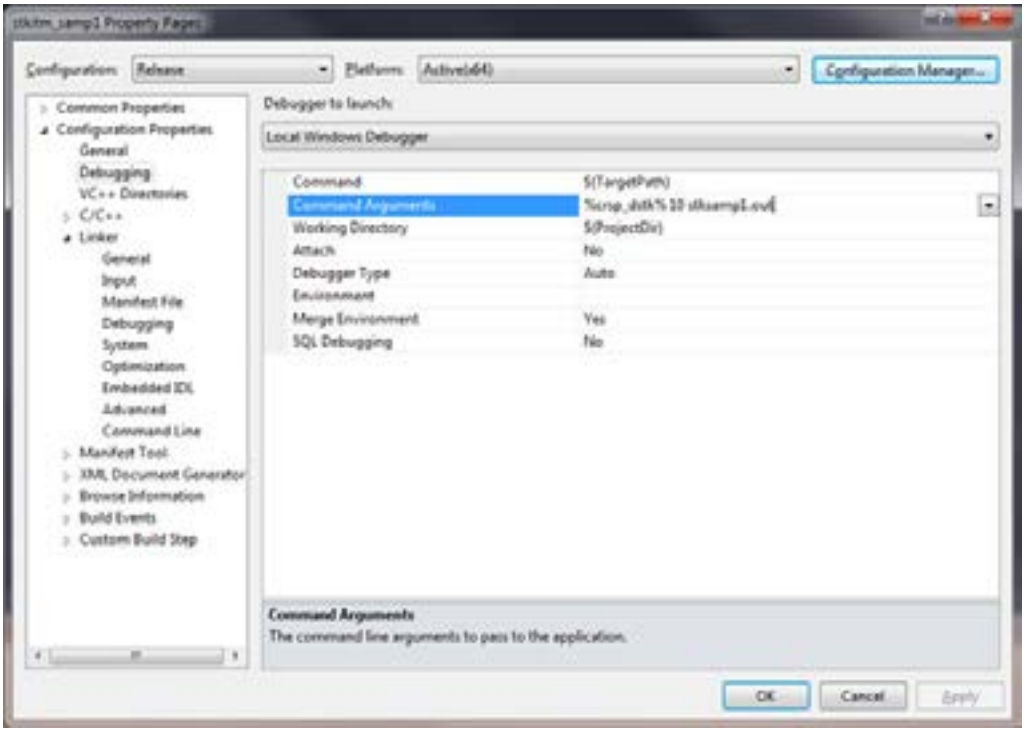
STEP 7: At this point, there are several actions to take and there is no specific order necessary. First, in the Configuration options in the upper left corner of the screen, click on the dropdown and select Release.



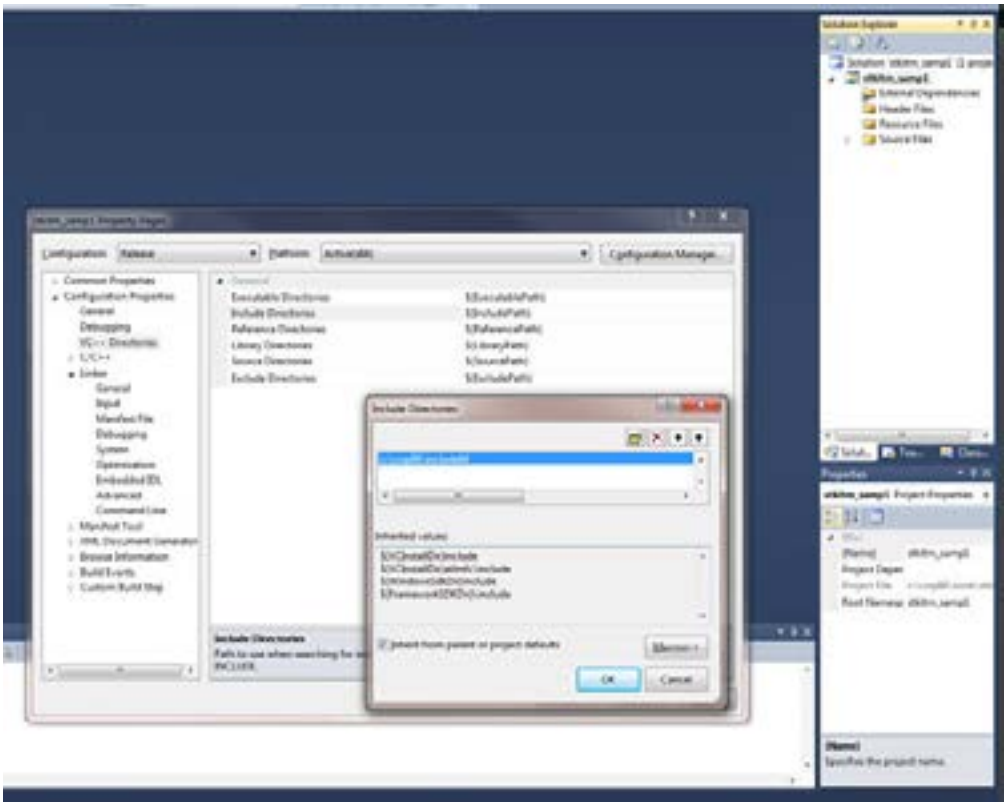
STEP 8: On the top right corner of the same screen (see above), click on the Configuration Manager. From the Active solution platform dropdown, select x64 and click OK. If x64 doesn't exist as an option, from this same dropdown click on New and add x64 as an option, click OK, and then Close.



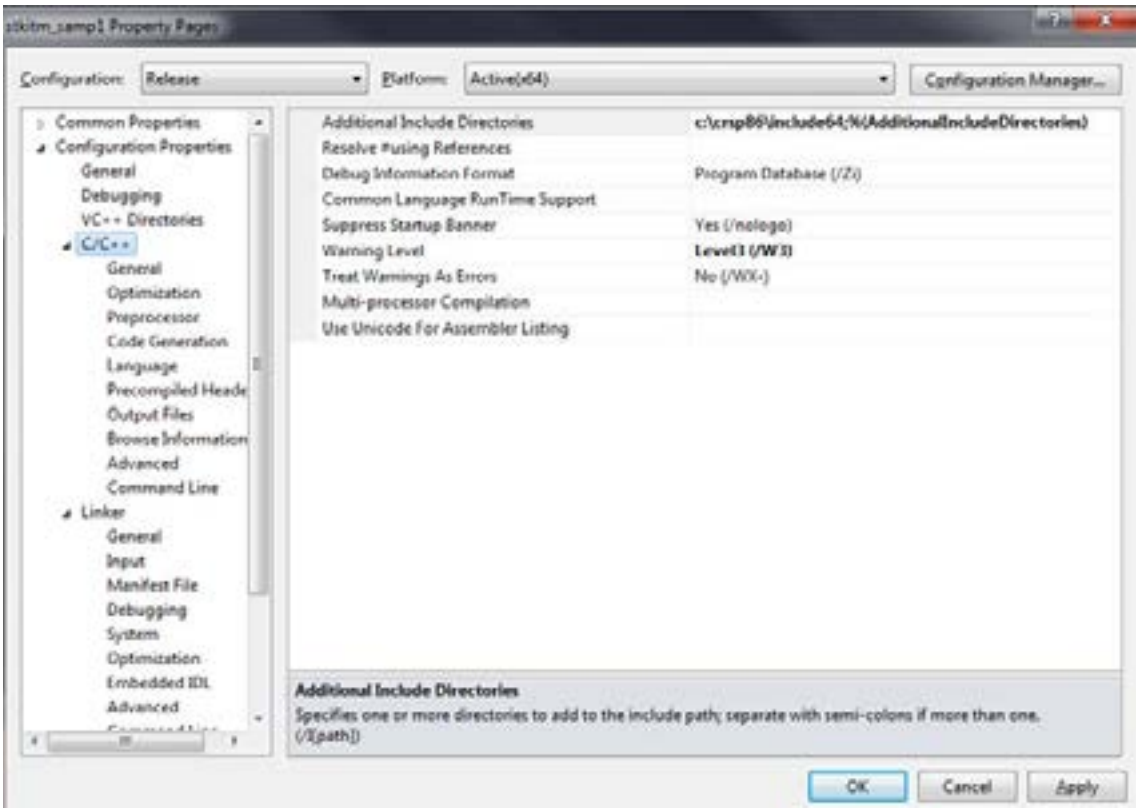
STEP 9: Back to the Property Pages, under Configuration Properties, click on Debugging. In the Command Arguments line, define the database that you will use, and enter a name for the output file. In this example, `%crsp_dstk%` is using environment variables that are pointing to the CRSP daily stock database. "10" is the daily stock setid. `stksamp1.out` is the file that will be generated once the project is built and run.



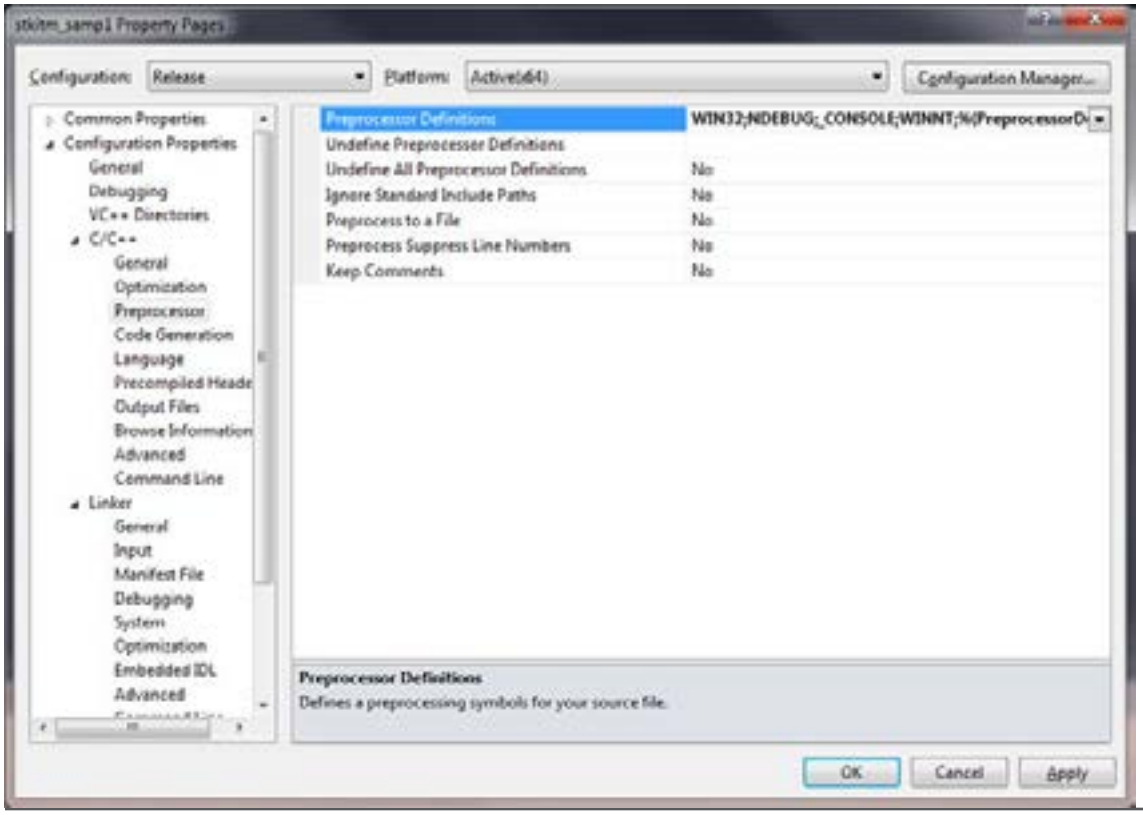
STEP 10: Still under the Configuration Properties, click on VC++ Directories. Highlight the Include Directories row and click on the dropdown. Click on Edit and add the location of the Include folder in the CUPL tools. In this example, `c:\crsp86\include64`. Click OK.



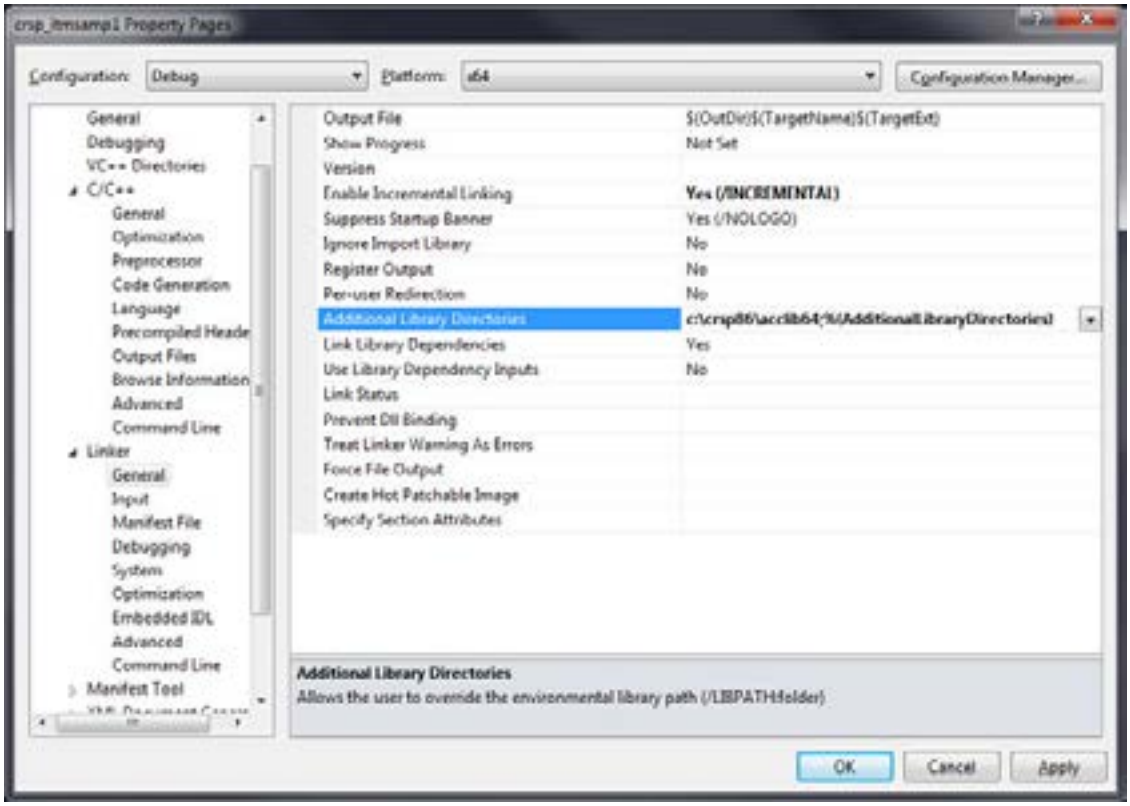
STEP 11: Next, expand the C/C++ directory select the General tab. Highlight the Additional Include Directories and click on the dropdown and Edit. Enter the path for the CRSP include files. In the example, the path is `c:\crsp86\include64`. Click OK to close the window.



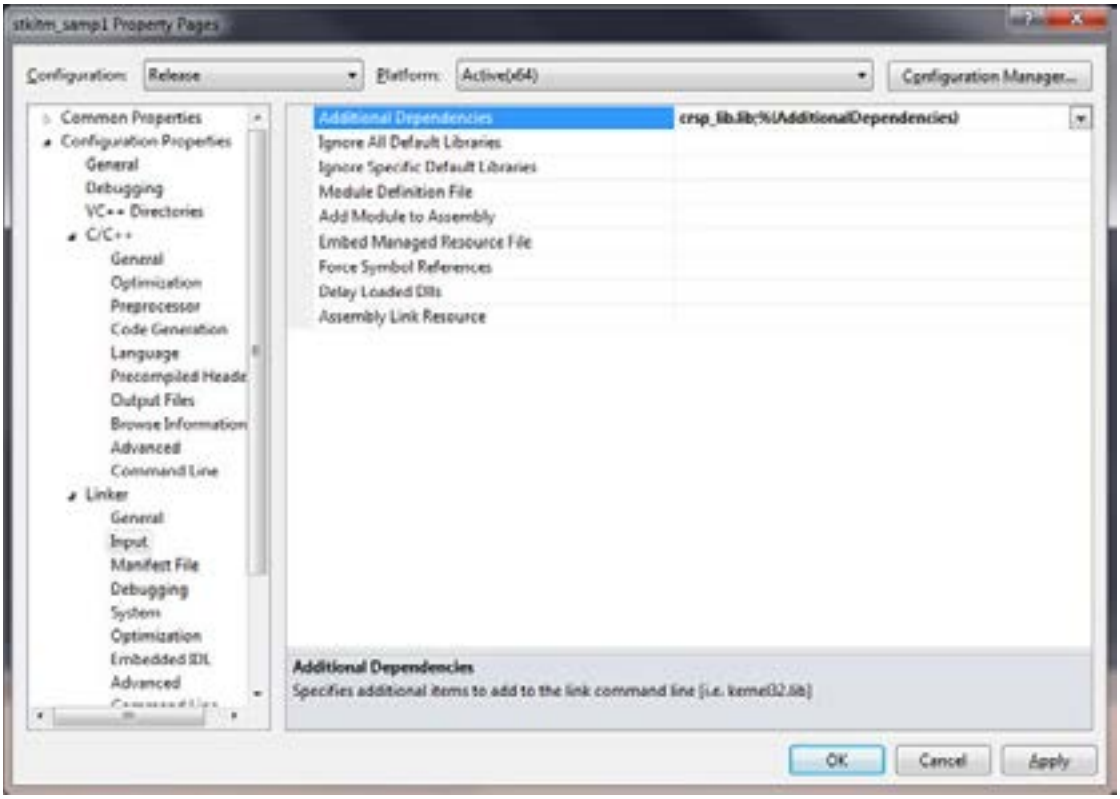
STEP 12: Still in the the C/C++ folder, select Preprocessor. Highlight Preprocessor Definitions, click on the dropdown and Edit. Enter `WINNT` and click OK to close the window.



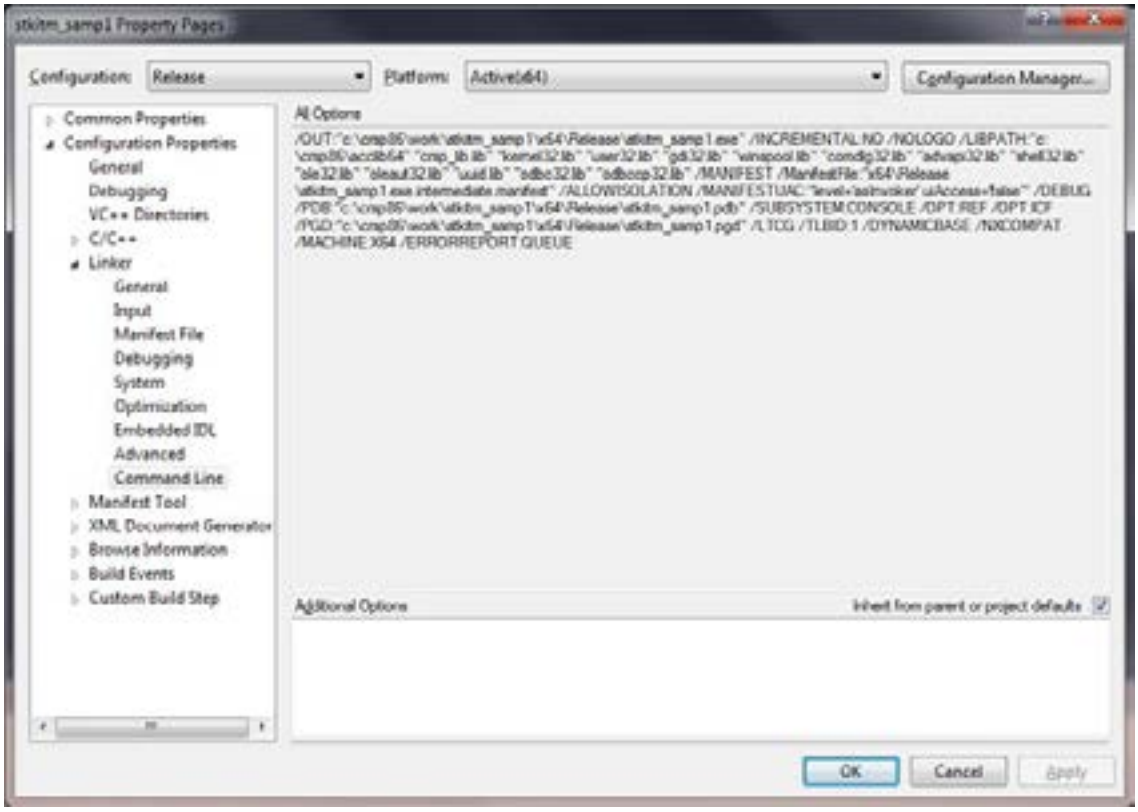
STEP 13: Next in Configuration Properties, expand the Linker folder and select General. Highlight the Additional Library Directories row and click on the dropdown. Enter the path for the CRSP libraries. In this example, it is `c:\crsp86\acclib64`.



STEP 14: Stay in the Linker folder and select Input. Click on the Additional Dependencies row, click on the dropdown and Edit. Enter the CRSP library file name, `crsp_lib.lib` and click OK to close the window.



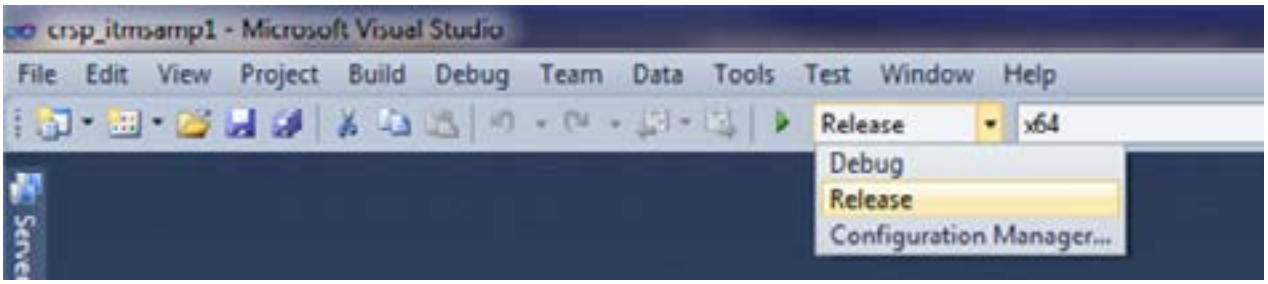
STEP 15: Finally, within Linker, select Command Line and click Apply in the lower right corner of the screen. Click OK to close the Properties Pages.



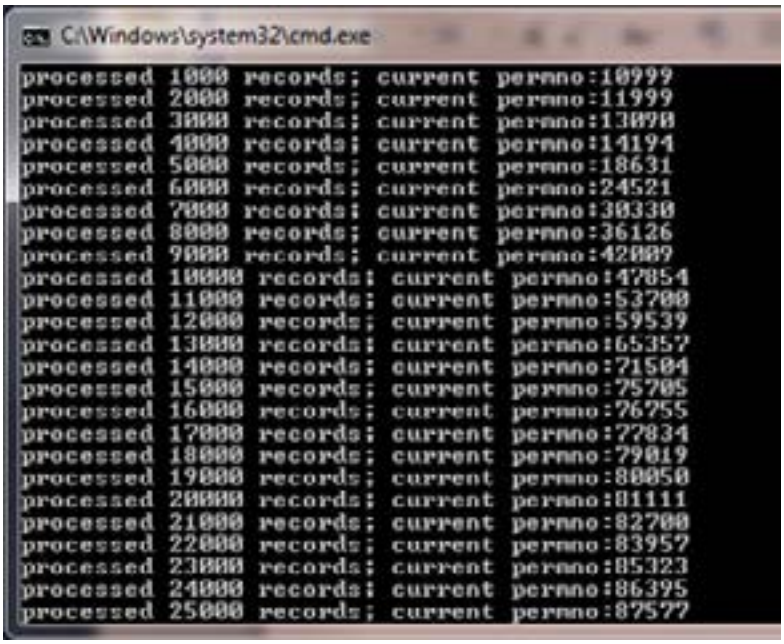
STEP 16: At this point, all entries should have been made in order to build the solution. From the menu bar, select Build > Build `stkitm_samp1`. Assuming that the build runs successfully to completion, you ill see the following message once the build is complete:

Build: 1 succeeded, 0 failed, 0 skipped.

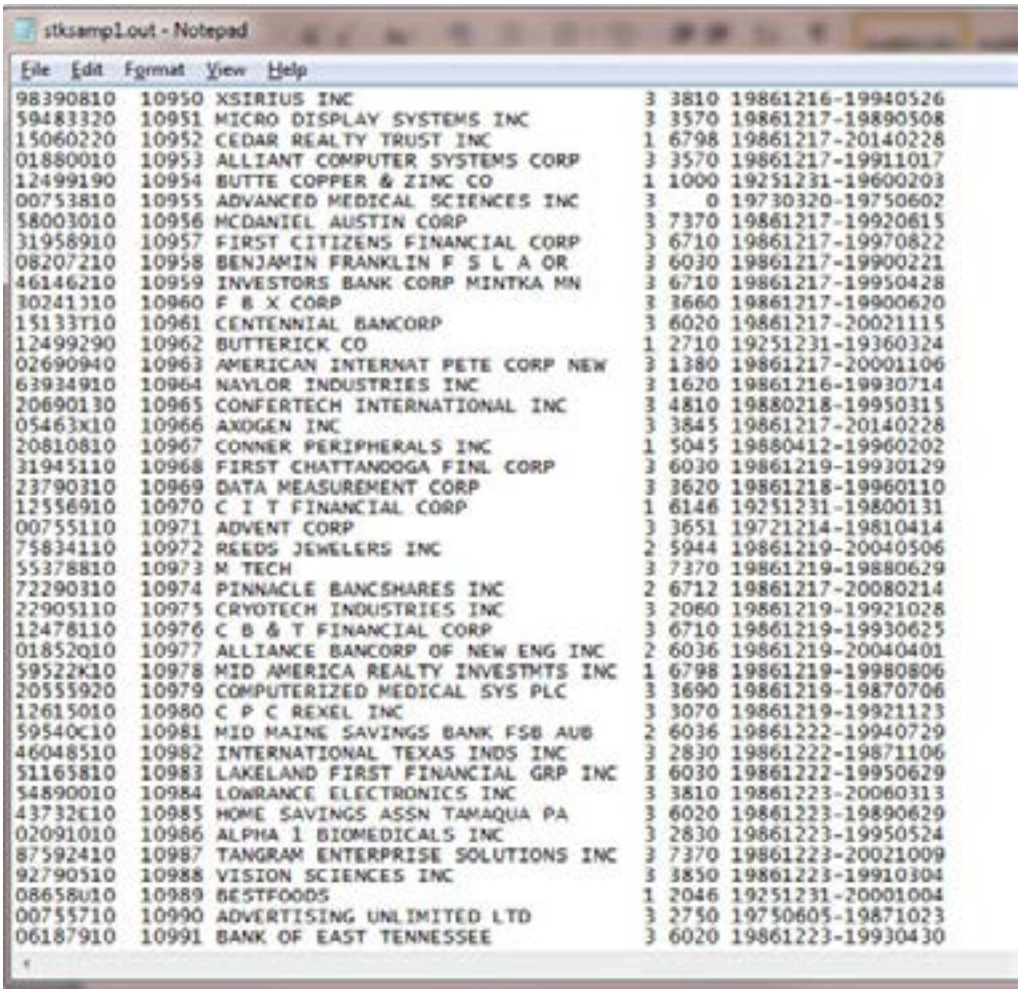
Prior to running your program, check the Visual Studio Menu bar to confirm that the Solution Configurations set the mode to Release. (Note: At CRSP, if not set to Release mode, we encountered an error message stating that MSVCR100.dll is not found)



STEP 17: Once you have built your program successfully, you can now run it to generate output. From the Menu Bar, click on Debug > Start Without Debugging. The program will begin running and for this example, will work sequentially through the universe of CRSP PERMNOs.



Your output will be located in `c:\CRSP86\work`, or as specified in your project.

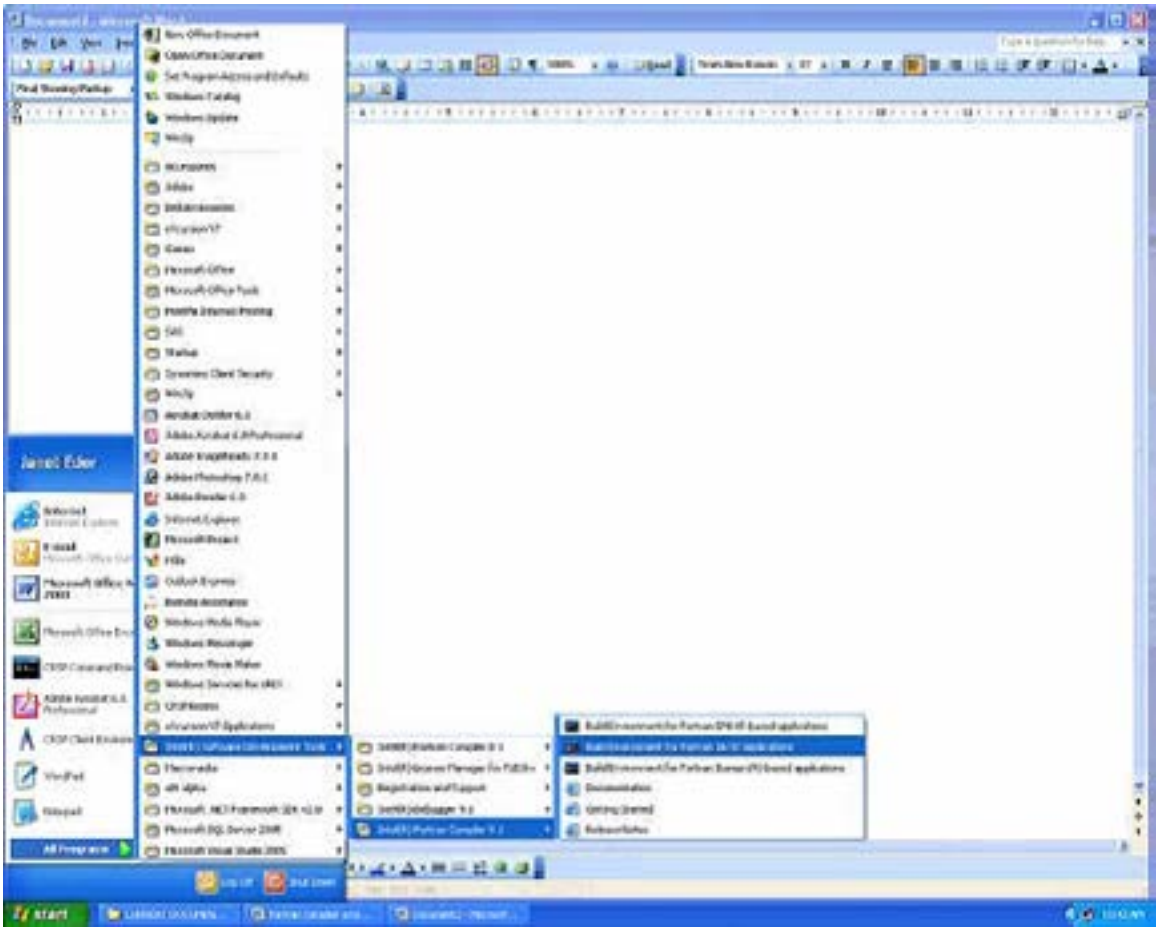


WINDOWS COMMAND PROMPT

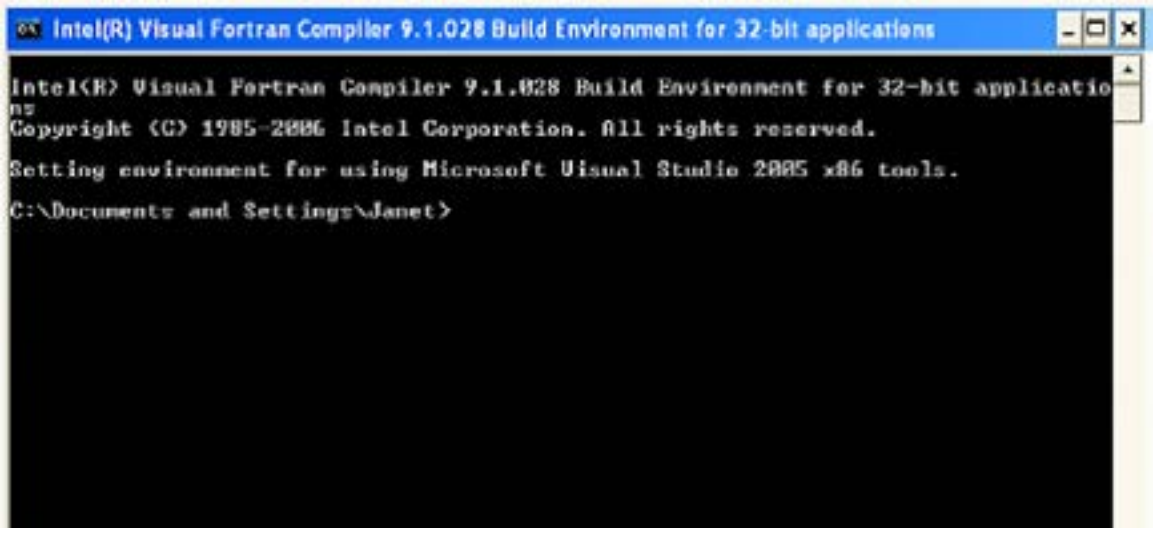
The sample programs can also be compiled and run from a command prompt window. In order to do so, the environment must be set for Intel Fortran to run.

To set the Windows 32-bit environment to Intel(R) Fortran, click on Start > All Programs > Intel(R) Software Development Tools > Intel(R) Fortran Compiler 9.1 > Build Environment for Fortran IA-32 applications.

To set the Windows 64-bit environment to Intel(R) Fortran, click on Start > All Programs > Intel Parallel Studio XE 2011 > Command Prompt > Parallel Studio XE with Intel Compiler > Intel 64 Visual Studio 2008 mode.



A DOS window will open ready for you to run your Fortran 95 programs.



Compiling from the Command Prompt

To compile a sample program from the command prompt using the Intel Fortran Compiler, copy it to your program directory and invoke the invoke the ifort command as shown below.

```
>  copy %crsp_sample%\stkitm_fsamp1.f90

>  ifort /I%crsp_include% stkitm_fsamp1.f90 %crsp_lib%\crsp_lib.lib %crsp.lib%\crsp_lib_F95.lib
```

To run the program:

```
>  .\stkitm_fsamp1
```

Using a Make File

Sample programs can also be compiled and linked at the command prompt using the nmake utility. A sample description file, f95_samp.mak, exists in the %crsp_sample% directory. To use the sample description file with your own program, copy it to your program directory and modify it to include your program instead of the sample.

```
>  copy  %crsp_sample%\f95_samp.mak  .
```

To compile a specific sample program:

```
>  nmake /f f95_samp.mak stkitm_fsamp1.exe
```

To compile all sample programs:

```
>  nmake /f f95_samp.mak
```

To run the program:

```
>  .\stkitm_fsamp1
```

CHAPTER 5: LEGACY SET ACCESS IN C

CRSPACCESS C DATA STRUCTURES

C Programming allows complete support for CRSP databases, including random access on PERMNO, CUSIP and other header variables, and full support of all data items. There are sample programs, header files, and an object library available.

DATA ORGANIZATION FOR C PROGRAMMING

The basic levels of a CRSPAccess database are the database, set type, set id, module, object, and array. They are defined as follows:

- Database (`CRSPDB`) is the directory containing the database files. A `CRSPDB` is identified by the database path.
- Set Type is a predefined type of financial data. Each set type has its own defined set of data structureH6s, spe- cialized access functions, and keys. CRSPAccess stock databases support stock (`STK`) and index (`IND`) set types. A `CRSPDB` can include more than one set type.
- Set Identifier (`SETID`) is a defined subset of a set type. `SETID`s of the same set type use the same access func- tions, structures, and keys, but have different characteristics within those structures. For example, daily stock sets use the same data structure as monthly stock sets, but time series are associated with different calendars. Multiple `SETID`s of the same set type can be present in one `CRSPDB`.
- Modules are the groupings of data found in the data files in a `CRSPDB`. Multiple data items can be present in a module. Data are retrieved at a module level, and access functions retrieve data items for keys based on selected modules. Modules correspond to the physical data files.
- Objects are the fundamental data types defined for each set type. There are three fundamental object types: time series (`CRSP_TIMESERIES`), event arrays (`CRSP_ARRAY`), and headers (`CRSP_ROW`). Objects contain header information such as counts, ranges, or associated calendars (`CRSP_CAL`) plus arrays of data for zero or more observations. Some set types allow arrays of objects of one type. In this case, the number of available objects is determined by the `SETID`, and each of the objects in the list has independent counts, ranges, or associ- ated calendars.
- Arrays are attached to each object. The array contains the set of observations and is the basic level of program- ming access. An observation can be a simple data type such as an integer for an array of volumes, or a complex structure such as for a name history. When there is an array of objects, there is a corresponding array of arrays with the data.

DATA OBJECTS

There are four basic types of information stored in CRSP databases. Each is associated with a CRSP object structure.

- **Header Information.** These are identifiers with no implied time component.
- **Event Arrays.** Arrays can represent status changes, random events, or observations. The time of the event and relevant information is stored for each observation. There is a count of the number of observations for each type of event data.
- **Time Series Arrays.** An observation is available for each period in an associated calendar. A beginning and ending point of valid data are available for each type of time series data. Data are stored for each period in the range — missing values are stored as placeholders if information is not available for a period.
- **Calendar Arrays.** Each time series is tied to an array of relevant time periods. This calendar is used in conjunction with the time series arrays to attach times to the observations.

An observation can be a simple value or contain multiple components such as codes and amounts. Time series, except Portfolios, are based on calendars which share the frequency of the database. In a monthly database, the time series are based on a month-end trading date calendar. In a daily database, the time series are based on a daily trading date calendar excluding market holidays. Portfolio calendars are dependent on the rebalancing methodology of the specific portfolio type. All calendars are attached automatically to each wanted time series object when the database is opened.

There are four base CRSPAccess C structures called objects used in `CRSPDB`s. The following table contains each of the objects in all caps, followed by the components, lower case and indented, that each object type contains. All data items are defined in terms of the following objects:

OBJECT OR FIELD	USAGE	DATA TYPE
`CRSP_ARRAY`	Structure for storing event-type data	
`objtype`	object type code identifies the structure as a `CRSP_ARRAY`, always = 3	`int`
`arrtype`	array type code defines the structure in the array. Base C types or CRSP- defined structures each have associated codes defined in the constants header file	`int`
`subtype`	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields	`int`
`size_of_array_width`	number of bytes in each array element	`int`
`maxarr`	maximum number of array elements containing valid data	`int`
`num`	number of array elements containing valid data	`int`
`dummy`	data secondary subtype code	`int`
`arr`	object array is a pointer to the array containing the actual data. The array can be a base C data type or a CRSP-defined structure. Its size and type are determined by `arrtype`, `size_of_array_width`, and `maxarr`	`void *`
`CRSP_ROW`	Structure for storing header data	
`objtype`	object type code identifies the structure as a `CRSP_ROW`, always = 5	`int`
`arrtype`	array type code defines the structure in the array. Base C types or CRSP- defined structures each have associated codes defined in the constants header file	`int`
`subtype`	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields	`int`
`size_of_array_width`	array structure size in bytes	`int`
`arr`	object array is a pointer to the array containing the actual data. The array can be a base C data type or a CRSP-defined structure. Its size and type are determined by arrtype and size_of_array_width. The array size is always 1	`void *`
`CRSP_TIMESERIES`	Structure for storing time series data	
`objtype`	object type code identifies the structure as a CRSP_TIMESERIES, always = 2	`int`
`arrtype`	array type code defines the structure in the array. Base C types or CRSP- defined structures each have associated codes defined in the constants header file	`int`
`subtype`	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields	`int`
`size_of_array_width`	array structure size in bytes	`int`
`maxarr`	maximum number of array elements	`int`
`beg`	first array index with valid data for the current record, or 0 if no valid range	`int`
`end`	last array index with valid data for the current record, or 0 if no valid range	`int`
`caltype`	calendar time period description code describes the type of time periods. Calendar Type (caltype) is always 2, indicating time periods are described in the Calendar Trading Date (caldt) array by the last trading date in the period	`int`
`cal`	calendar associated with time series is a pointer to the calendar associated with the time series array. The calendar includes the matching period- ending dates for each array index	`CRSP_CAL *`

OBJECT OR FIELD	USAGE	DATA TYPE
arr	object array is a pointer to the array containing the actual data. The array can be a base C data type or a CRSP-defined structure. Its size and type are determined by arrtype, size_of_array_width, and maxarr	void *
CRSP_CAL	Structure for storing calendar period data	
objtype	object type code identifies the structure as a CRSP_CAL, always = 1	int
calid	calendar identification number is an identifier assigned to each specific calendar by CRSP	int
type	generic group code of calendar, ie. daily or monthly. All current time series use 2 for calendar trading date (caldt) only	int
loadflag	calendar type availability flag is a code indicating the types of calendar arrays loaded. Currently = 2 for calendar trading date (caldt) only	int
maxarr	maximum number of trading periods allocated for the calendar	int
ndays	number of days is the index of the last calendar period	int
name	the calendar name in text	char[80]
callist	calendar period grouping identifiers reserved for array of alternate grouping identifiers for calendar periods	int *
caldt	calendar trading date is an array of calendar period ending dates, stored in YYYYMMDD. Calendars start at element 1 and end at element number of days (ndays)	int *
calmap	used to store array of first and last calendar period array elements in a linked calendar to elements in this calendar	CRSP_CAL_MAP *
basecal	used to point to a calendar linked in calmap	CRSP_CAL *

SET STRUCTURES AND USAGE

Stock and indexes access functions initialize and load data to C top-level defined set structures. Top-level structures are built from general object and array structure definitions and contain object and array pointers that have memory allocated to them by access open functions.

Two set types and six set identifiers are currently supported for stock and indexes data. The identifier must be speci- fied when opening or accessing data from the set.

DATA	SET TYPE	SET IDENTIFIERS	FREQUENCY
CRSP Stock Data	STK	10	Daily
		20	Monthly
CRSP Indexes Data	IND	400	Monthly Groups (in IX product only)
		420	Monthly Series
		440	Daily Groups (in IX product only)
		460	Daily Series

Each set structure has three types of pointer definitions.

- Module pointers point to `CRSP_OBJECT_ELEMENT` linked lists and are only needed internally to keep track of the objects in a module. These have the suffix _obj and can be ignored by ordinary programming.
- Object pointers define a `CRSP_ARRAY`, `CRSP_ROW`, or `CRSP_TIMESERIES` object type. A suffix, `_arr`,
- `_ts`, or `_row` is appended to the variable name. Range variables `num`, `beg`, and `end` are accessed from these variables.
- Array pointers define the data item array. The array has the same rank as the object but without the suffix. It is a pointer to the array element of the object and is used for general access of the data item.

If a module has multiple types of objects, a group structure is created with definitions for those objects and is included in the main structure.

If a module has a variable number of objects of one type, an integer variable keeps track of the actual number. These variables end with the suffix `types` and are based on the set type.

Each of the top-level structures contains three standard elements:

- PERMNO – the actual key loaded
- loadflag, a binary flag matching the set wanted parameters indicating which pointers have been allocated. See the open function for the set for more information about wanted parameters.
- setcode, a constant identifying the type of set (1=`STK`, 3=`IND`)

For example, a Stock Structure has `CRSP_TIMESERIES` object called `prc_ts` containing an array called `prc`.

C LANGUAGE DATA OBJECTS FOR CRSP STOCK DATA

Each stock structure is comprised of a fixed set of objects. Objects contain the header information required to use the CRSP data structures and the data arrays. Data elements are described in the C Data Structure Table under the array name.

Time series beg and end are both equal to 0 if there are no data. Otherwise beg > 0, beg <= end, and end <= maxarr. The 0th element of a time series array is reserved for the missing value of the underlying data type for that time series.

The stock structure contains an array of portfolio time series. Each member contains the portfolio statistic and assignment data for one portfolio type. Each member can have a different range and calendar. The count of Portfolio Types is found in the port types variable.

MODULE	OBJECT	NAME	OBJECT TYPE	ARRAY TYPE	DATA SUBTYPE	ARRAY STRUCTURE SIZE	RANGE ELEMENTS ON A SECURITY BASIS	ELEMENTS OF A SET BASIS	ARRAY NAME
`STK_HEAD` Header Module	`header_row`	Stock Header Structure	`CRSP_ROW`	`CRSP_STK_HEADER_NUM = 50`	0	172	none	none	`stk.header`
`STK_EVENTS` Event Arrays Module	`names_arr`	Security Name History	`CRSP_ARRAY`	`CRSP_STK_NAME_NUM = 51`	0	160	`num`	`maxarr`	`stk.events.names`
`STK_EVENTS` Event Arrays Module	`dists_arr`	Distribution History Array	`CRSP_ARRAY`	`CRSP_STK_DIST_NUM = 52`	0	40	`num`	`maxarr`	`stk.events.dists`
`STK_EVENTS` Event Arrays Module	`shares_arr`	Shares Structure Array	`CRSP_ARRAY`	`CRSP_STK_SHARE_NUM = 53`	`CRSP_SHARES_IMP_NUM = 0`	16	`num`	`maxarr`	`stk.events. shares`
`STK_EVENTS` Event Arrays Module	`delist_arr`	Delisting Structure Array	`CRSP_ARRAY`	`CRSP_STK_DELIST_NUM = 54`	0	40	`num`	`maxarr`	`stk.events. delist`
`STK_EVENTS` Event Arrays Module	`nasdin_arr`	Nasdaq Structure Array	`CRSP_ARRAY`	`CRSP_STK_NASDIN_NUM = 55`	0	24	`num`	`maxarr`	`stk.events. nasdin`

MODULE	OBJECT	NAME	OBJECT TYPE	ARRAY TYPE	DATA SUBTYPE	ARRAY STRUCTURE SIZE	RANGE ELEMENTS ON A SECURITY BASIS	ELEMENTS OF A SET BASIS	ARRAY NAME
STK_PORTS Portfolios Module	port_ts[ ]	Portfolio Statistics and Assignments	CRSP_TIMESERIES	CRSP_STK_PORT_NUM = 56	Each Portfolio time series in the array has subtype equal to the Permanent Index Identification Number of the associated group index	4	beg and end (for each portfolio time series)	maxarr, cal, stk.porttypes	stk.porttypes-1
STK_GROUPS Groups Module	group_arr[ ]	Array of Group Arrays	CRSP_ARRAY	CRSP_STK_GROUP_NUM=57	Each Group CRSP_ARRAY in the array has subtype equal to the Permanent Index Identification Number of an associated group index	16	num (for each group array)	maxarr, stk.grouptypes	stk.group
STK_LOW Bid or Low Data	bidlo_ts	Bid or Low	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.bidlo
STK_HIGHS Ask or High Data	askhi_ts	Ask or High	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	8	beg and end	maxarr, cal	stk.askhi
STK_PRCS Prices Module	prc_ts	Closing Price or Bid/ Ask Average	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.prc
STK_RETURNS Returns Module	ret_ts	Holding Period Return	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_RETURN_NUM = 2	4	beg and end	maxarr, cal	stk.ret
STK_VOLUMES Volumes Module	vol_ts	Share Volume	CRSP_TIMESERIES	CRSP_INTEGER_NUM = 2	CRSP_VOLUME_NUM = 6	4	beg and end	maxarr, cal	stk.vol
STK_BIDS Bids Module	bid_ts	Nasdaq Closing Bid	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.bid
STK_ASKS Asks Module	ask_ts	Nasdaq Closing Ask	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.ask
STK_RETX Returns Without Dividends Module	retx_ts	Return Without Dividends	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_RETURN_NUM = 2	4	beg and end	maxarr, cal	stk.retx
STK_SPREADS Bid/Ask Spreads Module	spread_ts	Month End Bid/Ask Spread	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.spread

MODULE	OBJECT	NAME	OBJECT TYPE	ARRAY TYPE	DATA SUBTYPE	ARRAY STRUCTURE SIZE	RANGE ELEMENTS ON A SECURITY BASIS	ELEMENTS OF A SET BASIS	ARRAY NAME
STK_TRADES Number of Trades Module (Daily) STK_ ALTPRCDTS Alternate Price Date Module (Monthly)	numtrd_ts or altprcdt_ts	Nasdaq Number of Trades or Alternate Price Date	CRSP_TIMESERIES	CRSP_INTEGER_NUM = 2	CRSP_COUNT_NUM = 7, or CRSP_DATE_NUM = 26	4	beg and end	maxarr, cal	stk.numtrd or stk.altprcdt
STK_OPENPRCS Open Price Module (Daily) STK_ALTPRCS Alternate Prices Module (Monthly)	openprc_ts or altprc_ts	Open Price or Alternate Price	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_PRICE_NUM = 1	4	beg and end	maxarr, cal	stk.openprc or stk.altprc

C LANGUAGE DATA STRUCTURE FOR CRSP STOCK DATA

All CRSP-defined data type structures have names in all capitals beginning with CRSP_ and are immediately followed by the definitions in the next level of indentation

Index and Date Ranges for all elements in a structure are the same as for the structure itself. There are three structure levels indicated by the indentation in the mnemonic field. Pointers at any level can be used in a program. The top level contains all other items and is used in all access functions. The second level indicates data grouped in modules. See the CRSPAccess Stock Users Guide for data item definitions.

All character strings, indicated by char[#], are NULL terminated. The number of characters -1 is the maximum string length allowed. Actual maximums may be lower. The top level stk structure is an example used by CRSP Stock sample programs. Other names can be used, and multiple CRSP_STK_STRUCTs can be declared in a program. See the CRSP_STK open access function for initializing a stock structure.

MNEMONIC	NAME	DATA TYPE	DATA USAGE	INDEX RANGE	DATE USAGE	OBJECT USAGE
stk	Master Stock Structure	**CRSP_STK_ STRUCKT**	stk			
header	Stock Header Structure					
permno	PERMNO	int	stk.header->permno			stk.header_row
permco	PERMCO	int	stk.header->permco			stk.header_row
compno	Nasdaq Company Number	int	stk.header->compno			stk.header_row
issuno	Nasdaq Issue Number	int	stk.header->issuno			stk.header_row

MNEMONIC	NAME	DATA TYPE	DATA USAGE	INDEX RANGE	DATE USAGE	OBJECT USAGE
hexcd	Exchange Code - Header	int	stk.header->hexcd			stk.header_row
hsiccd	Standard Industrial Classification (SIC) Code - Header	int	stk.header->hsiccd			stk.header_row
begdt	Begin of Stock Data	int	stk.header->begdt			stk.header_row
enddt	End of Stock Data	int	stk.header->enddt			stk.header_row
dlstcd	Delisting Code - Header	int	stk.header->dlstcd			stk.header_row
hcusip	CUSIP - Header	char[16]	stk.header->hcusip			stk.header_row
htick	Ticker Symbol - Header	char[16]	stk.header->htick			stk.header_row
hcomnam	Company Name - Header	char[36]	stk.header->hcomnam			stk.header_row
hnaics	North American Industry Classification System (NAICS) - Header	char[8]	stk.header->hnaics			stk.header_row
htsymbol	Trading Ticker Symbol - Header	char[12]	stk.header->htsymbol			stk.header_row
trdstat	Trading Status - Header	char[1]	stk.header->htrdstat			stk.header_row
hsecstat	Security Status - Header	char[1]	stk.header->hsecstat			stk.header_row
events	Master Stock Structure	**CRSP_STKEVENT_STRUCT**	**stk.events**			
names	Security Name History			i between 0 and stk.events.names_arr->num-1	name effective from stk.events.names[i].namedt to stk.events.names[i].nameenddt	**stk.events.names_arr**
namedt	Name Effective Date	int	stk.events.names[i].namedt			
nameenddt	Last Date of Name	int	stk.events.names[i].nameenddt			
ncusip	CUSIP	char[16]	stk.events.names[i].ncusip			
ticker	Ticker Symbol	char[8]	stk.events.names[i].ticker			
comnam	Company Name	char[36]	stk.events.names[i].comnam			
shrcls	Share Class	char[4]	stk.events.names[i].shrcls			
shrcd	Share Code	int	stk.events.names[i].shrcd			
exchcd	Exchange Code	int	stk.events.names[i].exchcd			
siccd	Standard Industrial Classification (SIC) Code	int	stk.events.names[i].siccd			
naics	North American Industry Classification System (NAICS)	char[8]	stk.events.names[i].naics			

MNEMONIC	NAME	DATA TYPE	DATA USAGE	INDEX RANGE	DATE USAGE	OBJECT USAGE
tsymbol	Trading Ticker Symbol	char[12]	stk.events.names[i].tsymbol			
trdstat	Trading Status	char[1]	stk.events.names[i].trdstat			
secstat	Security Status	char[1]	stk.events.names[i].secstat			
dists	Distribution History Array			i between 0 and stk.events.dists_arr->num-1	distribution effective on stk.events.dists[i].exdt	**stk.events.dists_arr**
distcd	Distribution Code	int	stk.events.dists[i].distcd			
divamt	Dividend Cash Amount	float	stk.events.dists[i].divamt			
facpr	Factor to Adjust Price	float	stk.events.dists[i].facpr			
facshr	Factor to Adjust Shares Outstanding	float	stk.events.dists[i].facshr			
dclrdt	Distribution Declaration Date	int	stk.events.dists[i].dclrdt			
exdt	Ex-Distribution Date	int	stk.events.dists[i].exdt			
rcrddt	Record Date	int	stk.events.dists[i].rcrddt			
paydt	Payment Date	int	stk.events.dists[i].paydt			
acperm	Acquiring PERMNO	int	stk.events.dists[i].acperm			
accomp	Acquiring PERMCO	int	stk.events.dists[i].accomp			
shares	Shares Structure Array			i between 0 and stk.events.shares_arr->num-1	shares observation effective from **stk.events.shares[i].shrsdt** to **stk.events.shares[i].shrsenddt**	**stk.events.shares_arr**
shrout	Shares Outstanding	int	stk.events.shares[i].shrout			
shrsdt	Shares Outstanding Observation Date	int	stk.events.shares[i].shrsdt			
shrsenddt	Shares Outstanding Observation End Date	int	stk.events.shares[i].shrsenddt			
shrflg	Shares Outstanding Observation Flag	int	stk.events.shares[i].shrflg			
delist	Delisting Structure Array			i between 0 and stk.events.delist_arr->num-1	delist observation on stk.events.delist[i].dlstdt	**stk.events.delist_arr**
dlstdt	Delisting Date	int	stk.events.delist[i].dlstdt			
dlstcd	Delisting Code	int	stk.events.delist[i].dlstcd			
nwperm	New PERMNO	int	stk.events.delist[i].nwperm			
nwcomp	New PERMCO	int	stk.events.delist[i].nwcomp			
nextdt	Delisting Date of Next Available Information	int	stk.events.delist[i].nextdt			
dlamt	Amount After Delisting	float	stk.events.delist[i].dlamt			

MNEMONIC	NAME	DATA TYPE	DATA USAGE	INDEX RANGE	DATE USAGE	OBJECT USAGE
dlretx	Delisting Return without Dividends	float	stk.events.delist[i].dlretx			
dlprc	Delisting Price	float	stk.events.delist[i].dlprc			
dlpdt	Delisting Payment Date	int	stk.events.delist[i].dlpdt			
dlret	Delisting Return	float	stk.events.delist[i].dlret			
nasdin	NASDAQ Structure Array			i between 0 and stk.events.nasdin_arr->num-1	NASDAQ status effective from **stk.events.nasdin[i].trtsdt** to **stk.events.nasdin[i].trtsenddt**	**stk.events.nasdin_arr**
trtsdt	NASDAQ Traits Date	int	stk.events.nasdin[i].trtsdt			
trtsenddt	NASDAQ Traits End Date	int	stk.events.nasdin[i].trtsenddt			
trtscd	NASDAQ Traits Code	int	stk.events.nasdin[i].trtscd			
nmsind	NASDAQ National Market Indicator	int	stk.events.nasdin[i].nmsind			
mmcnt	Market Maker Count	int	stk.events.nasdin[i].mmcnt			
nsdinx	NASD Index Code	int	stk.events.nasdin[i].nsdinx			
port	Portfolio Statistics and Assignments			j between 0 and stk.porttypes-1, i between stk.port_ts[j]->beg and stk.port_ts[j]->end	value for period ending stk.port_ts[j]->cal->caldt[i]	array of stk.port_ts
port	Portfolio Assignment Number	int	stk.port[j][i].port			
stat	Portfolio Statistic Value	double	stk.port[j][i].stat			
groups	Group Array			j between 0 and stk.group-1, i between stk.group_arr[j]->beg and stk.group_arr[j]->end	value for period ending stk.group_arr[j]->cal->caldt[i]	array of stk.group_arr
grpdt	Begin of Group Data	int	stk.group->grpdt			
grpenddt	End of Group Data	int	stk.group->grpenddt			
grpflag	Group Flag of Associated Index	int	stk.group->grpflag			
grpsubflag	Group Secondary Flag	int	stk.group->grpsubflag			
Time Series Data Arrays						
bidlo	Bid or Low Price	float *	stk.bidlo[i]	i between stk.bidlo_ts->beg and stk.bidlo_ts->end	value on date stk.bidlo_ts->cal->caldt[i]	stk.bidlo_ts
askhi	Ask or High Price	float *	stk.askhi[i]	i between stk.askhi_ts->beg and stk.askhi_ts->end	value on date stk.askhi_ts->cal->caldt[i]	stk.askhi_ts

MNEMONIC	NAME	DATA TYPE	DATA USAGE	INDEX RANGE	DATE USAGE	OBJECT USAGE
prc	Price or Bid/Ask Average	float *	stk.prc[i]	i between stk.prc_ts->beg and stk.prc_ts->end	value on date stk.prc_ts->cal->caldt[i]	stk.prc_ts
ret	Holding Period Total Return	float *	stk.ret[i]	i between stk.ret_ts->beg and stk.ret_ts->end	value on date stk.ret_ts->cal->caldt[i]	stk.ret_ts
vol	Volume Traded	int *	stk.vol[i]	i between stk.vol_ts->beg and stk.vol_ts->end	value on date stk.vol_ts->cal->caldt[i]	stk.vol_ts
bid	Bid	float *	stk.bid[i]	i between stk.bid_ts->beg and stk.bid_ts->end	value on date stk.bid_ts->cal->caldt[i]	stk.bid_ts
ask	Ask	float *	stk.ask[i]	i between stk.ask_ts->beg and stk.ask_ts->end	value on date stk.ask_ts->cal->caldt[i]	stk.ask_ts
retx	Return Without Dividends	float *	stk.retx[i]	i between stk.retx_ts->beg and stk.retx_ts->end	value on date stk.retx_ts->cal->caldt[i]	stk.retx_ts
spread	Spread Between Bid and Ask	float *	stk.spread[i]	i between stk.spread_ts->beg and stk.spread_ts->end	value on date stk.spread_ts->cal->caldt[i]	stk.spread_ts
altprc or numtrd	Price Alternate Date (monthly only) or Nasdaq Number of Trades (daily only)	int * int *	stk.altprcdt[i] or stk.numtrd[i]	i between stk.altprcdt_ts->beg and stk.numtrd_ts->end or i between stk.numtrd_ts->beg and stk.numtrd_ts->end	value on date stk.altprcdt_ts->cal->caldt[i] or value on date stk.numtrd_ts->cal->caldt[i]	stk.altprcdt_ts or stk.numtrd_ts
openprc or altprc	Open Price (daily only) or Price Alternate (monthly only)	float * float *	stk.openprc[i] or stk.altprc[i]	i between stk.openprc_ts->beg and stk.openprc_ts->end or i between stk.altprc_ts->beg and stk.altprc_ts->end	value on date stk.openprc_ts->cal->caldt[i] or value on date stk.altprc_ts->cal->caldt[i]	stk.openprc_ts or stk.altprc_ts

EXAMPLES OF C VARIABLE USAGE FOR CRSP STOCK DATA

These assume a variable stk of type `CRSP_STK_STRUCT`.

CRSP Row/Header Data

Object Variable: `stk.header_row`
Data Structure: `stk.header`
Sample Print Statement:

```
printf ("%d %8d-%8d\n", stk.header->permno,
stk.header->begdt, stk.header->enddt);
```

CRSP Array/Distributions

Object Variable: `stk.events.dists_arr`
Data Array: `stk.events.dists`
Sample Print Statement: This sample loop prints all distribution codes and ex-distribution dates.

```
for (i = 0; i < stk.events.dists_arr->num; ++i)

printf ("%4d %8d\n", stk.events.dists[i].distcd, stk.events.
dists[i].exdt);
```

CRSP Time Series/Prices

Object Variable: `stk.prc_ts`
Data Array: `stk.prc`
Sample Print Statement: This sample loop prints all prices and dates in the issue's range.

```
for(i = stk.prc_ts->beg; i <= stk.prc_ts->end; ++i)

printf("%11.5f %8d\n", stk.prc[i], stk.prc_ts->cal-
>caldt[i]);CRSP
```

CRSP Array of Time Series/Portfolios

Object Variable: `stk.port_ts[j]`
Data Array: `stk.port[j]`
(There are `stk.porttypes` portfolios available; j above is between 0 and stk. porttypes -1)

Sample Print Statement: This prints the associated `indno` and the sample loop prints the date and assignment for each year in the issue's range for `porttype=0` NYSE/NYSEMKT/NASDAQ Capitalization deciles.

```
printf ("indno = %d\n", stk.port_ts[0].subtype);
for (i = stk.port_ts[0]->beg; i <= stk.port_ts[0]->end; ++i)
printf ("%8d %2d\n", stk.port_ts[0]->cal->caldt[i],
stk.port[0][i].port);
```

CRSP Array of Group Arrays

Object Variable: `stk.group_arr[j]`
Data Array: `stk.group[j]`
(There are `stk.grouptypes` groups available; j above is between 0 and stk. grouptypes -1)
Sample Print Statement: This only prints if the security has ever been included in the S&P 500 universe (`grouptype` = 16).

```
j = 16 - 1;
for (i = 0; i < stk.group_arr[15]->num; ++i)
printf ("%8d %8d %2d %2d \n",
stk.group[j][i].grpdt,
stk.group[j][i].grpenddt,
stk.group[j][i].grpflag,
stk.group[j][i].grpsubflag);
```

C LANGUAGE DATA OBJECTS FOR CRSP INDEXES DATA

CRSP assigns a Permanent Index Identification Number (indno) to access the indexes data in C for individual series or portfolio groups. In the CRSP US Stock Database, a subset of market series is available. Additional series and groups are available when you subscribe to the CRSP US Indexes Database and Security Portfolio Assignment Module. The index structure supports data for one series or group and includes header, rebalancing, and result information for one or more portfolios comprising the index.

Each index structure contains a fixed set of possible objects. Objects contain the header information needed to use the CRSP data structures as well as the data arrays. Data elements are described in the C Data Structure Table under the array name.

Time series beg and end are both equal to 0 if there are no data. Otherwise beg > 0, beg <= end, and end < maxarr. The 0th element of a time series array is reserved for the missing value for that data type.

Multiple series in the index structure refers to portfolio subgroups. Each of these will have the same beg, end, and calendar. In a SERIES SETID, the multiple series has a count of 1. In a GROUP SETID, the count of series is found in the corresponding xxxtypes variable.

MODULE	OBJECT	NAME	OBJECT TYPE	ARRAY TYPE	DATA SUBTYPE	ARRAY STRUCTURE SIZE	RANGE ELEMENTS ON A SECURITY BASIS	ELEMENTS ON A SET BASIS	ARRAY NAME
IND_HEAD Index Description	indhdr_row	Indexes Header Object	CRSP_ROW	CRSP_IND_HEADER_NUM = 200	0	300	none	none	ind.indhdr
IND_REBAL Rebalancing Data	rebal_arr[ ]	Rebalancing Arrays	CRSP_ARRAY	CRSP_IND_REBAL_NUM = 201	0	64	num for each series	maxarr, ind. rebaltypes	ind.rebal[j], j from 0 to ind.rebaltypes - 1
IND_LISTS Issue Lists	list_arr[ ]	List Arrays	CRSP_ARRAY	CRSP_IND_LIST_NUM = 202	0	24	num for each series	maxarr, ind. listtypes	ind.list[j], j 0 to ind.listtypes -1
IND_USDCNTS Portfolio Used Counts	usdcnt_ts[ ]	Used Count Time Series	CRSP_TIMESERIES	CRSP_INTEGER_NUM = 2	CRSP_COUNT_NUM = 7	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.usdcnt[j], j from 0 to ind.indtypes -1
IND_TOTCNTS Portfolio Total Counts	totcnt_ts[ ]	Total Count Time Series	CRSP_TIMESERIES	CRSP_INTEGER_NUM = 2	CRSP_COUNT_NUM = 7	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.totcnt[j], j from 0 to ind.indtypes -1
IND_USDVALS Portfolio Used Weights	usdval_ts[ ]	Used Value Time Series	CRSP_TIMESERIES	CRSP_DOUBLE_NUM = 4	CRSP_WEIGHT_NUM = 4	8	beg and end for each series	maxarr, cal, ind.indtypes	ind.usdval[j], j from 0 to ind.indtypes -1
IND_TOTVALS Portfolio Total Weights	totval_ts[ ]	Total Value Time Series	CRSP_TIMESERIES	CRSP_DOUBLE_NUM = 4	CRSP_WEIGHT_NUM = 4	8	beg and end for each series	maxarr, cal, ind.indtypes	ind.totval[j], j from 0 to ind.indtypes -1
IND_TRETURNS Portfolio Total Returns	tret_ts[ ]	Total Return Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_RETURN_NUM = 2	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.tret[j], j from 0 ind.indtypes -1

MODULE	OBJECT	NAME	OBJECT TYPE	ARRAY TYPE	DATA SUBTYPE	ARRAY STRUCTURE SIZE	RANGE ELEMENTS ON A SECURITY BASIS	ELEMENTS ON A SET BASIS	ARRAY NAME
IND_ARETURNS Portfolio Capital Appreciation Returns	aret_ts[ ]	Capital Appreciation Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_RETURN_NUM = 2	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.aret[j], j from 0 ind.indtypes -1
IND_IRETURNS Portfolio Income Returns	iret_ts[ ]	Income Return Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_RETURN_NUM = 2	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.iret[j], j from 0 ind.indtypes -1
IND_TLEVELStal Return Index Levels	tind_ts[ ]	Total Return Index Level Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_LEVEL_NUM = 3	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.tind[j], j from 0 ind.indtypes -1
IND_ALEVELS Capital Appreciation Index Levels	aind_ts[ ]	Capital Appreciation Index Level Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_LEVEL_NUM = 3	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.aind[j], j from 0 ind.indtypes -1
IND_ILEVELS Income Return Index Levels	iind_ts[ ]	Income Return Index Level Time Series	CRSP_TIMESERIES	CRSP_FLOAT_NUM = 1	CRSP_LEVEL_NUM = 3	4	beg and end for each series	maxarr, cal, ind.indtypes	ind.iind[j], j 0 to ind.indtypes -1

C LANGUAGE DATA STRUCTURE FOR CRSP INDEXES DATA

All CRSP-defined data types have names in all capitals beginning with `CRSP_` and are immediately followed by the definitions in the next indented level.

Index and date ranges for all elements in a structure are the same as for the structure itself. There are four structure levels indicated by the indentation in the Mnemonic field. Pointers at any level can be used in a program. The top level contains all other items and is used in all access functions. The second level indicates data grouped in modules. See the Data Description Guide for data item definitions.

All character strings, indicated by `char[#]`, are null terminated. The number of characters - 1 is the maximum string length allowed. Actual maximums may be lower. The top level `ind` structure is an example used by CRSP Indexes sample programs. Other names can be used, and multiple `CRSP_IND_STRUCTs` may be declared in a program.

MNEMONIC	NAME	C DATA TYPE	C DATA USAGE	C INDEX RANGE	C DATE USAGE	C OBJECT TYPE
ind	Master Indexes Structure	**CRSP_IND_STRUCT**	**ind**			
indhdr	Indexes Header Object					**ind.indhdr_row**
indno	INDNO	int	ind.indhdr->indno			
indco	INDCO	int	ind.indhdr->indco			
primflag	Index Primary Link	int	ind.indhdr->primflag			
portnum	Portfolio Number if Subset Series	int	ind.indhdr->portnum			
indname	Index Name	char[80]	ind.indhdr->indname			
groupname	Index Group Name	char[80]	ind.indhdr->typename			
method	Index Methodology Description Structure	CRSP_IND_METHOD	ind.indhdr->method			
methcode	Index Method Type Code	int	ind.indhdr>method.methcode			
primtype	Index Primary Methodology Type	int	ind.indhdr>method.primtype			
subtype	Index Secondary Methodology Group	int	ind.indhdr->method.subtype			
wgttype	Index Reweighting Type Flag	int	ind.indhdr->method.wgttype			
wgtflag	Index Reweighting Timing Flag	int	ind.indhdr->method.wgtflag			
flags	Index Exception Handling Flags	CRSP_IND_FLAGS	ind.indhdr->flags			
flagcode	Index Basic Exception Types Code	int	ind.indhdr->flags.flagcode			
addflag	Index New Issues Flag	int	ind.indhdr->flags.addflag			
delflag	Index Ineligible Issues Flag	int	ind.indhdr->flags.delflag			
delretflag	Return of Delisted Issues Flag	int	ind.indhdr>flags.delretflag			
missflag	Index Missing Data Flag	int	ind.indhdr->flags.missflag			
partuniv	Index Subset Screening Structure	CRSP_UNIV_PARAM	ind.indhdr->partuniv			

MNEMONIC	NAME	C DATA TYPE	C DATA USAGE	C INDEX RANGE	C DATE USAGE	C OBJECT TYPE
partunivcode	Universe Subset Types Code in a Partition Restriction	int	ind.indhdr>partuniv.univcode			
begdt	Partition Restriction Beginning Date	int	ind.indhdr->partuniv.begdt			
enddt	Partition Restriction End Date	int	ind.indhdr->partuniv.enddt			
wantexch	Valid Exchange Codes in the Universe in a Partition Restriction	int	ind.indhdr>partuniv.wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in a Partition Restriction	int	ind.indhdr>partuniv.wantnms			
wantwi	Valid When-Issued Securities in the Universe in a Partition Restriction	int	ind.indhdr>partuniv.wantwi			
wantinc	Valid Incorporation of Securities in the Universe in a Partition Restriction	int	ind.indhdr>partuniv.wantinc			
shrcd	Share Code Screen Structure in a Partition Restriction	CRSP_UNIV_SHRCD	ind.indhdr->partuniv.shrcd			
sccode	Share Code Groupings for Subsets in a Partition Restriction	int	ind.indhdr>partuniv.shrcd. sccode			
fstdig	Valid First Digit of Share Code in a Partition Restriction	int	ind.indhdr>partuniv.shrcd. fstdig			
secdig	Valid Second Digit of Share Code in a Partition Restriction	int	ind.indhdr>partuniv.shrcd. secdig			
induniv	Partition Subset Screening Structure	CRSP_UNIV_PARAM	ind.indhdr->induniv			
indunivcode	Universe Subset Types Code in an Index Restriction	int	ind.indhdr>induniv.univcode			
begdt	Restriction Begin Date	int	ind.indhdr->induniv.begdt			
enddt	Restriction End Date	int	ind.indhdr->induniv.enddt			
wantexch	Valid Exchange Codes in the Universe in an Index Restriction	int	ind.indhdr>induniv.wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in an Index Restriction	int	ind.indhdr>induniv.wantnms			
wantwi	Valid When-Issued Securities in the Universe in an Index Restriction	int	ind.indhdr->induniv.wantwi			

MNEMONIC	NAME	C DATA TYPE	C DATA USAGE	C INDEX RANGE	C DATE USAGE	C OBJECT TYPE
wantinc	Valid Incorporation of Securities in the Universe in an Index Restriction	int	ind.indhdr>induniv.wantinc			
shrcd	Share Code Screen Structure in an Index Restriction	CRSP_UNIV_SHRCD	ind.indhdr->induniv.shrcd			
sccode	Share Code Groupings for Subsets in an Index Restriction	int	ind.indhdr>induniv.shrcd.sccode			
fstdig	Valid First Digit of Share Code in an Index Restriction	int	ind.indhdr>induniv.shrcd.fstdig			
secdig	Valid Second Digit of Share Code in an Index Restriction	int	ind.indhdr>induniv.shrcd.secdig			
rules	Portfolio Building Rules Structure	CRSP_IND_RULES	ind.indhdr->rules			
rulecode	Index Basic Rule Types Code	int	ind.indhdr->rules.rulecode			
buyfnct	Index Function Code for Buy Rules	int	ind.indhdr->rules.buyfnct			
sellfnct	Index Function Code for Sell Rules	int	ind.indhdr->rules.sellfnct			
statfnct	Index Function Code for Generating Statistics	int	ind.indhdr->rules.statfnct			
groupflag	Index Statistic Grouping Code	int	ind.indhdr>rules.groupflag			
assign	Related Assignment Information	CRSP_IND_ASSIGN	ind.indhdr->assign			
assigncode	Index Basic Assignment Types Code	int	ind.indhdr>assign.assigncode			
asperm	INDNO of Associated Index	int	ind.indhdr->assign.asperm			
asport	Portfolio Number in Associated Index	int	ind.indhdr->assign.asport			
rebalcal	Calendar Identification Number of Rebalancing Calendar	int	ind.indhdr>assign.rebalcal			
assigncal	Calendar Identification Number of Assignment Calendar	int	ind.indhdr>assign.assigncal			
calccal	Calendar Identification Number of Calculations Calendar	int	ind.indhdr->assign.calccal			
rebal	Array of Rebalancing Arrays	**int**	**ind.rebal[j][i].rbbegdt**	**j** between 0 and **ind.rebaltypes - 1**, **i** between 0 and **ind.rebal_arr[j]->num-1**	data valid from **ind.rebal[j][i].rbbegdt** to **ind.rebal[j][i].rbenddt**	array of **ind.rebal_arr**
rbbegdt	Index Rebalancing Begin Date	int	ind.rebal[j][i].rbbegdt			
rbenddt	Index Rebalancing End Date	int	ind.rebal[j][i].rbenddt			
usdcnt	Count Used as of Rebalancing	int	ind.rebal[j][i].usdcnt			

MNEMONIC	NAME	C DATA TYPE	C DATA USAGE	C INDEX RANGE	C DATE USAGE	C OBJECT TYPE
maxcnt	Maximum Count During Period	int	ind.rebal[j][i].maxcnt			
totcnt	Count Available as of Rebalancing	int	ind.rebal[j][i].totcnt			
endcnt	Count at End of Rebalancing Period	int	ind.rebal[j][i].endcnt			
minid	Statistic Minimum Identifier	int	ind.rebal[j][i].minid			
maxid	Statistic Maximum Identifier	int	ind.rebal[j][i].maxid			
minstat	Statistic Minimum in Period	double	ind.rebal[j][i].minstat			
maxstat	Statistic Maximum in Period	double	ind.rebal[j][i].maxstat			
medstat	Statistic Median in Period	double	ind.rebal[j][i].medstat			
avgstat	Statistic Average in Period	double	ind.rebal[j][i].avgstat			
list				j between 0 and ind.listtypes - 1, i between 0 and ind.list_arr[j]->num-1	valid from ind.list[j][i].beg to ind.list[j][i].enddt	array of ind.list_arr
list	List Arrays	int	ind.list[j][i].permno			
permno	Permanent Number of Securities in Index List	int	ind.list[j][i].permno			
begdt	First Date Included in List	int	ind.list[j][i].begdt			
enddt	Last Date Included in a List	int	ind.list[j][i].enddt			
subind	Index Subcategory Code	int	ind.list[j][i].subind			
weight	Weight of an Issue	double	ind.list[j][i].weight			
Time Series Data Arrays						
aind	Index Capital Appreciation Index Level	float*	ind.aind[j][i]	j between 0 and indtypes-1, i between ind.aind_ts[j]->beg and ind.aind_ts[j]->end	value on date ind.aind_ts[j]->cal>caldt[i]	array of ind.aind_ts
aret	Index Capital Appreciation Return	float*	ind.aret[j][i]	j between 0 and indtypes-1, i between ind.aret_ts[j]->beg and ind.aret_ts[j]->end	value on date ind.aret_ts[j]->cal>caldt[i]	array of ind.aret_ts
iind	Index Income Index Level	float*	ind.iind[j][i]	j between 0 and indtypes-1, i between ind.iind_ts[j]->beg and ind.iind_ts[j]->end	value on date ind.iind_ts[j]->cal>caldt[i]	array of ind.iind_ts

MNEMONIC	NAME	C DATA TYPE	C DATA USAGE	C INDEX RANGE	C DATE USAGE	C OBJECT TYPE
iret	Index Income Return	float*	ind.iret[j][i]	j between 0 and indtypes-1, i between ind.iret_ts[j]->beg and ind.iret_ts[j]->end	value on date ind.iret_ts[j]->cal>caldt[i]	array of ind.iret_ts
tind	Index Total Return Index Level	float*	ind.tind[j][i]	j between 0 and indtypes-1, i between ind.tind_ts[j]->beg and ind.tind_ts[j]->end	value on date ind.tind_ts[j]->cal>caldt[i]	array of ind.tind_ts
tret	Index Total Return	float*	ind.tret[j][i]	j between 0 and indtypes-1, i between ind.tret_ts[j]->beg and ind.tret_ts[j]->end	value on date ind.tret_ts[j]->cal>caldt[i]	array of ind.tret_ts
usdcnt	Index Used Count	float*	ind.usdcnt[j][i]	j between 0 and indtypes-1, i between ind.usdcnt_ts[j]->beg and ind.usdcnt_ts[j]->end	value on date ind.usdcnt_ts[j]>cal->caldt[i]	array of ind.usdcnt_ts
totcnt	Index Total Count	float*	ind.totcnt[j][i]	j between 0 and indtypes-1, i between ind.totcnt_ts[j]->beg and ind.totcnt_ts[j]->end	value on date ind.totcnt_ts[j]>cal->caldt[i]	array of ind.totcnt_ts
usdval	Index Used Value	float*	ind.usdval[j][i]	j between 0 and indtypes-1, i between ind.usdval_ts[j]->beg and ind.usdval_ts[j]->end	value on date ind.usdval_ts[j]>cal->caldt[i]	array of ind.usdval_ts
totval	Index Total Value	float*	ind.totval[j][i]	j between 0 and indtypes-1, i between ind.totval_ts[j]->beg and ind.totval_ts[j]->end	value on date ind.totval_ts[j]>cal->caldt[i]	array of ind.totval_ts

C SAMPLE PROGRAMS

There are two sample programs provided that can process the CRSP Stock Database using C. These programs can load stock and indexes data structures for processing. The sample program code contains additional comment information.

STK_SAMP1.C	Read Stock Data Sequentially	*stk_samp1.c* creates a namelist of current names by reading a stock database sequentially in PERMNO order. It loads one index series before processing the stock data. Output is one line of header information per security. stk_samp1.c accepts parameters for database directory, stock set identifier, indexes set identifier, INDNO, CRSP's permanent index identification number, and output file name.
STK_SAMP2.C	Read Stock Data with a PERMNO List File	*stk_samp2.c* reads a stock database using an input file of PERMNOs. It loads one set of indexes before processing the input list. Output is one line of header information per security. *stk_samp2.c* accepts parameters for database directory, stock set identifier, indexes set identifier, Permanent Index Identification Number, input file name, and output file name.
STK_SAMP3.C	Process an Input File of PERMNOs with Date Ranges	*stk_samp3.c* uses CRSP C library functions to read a space-delimited text input file with PERMNOs and beginning and ending date ranges in YYYYMMDD format. It outputs date, PERMNO, end of previous week, exchange code, end of current week adjusted price, end of current week index level for a selected index, end of previous week capitalization, and weekly total returns.

C HEADER FILES AND DATA STRUCTURES

Header files contain all needed structure definitions, constants, and function prototypes. Two C header files are suffi- cient to define all CRSP structures, constants, and functions.

1. *crsp.h* defines all structures and constants used by the CRSP C access and utility functions, and the function definitions. *crsp.h* includes several other header files. The primary definitions needed for stock databases are in *crsp_objects.h, crsp_const.h, crsp_stk_objects.h,* and *crsp_stk_const.h*. The primary definitions needed for the indexes data are in *crsp_objects.h, crsp_const.h, crsp_ind_objects.h,* and *crsp_ind_const.h*.

2. *crsp_init.h* declares internal variables needed to store initialization and error information. This should only be included in the main program and not in any function modules.

The following list is a more complete summary of individual stock and indexes header files that are included by *crsp.h*. All header files are kept in the `CRSP_INCLUDE` directory.

HEADER FILE	DESCRIPTION
`crsp_stk.h`	top level stock header file includes all needed header files for CRSP Stock access
`crsp_stk_objects.h`	defines top level CRSP_STK_STRUCT structure for Stock Data
`crsp_objects.h`	defines all object structures and data array structures for all supported types
`crsp_stk_const.h`	defines stock constants and wanted parameters
`crsp_const.h`	defines generic CRSP constants
`crsp_access_stk.h`	defines stock access function prototypes
`crsp_util_stk.h`	defines stock utility function prototypes
`crsp_ind.h`	top level indexes header file includes all needed header files for CRSP Indexes access
`crsp_ind_objects.h`	defines top level CRSP_IND_STRUCT structure for Indexes Data
`crsp_ind_const.h`	define indexes constants and wanted parameters
`crsp_access_ind.h`	defines index access function prototypes
`crsp_util_ind.h`	defines index utility function prototypes
`crsp_sysio.h`	defines system-specific constants
`crsp_maint.h`	defines internal data structures

CRSPACCESS C LIBRARY

The CRSPAccess C Library contains the Application Programming Interface (API) used to access and to process the data. The library is broken into sections based on the type of operations. The following major groups are available. Each can be further subdivided into subgroups. Functions within subgroups are alphabetical. Each function includes a function prototype, description, list of arguments, return values, side effects, and preconditions for use.

C LIBRARY CATEGORY	DESCRIPTION	PAGE
Stock Access Functions	Functions used to load stock data from the database into structures	page 114
Index Access Functions	Functions used to load index data from the database into structures	page 128
General Access Functions	General calendar and access functions	page 137
General Utility Functions	Functions utility to process base CRSPAccess structures	page 148
Data Utility Functions	Functions used to manipulate stock or indexes data	page 171

STOCK ACCESS FUNCTIONS

The following tables list the available functions to access CRSPAccess Stock Data. Standard usage is to use an open function, followed by successive reads and a close. Different databases and sets can be processed simultaneously if there is a matching structure defined for each one.

FUNCTION	DESCRIPTION	PAGE
`crsp_stk_clear`	Loads Missing Values to Arrays in a Stock Set Structure	page 114
`crsp_stk_close`	Closes a Stock Set	page 115
`crsp_stk_free`	Deallocates Memory and Reinitializes a Stock Set Structure	page 115
`crsp_stk_init`	Initializes a CRSPAccess Database for Stock Access	page 118
`crsp_stk_open`	Opens a Stock Set in a CRSPAccess Database	page 116
`crsp_stk_read`	Loads Wanted Stock Data For a PERMNO	page 120
`crsp_stk_read_cus`	Loads Wanted Stock Data Using Header CUSIP Identifier, Header as the Key	page 117
`crsp_stk_read_permco`	Loads Wanted Stock Data Using PERMCO as the Key	page 122
`crsp_stk_read_hcus`	Loads Wanted Stock Data Using Historical CUSIP as the Key	page 118
`crsp_stk_read_siccd`	Loads Wanted Stock Data Using Historical SIC Code as the Key	page 122
`crsp_stk_read_ticker`	Loads Wanted Stock Data Using Ticker Symbol, Header as the Key	page 120
`crsp_stk_read_subset`	Loads Wanted Stock Data for a PERMNO Applying All Subsetting Filters	page 121
`crsp_stk_read_key`	Loads Wanted Stock Data Using Any Supported Key	page 123
`crsp_stk_read_key_subset`	Loads Wanted Stock Data Using Supported Key Applying Subsetting Filters	page 121
`crsp_stk_alloc`	Allocates and Initializes Stock Structures	page 122
`crsp_stk_copy`	Copies Data from One Stock Structure to Another	page 122
`crsp_stk_delete`	Deletes Stock Data for an Existing PERMNO	page 122
`crsp_stk_insert`	Inserts New Stock Data for a PERMNO	page 123
`crsp_stk_modload`	Allocates and Loads a Module Structure	page 123
`crsp_stk_newset`	Inserts a Set of Stock Modules to a CRSP Root Directory	page 123
`crsp_stk_null`	Function to Zero out the Stock Structure Before Used	page 127
`crsp_stk_update`	Updates Stock Data for an Existing PERMNO	page 129
`crsp_stk_del_fromset`	Removes Modules from Stock Set from a CRSP Root Directory	page 124
`crsp_stk_add_toset`	Adds Modules to Stock Set to a CRSP Root Directory	page 124
`crsp_stk_get_allissues_key`	Get all Issues for a key	page 125

`crsp_stk_clear` Loads Missing Value Arrays in a Stock Set Structure

PROTOTYPE:	`int crsp_stk_clear (CRSP_STK_STRUCT *stk, int clearflag)`
DESCRIPTION:	Function to clear the stk structure before used. Load defined missing values to all allocated objects in a stock set structure. It is assumed that the pointers are either `NULL` or have been allocated by a set open function. The function allows clearing on a range level, range and array level, or array level.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to a stock structure pointer to be cleared. int `clearflag` – constant identifying the level of clearing. Supported values are: • `CRSP_CLEAR_INIT` – only reset num for CRSP_ARRAYs and beg and end for `CRSP_TIMESERIES`, and nothing for `CRSP_ROWs` • `CRSP_CLEAR_ALL` – set ranges to missing and sets missing values for all elements in the object arrays • `CRSP_CLEAR_RANGE` – set missing values for all elements in the object arrays within the range between beg and end in a `CRSP_TIMESERIES` or between 0 and num–1 in a `CRSP_ARRAY`, or the single element in a CRSP_ROW. • `CRSP_CLEAR_SET` – set ranges in the 0'th element of a `CRSP_TIMESERIES` array or the maxarr-1'th element of a `CRSP_ARRAY` to missing values specific to the array type, or missing values in `CRSP_ROW` element.
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters
SIDE EFFECTS:	The stock structure pointer has all allocated fields initialized according to the `clearflag`
PRECONDITIONS:	The stock structure must either have object fields set to `NULL` or allocated with a set open function.
CALL SEQUENCE:	Can be called after `crsp_stk_open` and before each `crsp_stk_read` call.

`crsp_stk_close` Closes a Stock Set

PROTOTYPE:	`int crsp_stk_close (int crspnum, int setid, CRSP_STK_STRUCT *stkptr)`
DESCRIPTION:	closes a stock set
ARGUMENTS:	`int crspnum` – identifier of CRSP database, as returned by open `int setid` – identifier of the stock set code to close `CRSP_STK_STRUCT *stkptr` – pointer to stock structure to be deallocated; if `NULL` nothing is deallocated
RETURN VALUES:	CRSP_SUCCESS: if successfully closed stock set CRSP_FAIL: if error closing a file or illegal parameter
SIDE EFFECTS:	All stock module files are closed, memory allocated by them are freed. If these are the last modules open in the database, the root is also closed. The stock structure associated with the set is deallocated if `stkptr` is not `NULL`.
PRECONDITIONS:	The crspnum and `setid` must be taken from a previous crsp_stk_open call
CALL SEQUENCE:	Called by external programs, must be preceded by call to crsp_stk_open calls crsp_closeroot, crsp_closemod.

`crsp_stk_free` Deallocates Memory and Reinitializes a Stock Set Structure

PROTOTYPE:	`int crsp_stk_free (int crspnum, int setid, CRSP_STK_STRUCT *stkptr)`
DESCRIPTION:	deallocates memory and reinitializes a stock set structure
ARGUMENTS:	`int crspnum` – identifier of crsp database, as returned by open `int setid` – identifier of the stock set code to close `CRSP_STK_STRUCT *stkptr` – pointer to stock structure
RETURN VALUES:	CRSP_SUCCESS: if successfully deallocated and reset stock structure, or stk structure is `NULL` CRSP_FAIL: if error deallocating memory, error in parameters
SIDE EFFECTS:	The stock structures are reset so all pointers are `NULL` and all settings are 0. All memory allocated to existing object element lists is freed.
PRECONDITIONS:	The `crspnum` must be known from a previous `crsp_stk_open` or `crsp_openroot` call. The `setcode` is an installation-defined code for the set.
CALL SEQUENCE:	Called by external programs or by `crsp_stk_close` must be preceded by call to `crsp_stk_alloc` calls `crsp_freemod`.

`crsp_stk_init` Initializes a CRSPAccess Database for Stock Access

PROTOTYPE:	`int crsp_stk_init( CRSP_STK_STRUCT *stkptr )`
DESCRIPTION:	initializes internal access for stock `CRSPDBs` and sets stock structure pointers to `NULL`. See `crsp_stk_clear` to clear data from a stock structure.
ARGUMENTS:	`CRSP_STK_STRUCT *stkptr` – pointer to a stock structure to initialize. This argument can be `NULL` to initialize a stock database without resetting the structure.
RETURN VALUES:	CRSP_SUCCESS: if stock internals successfully initialized CRSP_FAIL: if error opening or reading initialization file
SIDE EFFECTS:	Internal structures will be initialized, including the array of known stock sets. They will be stored in static structures in this module and used by other stk functions. All of the pointers in the stock structure stkptr will be set to NULL. If a structure is already initialized with crsp_stk_ open, crsp_stk_free should be used or memory will be lost.
PRECONDITIONS:	None; `crsp_stk_init` is called by `crsp_stk_open`

`crsp_stk_open` Opens an Existing Stock Set in a CRSPAccess Database

PROTOTYPE:	`int crsp_stk_open (char *root, int setid, CRSP_STK_STRUCT *stkptr, int wanted, char *mode, int bufferflag)`
DESCRIPTION:	opens an existing stock set in a CRSPAccess Database
ARGUMENTS:	`char *root` – path of root directory. If the root is `NULL` the `CRSP_DSTK` or `CRSP_MSTK` environment variables are used. `int setid` – the set identifier `10` – Daily CRSP Stock Database `20` – Monthly CRSP Stock Database `CRSP_STK_STRUCT *stkptr` – pointer within stock structure to be associated with this database. If wanted objects in stkptr are `NULL` then space for objects where the structure is allocated by this function. `int wanted` – mask indicating which modules will be used. The list below shows the wanted values for the stock modules. The wanted values can be summed or summary wanted values can be used to open multiple modules. Only modules that are selected in the wanted parameter have memory allocated in the stock structure and only those modules can be accessed in further access functions to the database. INDIVIDUAL MODULES: `STK_HEAD` 1 header structure `STK_EVENTS` 2 names, dists, shares, delists, nasdin `STK_LOWS` 4 lows `STK_HIGHS` 8 highs `STK_PRICES` 16 close or bid/ask average `STK_RETURNS` 32 total returns `STK_VOLUMES` 64 volumes `STK_PORTS` 128 portfolios `STK_BIDS` 256 bids `STK_ASKS` 512 asks `STK_RETXS` 1024 returns without dividends `STK_SPREADS` 2048 spreads `STK_TRADES/STK_ALTPRCDTS` 4096 number of trades or alternative price dates `STK_ALTPRCS/STK_OPENPRCS` 8192 alternate prices or open prices `STK_GROUPS` 6384 groups GROUP OF MODULES: `STK_INFOS` 3 header and event data `STK_DDATA` 124 price, high, low, volume and returns time series `STK_SDATA` 4864 bids, asks, and number of trades time series `STK_STD` 5119 header, events, prices, high, low, volume, returns, and ports `STK_ALL` 32767 all modules `char *mode` – usage while open (r=read, rw=read/write) `int bufferflag` – level of buffering: 0 : no buffering, 1 : use default, n : use factor of default

RETURN VALUES:	crspnum: (integer) if opened successfully. This crspnum is used in further access functions to the database. CRSP_FAIL: (integer) if error opening or loading files, if bad parameters, root already opened exclusively, stock set already opened rw, wanted not a subset of set's modules, set does not exist in root, set already opened and structure allocated, error allocating memory for internal or stock structures.
SIDE EFFECTS:	This will load root and stock initialization files if needed, open the root including loading the configuration structure and index structures to memory, opening the address file, and if necessary allocating memory to file buffers, loading the free list, and logging information to the log file. Files will be opened for all wanted modules. Associated calendars will be loaded if necessary. wanted stock structures will be allocated.
PRECONDITIONS:	None. The root may already be open under a different set in r mode.

crsp_stk_read Loads Wanted Stock Data for a Security by PERMNO

PROTOTYPE:	int crsp_stk_read (int crspnum, int setid, int *key, int keyflag, CRSP_STK_STRUCT*stkptr, int wanted)
DESCRIPTION:	loads wanted stock data for a PERMNO
ARGUMENTS:	int crspnum – crspdb root identifier returned by crsp_stk_open int setid – the set identifier used in crsp_stk_open int *key – specific PERMNO of data to load, or pointer to integer that will be loaded with the key found if a positional keyflag is used. int keyflag – CRSP_EXACT constant to search for the PERMNO in *key, or positional constant: CRSP_FIRST – the first key in the database CRSP_PREV – the previous key CRSP_LAST – the last key in the database CRSP_SAME – the same key CRSP_NEXT – the next key CRSP_STK_STRUCT *stkptr – structure to load data int wanted – mask of flags indicating which module data to load. See crsp_stk_open for module codes.
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_EOF: if next or previous key at end or beginning of file CRSP_FOUND_OTHER: if key found in root, but not for this setid CRSP_NOT_FOUND: if key not found in root CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum error in read, impossible wanted
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If keyflag is a positional qualifier, the actual PERMNO found is loaded to *key. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous crsp_stk_open call. stkptr must have been passed to a previous crsp_stk_open call. wanted must be a subset of the wanted parameter passed to the crsp_stk_open function.

crsp_stk_read_cus Loads Wanted Stock Data Using CUSIP Identifier, Header as the Key

PROTOTYPE:	int crsp_stk_read_cus (int crspnum, int setid, char *cusip, int keyflag,CRSP_STK_STRUCT *stkptr, int wanted)
DESCRIPTION:	loads wanted stock data for a security using the CUSIP Identifier, Header (hcusip) as the key

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`char *cusip` – CUSIP Identifier, Header to load, or pointer to string that will be loaded with the key found if a positional `keyflag` is used.
	`int keyflag` – qualify conditions of key searches:
	`CRSP_EXACT` – only accept an exact match
	`CRSP_BACK` – ind last previous key if no exact match
	`CRSP_FORWARD` – find the first following key if no exact match or positional constant:
	`CRSP_FIRST` – the first key in the database
	`CRSP_PREV` – the previous key
	`CRSP_LAST` – the last key in the database
	`CRSP_SAME` – the same key
	`CRSP_NEXT` – the next key
	`CRSP_STK_STRUCT *stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load. See `crsp_stk_open` for module codes.

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if key not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened crspnum and stknum, error in read, impossible wanted, invalid CUSIP index
SIDE EFFECTS:	Data from the `wanted` modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If `keyflag` is a positional qualifier, the actual CUSIP Identifier, Header found is loaded to `*cusip`. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous `crsp_stk_open` call. `stkptr` must have been passed to a previous `crsp_stk_open` call. wanted must be a subset of the wanted parameter passed to the `crsp_stk_open` function.

`crsp_stk_read_permco` Loads Wanted Stock Data Using PERMCO as the Key

PROTOTYPE:	`int crsp_stk_read_permco (int crspnum, int setid, int *permco, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	loads wanted stock data for a security using PERMCO as the key

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`int *permco` – PERMCO to load, or pointer to an integer that will be loaded with the key found if a positional `keyflag` is used.
	`int keyflag` – positional qualifier or match qualifier – see `crsp_stk_read_cus`
	`CRSP_STK_STRUCT *stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load. See `crsp_stk_open` for module codes.

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if key not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened `crspnum` and `stknum`, error in read, impossible `wanted`, invalid `PERMCO` index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If `keyflag` is a positional qualifier, the actual PERMCO found is loaded to *PERMCO. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous `crsp_stk_open` call. `stkptr` must have been passed to a previous `crsp_stk_open` call. wanted must be a subset of the wanted parameter passed to the `crsp_stk_open` function.

`crsp_stk_read_hcus` Loads Wanted Stock Data Using Historical CUSIP as the Key

PROTOTYPE:	`int crsp_stk_read_hcus (int crspnum, int setid, char *cusip, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	loads wanted stock data for a security using historical CUSIP as the key

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`char *cusip` – historical CUSIP to load, or pointer to string that will be loaded with the key found if a positional `keyflag` is used.
	`int keyflag` – positional qualifier or nomatch qualifier– see `crsp_stk_read_cus`
	`CRSP_STK_STRUCT * stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load. See `crsp_stk_open` for module codes.

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if key not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid CUSIP index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If `keyflag` is a positional qualifier, the actual historical CUSIP found is loaded to `*cusip`. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous `crsp_stk_open` call. `stkptr` must have been passed to a previous `crsp_stk_open` call. wanted must be a subset of the wanted parameter passed to the `crsp_stk_open` function.

`crsp_stk_read_siccd` Loads Wanted Stock Data Using Historical SIC Code as the Key

PROTOTYPE:	`int crsp_stk_read_siccd (int crspnum, int setid, int *siccd, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	loads wanted stock data for a security using Standard Industrial Classification (SIC) Code (`siccd`) as the key

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`int *siccd` – `siccd` to load, or pointer to integer that will be loaded with the key found if a positional `keyflag` is used.
	`int keyflag` – positional qualifier or no match qualifier– see `crsp_stk_read_cus`
	`CRSP_STK_STRUCT *stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load. See `crsp_stk_open` for module codes.

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if key not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened crspnum and stknum, error in read, impossible `wanted`, invalid `siccd` index
SIDE EFFECTS:	Data from the `wanted` modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If `keyflag` is a positional qualifier, the actual SIC Code found is loaded to `*siccd`. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data.
PRECONDITIONS:	The stock set must be previously opened. The `crspnum` must be returned from a previous `crsp_stk_open` call. `stkptr` must have been passed to a previous `crsp_stk_open` call. wanted must be a subset of the wanted parameter passed to the `crsp_stk_open` function.

`crsp_stk_read_ticker` Loads the Wanted Stock Data Using Ticker, Header as the Key

PROTOTYPE:	`int crsp_stk_read_ticker (int crspnum, int setid, char *ticker, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	loads wanted stock data for a security using Ticker, Header as the key

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`char *ticker` – pointer to header ticker to load, or pointer to string that will be loaded with the key found if a positional `keyflag` is used.
	`int keyflag` – positional qualifier or no match qualifier– see `crsp_stk_read_cus`
	`CRSP_STK_STRUCT *stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load. See `crsp_stk_open` for module codes.

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if ticker not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid ticker index

SIDE EFFECTS:	Data from the `wanted` modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If `keyflag` is a positional qualifier, the actual header ticker found is loaded to `*ticker`. Data is only loaded to wanted data structures within the range of valid data for the security. Use stock clear functions to erase previously loaded data.
PRECONDITIONS:	The stock set must be previously opened. The `crspnum` must be returned from a previous `crsp_stk_open` call. `stkptr` must have been passed to a previous `crsp_stk_open` call. wanted must be a subset of the wanted parameter passed to the `crsp_stk_open` function.

`crsp_stk_read_subset` Loads `Wanted` Stock Data for a PERMNO Applying Subsetting Filters

PROTOTYPE:	int crsp_stk_read_subset (`int crspnum`, `int setid`, int *key, `int keyflag`, `CRSP_STK_STRUCT *stkptr`, `int wanted`, CRSP_UNIV_PARAM_LOAD *subpar)
DESCRIPTION:	loads `wanted` stock data for a PERMNO applying all subsetting filters

ARGUMENTS:	`int crspnum` – `crspdb` root identifier returned by `crsp_stk_open`
	`int setid` – the set identifier used in `crsp_stk_open`
	`int *key` – PERMNO to load
	`int keyflag` – positional qualifier or no match qualifier
	`CRSP_STK_STRUCT *stkptr` – structure to load data
	`int wanted` – mask of flags indicating which module data to load
	`CRSP_SUBSET_PARAM_LOAD *subpar` – pointer to structure containing subsetting flags

RETURN VALUES:	`CRSP_SUCCESS`: if data loaded successfully `CRSP_EOF`: if next or previous key at end or beginning of file `CRSP_NOT_FOUND`: if PERMNO not found `CRSP_FAIL`: if error with bad parameters, invalid or unopened `crspnum` and stknum, error in read, impossible `wanted`, invalid PERMNO index
SIDE EFFECTS:	Data from the `wanted` modules will be loaded to the proper location in the structure. The data loaded in the module buffers may be changed. The position for further reads will be set to the location of the read. Multiple PERMNOs may be loaded on a positional read if subsetting totally eliminates PERMNOs that otherwise would be loaded.
PRECONDITIONS:	The stock set must be previously opened. The stknum and `crspnum` and `stkptr` are the same as opened and the `wanted` must be a subset of the `wanted` open. The subset parameter structure must be loaded with valid flags. See the crsp_stk_subset_parload function on page 181.

`crsp_stk_read_key` Loads Wanted Stock Data Using Any Supported Key

PROTOTYPE:	`int crsp_stk_read_key (int crspnum, int setid, void *key, int keytype, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	Loads `wanted` stock data using any supported key. Supported keys are PERMNO, Header CUSIP, Historical CUSIP, Historical SIC Code, Header Ticker and PERMCO.

ARGUMENTS:	int crspnum – crspdb root identifier returned by crsp_stk_open int setid – the set identifier used in crsp_stk_open void *key – key must point to a structure that matches the keytype int if keytype = CRSP_SCD_NUM CRSP_SCD_CUS if keytype = CRSP_SCD_CUSIP CRSP_SCD_CUS if keytype = CRSP_SCD_HCUSIP CRSP_SCD_INT if keytype = CRSP_SCD_SICCD CRSP_SCD_CUS if keytype = CRSP_SCD_TICKER CRSP_SCD_INT if keytype = CRSP_SCD_PERMCO int keytype – The keyword identifying the key to search on. Values are: CRSP_SCD_CUSIP – Header CUSIP CRSP_SCD_HCUSIP – Historical CUSIP CRSP_SCD_SICCD – Historical SIC Codes CRSP_SCD_TICKER – Header Ticker CRSP_SCD_PERMCO – PERMCO CRSP_SCD_NUM – PERMNO int keyflag – positional qualifier or no match qualifier. Positioned qualifiers are dependent on the keys selected. CRSP_STK_STRUCT *stkptr – structure to load data int wanted – mask of flags indicating which module data to load
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_EOF – if next or previous key at end or beginning of file CRSP_NOT_FOUND – if PERMNO not found CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum and stknum, error in read, impossible wanted.
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stkptr structure. The data loaded in the module buffers may be changed.
PRECONDITIONS:	The stock set must be previously opened. The crspnum and stkptr are the same as opened and the wanted must be a subset of the open wanted.

crsp_stk_read_key_subset Loads Wanted Stock Data Using Supported Key Applying Subset Filters

PROTOTYPE:	int crsp_stk_read_key_subset (int crspnum, int setid, void *key, int keytype, int keyflag, CRSP_STK_STRUCT *stkptr, int wanted, CRSP_UNIV_PARAM_LOAD *subpar)
DESCRIPTION:	loads wanted stock data using supported key applying subsetting filters applied. Supported keys are PERMNO, Header CUSIP, Historical CUSIP, Historical SIC Code, Header Ticker and PERMCO.
ARGUMENTS:	int crspnum – crspdb root identifier returned by crsp_stk_open int setid – the set identifier used in crsp_stk_open void *key – key must point to a structure that matches keytype int if keytype = CRSP_SCD_NUM CRSP_SCD_CUS if keytype = CRSP_SCD_CUSIP CRSP_SCD_CUS if keytype = CRSP_SCD_HCUSIP CRSP_SCD_INT if keytype = CRSP_SCD_SICCD CRSP_SCD_CUS if keytype = CRSP_SCD_TICKER CRSP_SCD_INT if keytype = CRSP_SCD_PERMCO int keytype - CRSP_SCD_CUSIP – Header CUSIP CRSP_SCD_HCUSIP – Historical CUSIP CRSP_SCD_SICCD – Historical SIC Code CRSP_SCD_TICKER – Header Ticker CRSP_SCD_PERMCO – PERMCO CRSP_SCD_NUM – PERMNO int keyflag – positional qualifier or no match qualifier CRSP_STK_STRUCT *stkptr – structure to load data int wanted – mask of flags indicating which module data to load CRSP_SUBSET_PARAM_LOAD *subpar – pointer to structure containing subsetting flags

RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_EOF: if next or previous key at end or beginning of file CRSP_NOT_FOUND: if key not found CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum and stknum, error in read, impossible wanted.
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stkptr structure. The data loaded in the module buffers may be changed. The position of further reads will be set to the location of the read. Multiple PERMNOs may be loaded and discarded on a positional read if subsetting totally eliminates PERMNOs that otherwise would be loaded.
PRECONDITIONS:	The stock set must be previously opened. The crspnum and stkptr are the same as opened and the wanted must be a subset of the open wanted. The subset parameter structure must be loaded with valid flags. See the crsp_stk_subset_parload function page 181.

crsp_stk_alloc Builds Stock Set Object Lists, Allocates Memory, and Sets Pointers

PROTOTYPE:	int crsp_stk_alloc (int crspnum, int setid, CRSP_STK_STRUCT *stk, int wanted)
DESCRIPTION:	Build stock set object lists, allocate memory, and set pointers
ARGUMENTS:	int crspnum - identifier of CRSP database, as returned by open int setid - identifier of the stock set to allocate CRSP_STK_STRUCT *stk - pointer to stock structure int wanted - binary code of modules wanted; see CRSP_STK_OPEN.
RETURN VALUES:	CRSP_SUCCESS - if successfully initialized and allocated stock structure CRSP_FAIL - if error allocating memory, error in parameters
SIDE EFFECTS:	Three levels of pointers are allocated in the stock structure. object_element list elements are created for each wanted module object types are allocated for each object in wanted modules, and object level pointers are set arrays are allocated for each object, and array level pointers are set setid and wanted are stored in the structure.
PRECONDITIONS:	The crspnum must be known from a previous crsp_stk_open or crsp_openroot call. The setid is an installation-defined code for the set.
CALL SEQUENCE:	Called by external programs or by crsp_stk_open. Must be preceded by call to crsp_stk_open or crsp_openroot. Calls crsp_allocmod

crsp_stk_copy Copies Data from One Stock Structure to Another

PROTOTYPE:	int crsp_stk_copy (CRSP_STK_STRUCT *stktrg, CRSP_STK_STRUCT *stksrc, int wanted)
DESCRIPTION:	Copies data from one stock structure to another
ARGUMENTS:	CRSP_STK_STRUCT *stktrg - pointer to stock structure target CRSP_STK_STRUCT *stksrc - pointer to stock structure source int wanted - wanted flag of modules to copy
RETURN VALUES:	CRSP_SUCCESS - if successfully copied data from source to target CRSP_FAIL - if incompatible structures
SIDE EFFECTS:	Data is copied from the source structure to the target structure. The loadflag field is used to identify all wanted modules to copy.
PRECONDITIONS:	Both structures must be allocated with crsp_stk_open. The wanted for the target must be a superset of the wanted in the source. The versions of the structure must be compatible - the source modules must not have missing objects or higher maxarrs than the counterparts in the target.
CALL SEQUENCE:	Called by external programs must be preceded by call to crsp_stk_open for each structure.

crsp_stk_delete Deletes a PERMNO from a Stock Set

PROTOTYPE:	int crsp_stk_delete (int crspnum, int setid, int key)
DESCRIPTION:	Deletes a PERMNO from a stock set in a CRSPAccess database.
ARGUMENTS:	int crspnum - identifier of root int setid - identifier of the set int key - PERMNO to erase from stock set

RETURN VALUES:	CRSP_SUCCESS - if successfully deleted from stock set CRSP_FAIL - if key not found, not found in set, not open for rewrite, or illegal parameter
SIDE EFFECTS:	If the PERMNO is found in the set it is erased. All space allocated in the module files for this permno will be added to the free list for those modules. If the PERMNO does not belong to any other modules, it will be erased from the index file and its address record placed on the address file free list. Otherwise, the index file module flags will be reset and the address records for these stock modules will be set to NULL.
PRECONDITIONS:	The root and stock set must be opened previously with crsp_stk_open. The open must use the rw mode.

crsp_stk_insert Adds a New PERMNO to a Stock Set

PROTOTYPE:	int crsp_stk_insert (int crspnum, int setid, int key, CRSP_STK_STRUCT *stkptr, int wanted)
DESCRIPTION:	Adds a new PERMNO to a stock set in a CRSPAccess database.
ARGUMENTS:	int crspnum - CRSP database identifier from open int setid - the set identifier CRSP_STK_STRUCT *stkptr - pointer to stock structure - data that will be added to the wanted modules int wanted - a stock wanted parameter indicating which stock data modules include data that will be saved.
RETURN VALUES:	CRSP_SUCCESS - if successfully added CRSP_FAIL - if bad parameters, database not open for rw, PERMNO already exists in this set
SIDE EFFECTS:	Data for all modules will be added to the proper file. The free list is used to find a location in the module file and may be updated if free space is used. If the key already exists but in different sets, the index file module flag is updated, and the new module addresses and sizes are added to the address file. If the key is totally new, a new index file row and address file record are created.
PRECONDITIONS:	The stock set must be opened previously with rw mode.

crsp_stk_modload Allocates a Module Structure and Loads a Module and Objects Information into Module Structure

PROTOTYPE:	int crsp_stk_modload (int crspnum, int modindex, int modid, CRSP_CONFIG_MOD **modstruct)
DESCRIPTION:	Allocates a module structure and loads a module and object information into it
ARGUMENTS:	int modindex - the index of the module in the CRSP_MODTYPE structure array CRSP_CONFIG_MOD *modstruct - pointer to the CRSP_CONFIG_MOD structure
RETURN VALUES:	CRSP_SUCCESS - if the module is loaded successfully CRSP_FAIL - if bad parameters or error allocating and loading module structures

crsp_stk_newset Adds a Set of Stock Modules to a CRSP Database

PROTOTYPE:	int crsp_stk_newset (char *root, int setid, int wanted)
DESCRIPTION:	Adds a set of stock modules to a CRSPAccess database. Creates a new database if one does not exist.
ARGUMENTS:	char *root – path of crspdb root directory int setid - known stock set number from initialization file int wanted - mask determining which stock modules are supported in the set; see CRSP_STK_OPEN.
RETURN VALUES:	CRSP_SUCCESS - if the stock set is added CRSP_FAIL - if bad parameters or error manipulating root structures
SIDE EFFECTS:	crsp_stk_newset creates a new CRSPAccess database if one does not exists; it then adds a set of stock modules to the existing crspdb. It will add the information about the set to the configuration file and recreate the address file with the new modules added to each record. Empty data files for the modules will be created. If the calendars are new to the root they will be added. The new modules will be assigned to the proper calendars.
CALL SEQUENCE:	crsp_newroot is called to create a CRSPAccess database if none exists. The root must be a CRSPAccess database, unopened, not including this setid, or an empty directory.

`crsp_stk_null` Function Zeros Out the Stock Structure Before it is Used

PROTOTYPE:	`int crsp_stk_NULL(CRSP_STK_STRUCT *stkptr)`
DESCRIPTION:	Function zeros out the stock structure before it is used. All pointers are set to `NULL` and integers set to 0. This does not free memory. Use `CRSP_STK_CLEAR` to reset data in an allocated structure.
ARGUMENTS:	`CRSP_STK_STRUCT *stkptr` - pointer to stock structure
RETURN VALUES:	`CRSP_SUCCESS` - if stock internals successfully initialized `CRSP_FAIL` - if error opening or reading initialization file
SIDE EFFECTS:	The stock structure will be set to zero according to the loadflag
PRECONDITIONS:	None.

`crsp_stk_update` Updates Stock Data for a Key

PROTOTYPE:	`int crsp_stk_update (int crspnum, int setid, int key, CRSP_STK_STRUCT *stkptr, int wanted)`
DESCRIPTION:	Updates stock data for a key
ARGUMENTS:	`int crspnum` - CRSP database root identifier returned by `crsp_stk_open` `int setid` - the set identifier used in `crsp_stk_open` int key - specific PERMNO of data to write `CRSP_STK_STRUCT *stkptr` - structure containing new data `int wanted` - mask of flags indicating which module data to write
RETURN VALUES:	`CRSP_SUCCESS` - if data written successfully `CRSP_FOUND_OTHER` - if key found in root, but not for this set `CRSP_NOT_FOUND` - if key not found in root `CRSP_FAIL` - if error with bad parameters, invalid or unopened `crspnum` and `stknum` for rw, error in write, impossible `wanted`.
SIDE EFFECTS:	Data from the `wanted` modules will be written to the proper locations in the module files. The address file may be updated for new offsets and sizes. If the new data does not fit within the allocated space for that key in the module file the data may be moved to a new location and the free list modified. The data loaded in the module buffers may be changed.
PRECONDITIONS:	The stock set must be previously opened with `rw`. The `crspnum` and `setid` are the same as opened and the `wanted` must be a subset of the `wanted` open. The `stkptr` must be compatible with the structure allocated by the open of this `crspnum` and `setid`.

`crsp_stk_del_fromset` Deletes Modules from Stock Set from a CRSP Root

PROTOTYPE:	`int crsp_stk_del_fromset (char *root, int setid, int wanted)`
DESCRIPTION:	Deletes modules from stock set from a CRSP root
ARGUMENTS:	`char *root` - path of CRSP database root `int setid` - identifier of the set `int wanted` - binary code of modules `wanted` to delete
RETURN VALUES:	`CRSP_SUCCESS` - if the stock modules are removed successfully `CRSP_FAIL` - if something wrong
SIDE EFFECTS:	`crsp_stk_del_fromset` removes the `wanted` modules associated with a given stock set from a CRSP database root. All `wanted` modules will be erased from the address file, which will be rewritten with a new restricted record length. The index file will also be rewritten, with keys changed to new module inclusion flags or erased altogether. The configuration file will be rewritten without the modules included in the stk set. `Wanted` module files of this set will be deleted. If all modules are deleted, delete and the set.
PRECONDITIONS:	`crsp_stk_del_fromset` is run off an unopened CRSP database
POSTCONDITIONS:	Will leave an unopened CRSP database root

`crsp_stk_add_toset` Adds Modules to an Existing Stock Set

PROTOTYPE:	`int crsp_stk_add_toset (char *root, int setid, int wanted)`
DESCRIPTION:	Adds modules to an existing stock set in a CRSPAccess database.
ARGUMENTS:	`char *root` – path of CRSP database root `int setid` - identifier of the set

RETURN VALUES:	`CRSP_SUCCESS` - if the modules are added `CRSP_FAIL` - if bad parameters
SIDE EFFECTS:	Extra module files are created and attach to the database configuration file. A new address file is created in the database.
PRECONDITIONS:	Database must exist with set included. It is unopened. Permission must exist to write to the database root.

`crsp_stk_get_allissues_key` Gets All Issues Associated with Key

PROTOTYPE:	`int crsp_stk_get_allissues_key (int crspnum, int setid, CRSP_ARRAY *issue_arr, void *key, int keytype, CRSP_UNIV_PARAM_LOAD *subpar, CRSP_STK_STRUCT *stk, int begdt, int enddt, int dateflag)`
DESCRIPTION:	Gets all issues associated with key and store them in `CRSP_TSP_ENTITY_LIST` array. Can be called multiple times to append to list.

ARGUMENTS:	`int crspnum` - CRSP database root identifier returned by `crsp_stk_open`
	`int setid` - the set identifier used in `crsp_stk_open`
	`CRSP_ARRAY *issue` - integer array to load found PERMNOs. The array type must be initialized to `CRSP_TSP_ENTITY_LIST`.
	`void *key` - key must point to a structure that matches keytype
	`int` if `keytype` = `CRSP_SCD_NUM`
	`CRSP_SCD_CUS if keytype == CRSP_SCD_CUSIP`
	`CRSP_SCD_CUS if keytype == CRSP_SCD_HCUSIP`
	`CRSP_SCD_INT if keytype == CRSP_SCD_SICCD`
	`CRSP_SCD_CUS if keytype == CRSP_SCD_TICKER`
	`CRSP_SCD_INT if keytype == CRSP_SCD_PERMCO`
	`int keytype - CRSP_SCD_CUSIP CRSP_SCD_HCUSIP`
	`CRSP_SCD_SICCD CRSP_SCD_TICKER CRSP_SCD_PERMCO`
	`CRSP_SCD_NUM` (primary - PERMNO)
	`CRSP_UNIV_PARAM_LOAD *subpar` - structure specifying subset restrictions. See `CRSP_STK_UBSET_PARAMLOAD` on page page 181 for options.
	`CRSP_STK_STRUCT *stk` - allocated stock structure used to store immediate data for determining matches.
	`int begdt` - yyyymmdd format. If stock data is not within begdt and enddt, ignore the PERMNO
	`int enddt` - yyyymmdd format. If stock data is not between begdt and enddt, ignore the PERMNO
	`int dateflag` - whether the date is relative date `CRSP_TSP_RELDATE (1)` or not `CRSP_TSP_NO_RELDATE (0)`

RETURN VALUES:	`CRSP_SUCCESS` - if array loaded successfully `CRSP_FAIL` - error in parameters, reads, or limits
SIDE EFFECTS:	Issue array is loaded with matching issues. Only the PERMNO field is loaded.
PRECONDITIONS:	The stock set must be previously opened. `crspnum` and `stkptr` are the same as opened. Stock structure must have `wanted` at least HEADER and EVENTS, and also PRICES if subset restrictions are used. Issue array must be allocated with enough space to store possible keys. If num is nonzero, new matches will be added to the end of the list.

INDEX ACCESS FUNCTIONS

The following tables list the available functions to access CRSPAccess index data. Standard usage is to use an open function, followed by successive reads and a close. Different databases and sets can be processed simultaneously if there is a matching structure defined for each one.

ACCESS FUNCTION	DESCRIPTION	PAGE
`crsp_ind_clear`	Loads Missing Value Arrays in an Indexes Set Structure	page 126
`crsp_ind_close`	Closes an Indexes Set	page 126
`crsp_ind_free`	Deallocates Memory And Reinitializes an Indexes Set Structure	page 126
`crsp_ind_init`	Initializes a CRSPAccess database for Indexes Access	page 127
`crsp_ind_open`	Opens an Indexes Set in a CRSPAccess Database	page 127
`crsp_ind_read`	Loads `Wanted` Data For an Index	page 131
`crsp_ind_alloc`	Allocates and Initializes Indexes Structures	page 129
`crsp_ind_copy`	Copies Data from One Stock Structure to Another	page 129
`crsp_ind_delete`	Deletes Indexes Data for an Existing INDNO	page 130
`crsp_ind_insert`	Inserts New Indexes Data for a PERMNO	page 130
`crsp_ind_modload`	Allocates and Loads a Module Structure	page 130

ACCESS FUNCTION	DESCRIPTION	PAGE
crsp_ind_newset	Inserts a Set of Indexes Modules to a Root	page 131
crsp_ind_NULL	Function to Zero Out the Index Structure Before it is Used	page 131
crsp_ind_read_subset	Reads Indexes Data for One INDNO Applying Subsets	page 131
crsp_ind_update	Updates Indexes Data for an Existing INDNO	page 131
crsp_ind_del_fromset	Removes Modules from Indexes Set from a Root Directory	page 132
crsp_ind_add_toset	Adds Modules to Indexes Set to a Root Directory	page 132
crsp_ind_free_ind	Frees Memory for Allocated Indexes Structure	page 132

crsp_ind_clear Loads Missing Value Arrays in an Index Set Structure

PROTOTYPE:	int crsp_ind_clear (CRSP_IND_STRUCT *ind, int clearflag)
DESCRIPTION:	load defined missing values to all allocated objects in an index set structure. It is assumed that the pointers are either NULL or have been allocated by a set open function. The function allows clearing on a range level, range and array level, or array level.
ARGUMENTS:	CRSP_IND_STRUCT *ind – pointer to an index structure pointer to be cleared. int clearflag – constant identifying the level of initialization. Supported values are: CRSP_CLEAR_INIT – only reset num for CRSP_ARRAYs and beg and end for CRSP_TIMESERIES, and set structure to missing values for CRSP_ROWs. CRSP_CLEAR_ALL – set ranges to missing and set missing values for all elements in the object arrays CRSP_CLEAR_RANGE – set missing values for all elements in the object arrays within the range between beg and end in a CRSP_TIMESERIES or between 0 and num–1 in a CRSP_ARRAY, or the single element in a CRSP_ROW. CRSP_CLEAR_SET – set ranges in the 0'th element of a CRSP_TIMESERIES array or the maxarr-1'th element of a CRSP_ARRAY to missing values specific to the array type, or sets missing values to the single element in a CRSP_ROW.
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters
SIDE EFFECTS:	The index structure pointer has all allocated fields initialized according to the clearflag
PRECONDITIONS:	The index structure must either have object fields set to NULL or allocated with a set open function.
CALL SEQUENCE:	call after crsp_ind_open and before each crsp_ind_read.

crsp_ind_close Closes an Index Set

PROTOTYPE:	int crsp_ind_close (int crspnum, int setid, CRSP_IND_STRUCT *indptr)
DESCRIPTION:	close an index set
ARGUMENTS:	int crspnum – identifier of the CRSP database, as returned by crsp_ind_open int setid – identifier of the index set code to close, as used in the open CRSP_IND_STRUCT *indptr – pointer to index structure to deallocate; if NULL, no deallocation occurs
RETURN VALUES:	CRSP_SUCCESS: if successfully closed index set CRSP_FAIL: if error closing a file or illegal parameter
SIDE EFFECTS:	All index module files are closed, and memory allocated by them in the index structure is freed. If these are the last modules open in the database, the root is also closed. If indptr is NULL, no structure memory deallocation occurs.
PRECONDITIONS:	The crspnum and setid must be taken from a previous crsp_ind_open call.

crsp_ind_free Deallocates Memory and Reinitializes an Index Set Structure

PROTOTYPE:	int crsp_ind_free (int crspnum, int setid, CRSP_IND_STRUCT *indptr)
DESCRIPTION:	deallocates memory and reinitializes an index set structure
ARGUMENTS:	int crspnum – identifier of CRSPDB database, as returned by open int setid – identifier of the index set code to free CRSP_IND_STRUCT *indptr – pointer to index structure

RETURN VALUES:	CRSP_SUCCESS: if successfully deallocated and reset index structure, or index structure is NULL CRSP_FAIL: if error deallocating memory, error in parameters
SIDE EFFECTS:	The index structures are reset so all pointers are NULL and all settings are 0. All memory allocated to existing objects is freed. There is no effect if indptr is NULL.
PRECONDITIONS:	The crspnum must be known from a previous crsp_ind_open call. The setid is a predefined identifier for the index daily or monthly series or group set of index data previously opened with crsp_ind_open.

crsp_ind_init Initializes a CRSPAccess Database for Indexes Access

PROTOTYPE:	int crsp_ind_init (CRSP_IND_STRUCT *indptr)
DESCRIPTION:	initializes an index structure by setting all pointers to NULL and all counts to zero. Initializes CRSP internal structures if no previous initialization has been done.
ARGUMENTS:	CRSP_IND_STRUCT *indptr – pointer to the index structure to be initialized. This argument can be NULL to initialize a CRSP internal database without resetting an existing structure.
RETURN VALUES:	CRSP_SUCCESS: if index internals successfully initialized CRSP_FAIL: if error opening or reading initialization file
SIDE EFFECTS:	Internal structures will be initialized, including the array of known sets. They will be stored in internal structures in this module and used by other CRSP functions. All the pointers in the index structure indptr will be set to NULL. If a structure is already initialized with crsp_ind_open, crsp_ind_free should be used or memory will be lost.
PRECONDITIONS:	None

crsp_ind_open Opens an Existing Index Set in an Existing CRSPDB

PROTOTYPE:	int crsp_ind_open (char *root, int setid, CRSP_IND_STRUCT *indptr, int wanted, char *mode, int bufferflag)
DESCRIPTION:	opens an existing index set in an existing crspdb. This opens database files, allocates needed memory to a structure, and initializes internal structures to index data can be used. See crsp_ind_clear for clearing data

ARGUMENTS:	char *root – path of root directory. If root is NULL the CRSP_DSTK or CRSP_MSTK environment variables are used. int setid – the set identifier 400 = monthly index groups 420 = monthly index series 440 = daily index groups 460 = daily index series CRSP_IND_STRUCT *indptr – pointer to index structure to be associated with this database. If indptr is NULL then space for a CRSP_IND_STRUCT is allocated by this function. int wanted – mask indicating which modules will be used. The list below shows the wanted values for the index modules. The wanted values can be summed or summary wanted values can be used to open multiple modules. Only modules that are selected in the wanted parameter have memory allocated in the index structure and only those modules can be accessed in further access functions to the database. IND_HEAD 1 header structure and index description IND_REBALS 2 rebalancing information for index groups IND_LISTS 4 issue lists IND_USDCNTS 8 portfolio used counts IND_TOTCNTS 16 portfolio total eligible counts IND_USDVALS 32 portfolio used weights IND_TOTVALS 64 portfolio eligible weights IND_TRETURNS 128 total returns IND_ARETURNS 256 capital appreciation returns IND_IRETURNS 512 income returns IND_TLEVELS 1024 total return index levels IND_ALEVELS 2048 capital appreciation index levels IND_ILEVELS 4096 income return index levels Symbols are available for common groups of modules. IND_ALL selects all the index data. IND_INFO = IND_HEAD + IND_REBALS + IND_LISTS IND_RETURNS = IND_TRETURNS + IND_ARETURNS + IND_IRETURNS IND_LEVELS = IND_TLEVELS + IND_ALEVELS + IND_ILEVELS IND_COUNTS = IND_USDCNTS + IND_TOTCNTS + IND_USDVALS+IND_TOTVALS IND_RESULTS = IND_HEAD + IND_USDCNTS + IND_USDVALS+IND_TRETURNS IND_ARESULTS = IND_HEAD + IND_USDCNTS + IND_USDVALS + IND_ARETURNS IND_IRESULTS = IND_HEAD + IND_USDCNTS + IND_USDVALS + IND_IRETURNS IND_STD = IND_HEAD + IND_COUNTS + IND_TRETURNS + IND_ARETURNS IND_ALL = IND_INFO + IND_RETURNS + IND_LEVELS+IND_COUNTS char *mode – usage while open. Possible string values are: r = read, rw = read/write int bufferflag – level of buffering: 0 : no buffering, 1 : use default, n : use factor of default int crspnum: if opened successfully. This crspnum is used in further access functions to the database
RETURN VALUES:	int CRSP_FAIL: if error opening or loading files, if bad parameters, root already opened exclusively, index set already opened rw, wanted not a subset of set's modules, set does not exist in root, set already opened and structure allocated, error allocating memory for internal or stock structures.
SIDE EFFECTS:	This will load root and index initialization files if needed, open the root including loading the configuration structure and index structures to memory, opening the address file, and if necessary allocating memory to file buffers, loading the free list, and logging information to the log file. Files will be opened for all wanted modules. Associated calendars will be loaded if necessary. wanted index structures will be allocated.
PRECONDITIONS:	None; the root may already be open. If a new index structure is passed additional fields may be allocated.

crsp_ind_read Loads Wanted Index Data For an INDNO

PROTOTYPE:	int crsp_ind_read (int crspnum, int setid, int *key, int keyflag, CRSP_IND_STRUCT
DESCRIPTION:	loads wanted index data for an INDNO
ARGUMENTS:	int crspnum – crspdb root identifier returned by crsp_ind_open int setid – the set identifier used in crsp_ind_open int *key – specific indno of data to load int keyflag – CRSP_EXACT constant to search for the indno in *key, or positional constant: CRSP_FIRST – the first key in the database CRSP_PREV – the previous key CRSP_LAST – the last key in the database CRSP_SAME – the same key CRSP_NEXT – the next key CRSP_STK_STRUCT *indptr – structure to load data int wanted – mask of flags indicating which module data to load. See crsp_ind_open on page 127 for module codes.
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_EOF: if next or previous key at end or beginning of file CRSP_FOUND_OTHER: if key found in root, but not for this set CRSP_NOT_FOUND: if key not found in database CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum and setid, error in read, impossible wanted
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the index structure. The position used for the next positional read is reset based on the key found. If keyflag is a positional qualifier, the actual INDNO found is loaded to *key. Data is only loaded to wanted data structures within the range of valid data for the index. Use index clear functions to erase previously loaded data.
PRECONDITIONS:	The index set must be previously opened. The crspnum must be returned from a previous crsp_stk_open call. indptr must have been passed to a previous crsp_stk_open call. wanted must be a subset of the wanted parameter passed to the crsp_ind_open function.

crsp_ind_alloc Builds Indexes Set Object Lists, Allocates Memory, and Sets Pointers

PROTOTYPE:	int crsp_ind_alloc (int crspnum, int setid, CRSP_IND_STRUCT *ind, int wanted)
DESCRIPTION:	Builds Indexes set object lists, allocates memory, and sets pointers
ARGUMENTS:	int crspnum - identifier of crsp database, as returned by open int setid - identifier of the Indexes set to allocate CRSP_IND_STRUCT *ind - pointer to Indexes structure int wanted - binary code of modules wanted
RETURN VALUES:	CRSP_SUCCESS - if successfully initialized and allocated Indexes structure CRSP_FAIL - if error allocating memory, error in paramters
SIDE EFFECTS:	Three levels of pointers are allocated in the Indexes structure. 1 - object_element list elements are created for each wanted module 2 - object types are allocated for each object in wanted modules, and object level pointers are set 3 - arrays are allocated for each object, and array level pointers are set. The setcode and wanted are stored in the structure.
PRECONDITIONS:	The crspnum must be known from a previous crsp_ind_open or crsp_openroot call. The setcode is an installation-defined code for the set.

crsp_ind_copy Copy Data from One Indexes Structure to Another

PROTOTYPE:	`int crsp_ind_copy (CRSP_IND_STRUCT *indtrg, CRSP_IND_STRUCT *indsrc, int wanted)`
DESCRIPTION:	Copy data from one indexes structure to another
ARGUMENTS:	`CRSP_IND_STRUCT *indtrg` - pointer to indexes structure target `CRSP_IND_STRUCT *indsrc` - pointer to indexes structure source `int wanted` - `wanted` flag of modules to copy
RETURN VALUES:	`CRSP_SUCCESS` - if successfully copied data from source to target `CRSP_FAIL` - if incompatible structures
SIDE EFFECTS:	Data is copied from the source structure to the target structure. The `loadflag` field is used to identify all `wanted` modules to copy.
PRECONDITIONS:	Both structures must be allocated with `crsp_ind_open`. The `wanted` for the target must be a superset of the `wanted` in the source. The versions of the structure must be compatible - the source modules must not have missing objects or higher `maxarr`s that the counterparts in the target.

crsp_ind_delete Deletes a PERMNO from an Indexes Set

PROTOTYPE:	`int crsp_ind_delete (int crspnum, int setid, int key)`
DESCRIPTION:	Deletes a PERMNO from an Indexes set
ARGUMENTS:	`int crspnum` - identifier of root `int setid` - identifier of the set `int key` - PERMNO to erase from Indexes set
RETURN VALUES:	`CRSP_SUCCESS` - if successfully closed Indexes set `CRSP_FAIL` - if key not found, not found in set, not open for rewrite, or illegal parameter
SIDE EFFECTS:	If the PERMNO is found in the set it is erased. All space allocated in the module file for this PERMNO will be added to the free list for that module. If the PERMNO does not belong to any other modules, it will be erased from the index file and its address record placed on the address file free list. Otherwise, the index file module flags will be reset and the address records for these Indexes modules will be set to `NULL`
PRECONDITIONS:	The root and Indexes set must be opened previously with `crsp_ind_open`. The open must use the `rw` mode.

crsp_ind_insert Adds a new PERMNO to an Indexes Set

PROTOTYPE:	`int crsp_ind_insert (int crspnum, int setid, int key, CRSP_IND_STRUCT *indptr, int wanted)`
DESCRIPTION:	Adds a new PERMNO to an Indexes set
ARGUMENTS:	`int crspnum` - `crspdb` identifier from open `int setid` - the set identifier `int key` - PERMNO to identify the new issue `CRSP_IND_STRUCT *indptr` - pointer to Indexes structure, data that will be added to the `wanted` modules `int wanted` - an Indexes `wanted` parameter indicating which Indexes data modules include data that will be saved.
RETURN VALUES:	`CRSP_SUCCESS` - if successfully added `CRSP_FAIL` - if bad parameters, `crspdb` and indnum not open for `RW`, PERMNO already exists in this set.
SIDE EFFECTS:	Data for all modules will be added to the proper file. The free list is used to find a location in the module file and may be updated if free space is used. If the key already exists but in different sets, the index file module flag is updated, and the new module addresses and sizes are added to the address file. If the key is totally new, a new index file row and address file record are created.
PRECONDITIONS:	the Indexes set must be opened previously with `RW` mode.

crsp_ind_modload Allocates a Module Structure, Loads a Module and Objects Information

PROTOTYPE:	`int crsp_ind_modload (int crspnum, int modindex, int modid, CRSP_CONFIG_MOD *modstruct)`
DESCRIPTION:	Allocates a module structure and loads a module and objects information into it
ARGUMENTS:	`int crsp_num` - `int modindex` - the index of the module in the `CRSP_MODTYPE` structure `array. int modid` - `CRSP_CONFIG_MOD *modstruct` - pointer to the `CRSP_CONFIG_MOD` structure

RETURN VALUES:	CRSP_SUCCESS - if the module is loaded successfully CRSP_FAIL - if bad parameters or error allocating and loading module structures

crsp_ind_newset Adds a Set of Indexes Modules to a crspdb

PROTOTYPE:	int crsp_ind_newset (char *root, int setid, int wanted)
DESCRIPTION:	Adds a set of Indexes modules to a crspdb
ARGUMENTS:	char *root - path of existing crspdb root directory int setid - known Indexes set number from initialization file int wanted - mask determining which Indexes modules are supported in the set
RETURN VALUES:	CRSP_SUCCESS - if the Indexes set is added CRSP_FAIL - if bad parameters or error manipulating root structures
SIDE EFFECTS:	crsp_ind_newset adds a set of Indexes modules to an existing crspdb. It will add the information about the set to the configuration file and recreate the address file with the new modules added to each record. Empty data files for the modules will be created. If the calendars are new to the root they will be added. The new modules will be assigned to the proper calendars.
PRECONDITIONS:	The root must exist and be unopened. It is created separately with the crsp_newroot function

crsp_ind_null Function to Zero Out the Index Structure Before it is Used

PROTOTYPE:	int crsp_ind_NULL(CRSP_IND_STRUCT *indptr)
DESCRIPTION:	Function to zero out the index structure before used
ARGUMENTS:	CRSP_IND_STRUCT *indptr - pointer to stock structure
RETURN VALUES:	CRSP_SUCCESS - if stock internals successfully initialized CRSP_FAIL - if error opening or reading initialization file
SIDE EFFECTS:	The index structure will be set to zero according to the loadflag

crsp_ind_read_subset Loads Wanted Indexes Data for an INDNO Applying All Subsetting Filters

PROTOTYPE:	int crsp_ind_read_subset (int crspnum, int setid, int *key, int keyflag, CRSP_IND_STRUCT *indptr, int wanted, CRSP_IND_SUBSET_PARAMS *subpar)
DESCRIPTION:	loads wanted indexes data for an INDNO applying all subsetting filters
ARGUMENTS:	int crspnum - crspdb root identifier returned by crsp_ind_open int setid - the set identifier used in crsp_ind_open int *key - PERMNO to load int keyflag - positional qualifier or no match qualifier CRSP_IND_STRUCT *indptr - structure to load data int wanted - mask of flags indicating which module data to load CRSP_IND_SUBSET_PARAMS *subpar - pointer to structure containing subsetting flags
RETURN VALUES:	CRSP_SUCCESS - if data loaded successfully CRSP_EOF - if next or previous key at end or beginning of file CRSP_NOT_FOUND - if PERMNO not found CRSP_FAIL - if error with bad parameters, invalid or un opened crspnum and setid, error in read, impossible wanted, invalid INDNO index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the structure. The data loaded in the module buffers may be changed. The position for further reads will be set to the location of the read. Multiple INDNOs may be loaded on a positional read if subsetting totally eliminates INDNOs that otherwise would be loaded.
PRECONDITIONS:	The indexes set must be previously opened. The setid and crspnum and indptr are the same as opened and the wanted must be a subset of the wanted open. The subset parameter structure must be loaded with valid flags

`crsp_ind_update` Update Indexes Data for a Key

PROTOTYPE:	`int crsp_ind_update (int crspnum, int setid, int key, CRSP_IND_STRUCT *indptr, int wanted)`
DESCRIPTION:	Update Indexes data for a key

ARGUMENTS:	`int crspnum` - `crspdb` root identifier returned by `crsp_ind_open`
	`int setid` - the set identifier used in `crsp_ind_open`
	`int key` - specific PERMNO of data to write
	`CRSP_IND_STRUCT *indptr` - structure containing new data
	`int wanted` - mask of flags indicating which module data to write

RETURN VALUES:	`CRSP_SUCCESS` - if data written successfully `CRSP_FOUND_OTHER` - if key found in root, but not for this set `CRSP_NOT_FOUND` - if key not found in root `CRSP_FAIL` - if error with bad parameters, invalid or unopened `crspnum` and indnum for RW, error in write, impossible `wanted`
SIDE EFFECTS:	Data from the `wanted` modules will be written to the proper locations in the module files. The address file may be updated for new offsets and sizes. If the new data does not fit within the allocated space for that key in the module file the data may be moved to a new location and the free list modified. The data loaded in the module buffers may be changed.
PRECONDITIONS:	The Indexes set must be previously opened. The indnum and `crspnum` and `indptr` are the same as opened and the`wanted` must be a subset of the `wanted` open.

`crsp_ind_del_fromset` Removes Modules from Index Set from a CRSP Root

PROTOTYPE:	`int crsp_ind_del_fromset (char *root, int setid, int wanted)`
DESCRIPTION:	Removes modules from Index set from a root
ARGUMENTS:	`char *root` - path of `crspdb` root `int setid` - identifier of the set `int wanted` - binary code of modules `wanted` to delete
RETURN VALUES:	`CRSP_SUCCESS` - if the ind modules are removed successfully `CRSP_FAIL` - if something wrong
SIDE EFFECTS:	`crsp_ind_del_fromset` removes the `wanted` modules associated with a given index set from a `crspdb` root. All `wanted` modules will be erased from the address file, which will be rewritten with a new restricted record length. The index file will also be rewritten, with keys changed to new module inclusion flags or erased altogether. The configuration file will be rewritten without the modules included in the ind set. `Wanted` module files of this set will be deleted. If all modules are deleted, delete and the set.
PRECONDITIONS:	will leave an unopened root

`crsp_ind_add_toset` Add modules to an existing Indexes set

PROTOTYPE:	`int crsp_ind_add_toset (char *root, int setid, int wanted)`
DESCRIPTION:	Add modules to an existing stock set
ARGUMENTS:	`char *root` - path of `crspdb` root `int setid` - identifier of the set `int wanted` - binary code of modules `wanted`
RETURN VALUES:	`CRSP_SUCCESS` - if the stock set is removed `CRSP_FAIL` - if bad parameters

`crsp_ind_free_ind` Function to free an Index structure

DESCRIPTION:	Function to free an Index structure
ARGUMENTS:	`CRSP_IND_STRUCT *INDptr` - pointer to index structure to be freed `int free_flag` - free only the array part or all
RETURN VALUES:	`CRSP_SUCCESS` - if successfully initialized `CRSP_FAIL` - if bad parameters

GENERAL ACCESS FUNCTIONS

The CRSPAccess general access functions include error functions and portable file operation functions.

FUNCTION GROUP	DESCRIPTION	PAGE
Error-handling	The CRSPAccess function for handling errors produced by CRSP functions	page 133
C Portable File Operations	These functions call standard I/O functions in the C run time library	page 138

Error Handling

PROTOTYPE:	`int crsp_errprintf (va_list)`

DESCRIPTION:	Builds error messages using an installation-wide file of messages, and supports basic handling of the results.
	Each error message, including a mnemonic name and text description, is stored in a file in the CRSP initialization directory. A unique integer error message number is assigned to each message. The function is passed a message number, an error number, flags for type of error and handling output, plus optionally arguments on where the results are sent and variables to modify the error messages. The message description is built into an output error message. If the error is from a system call, system error messages and error numbers can be appended to this message. The message is then written to the location specified in the print flag.
	There is a generic message available to users wishing to use the crsp_errprintf functionality and there are two environment variables users can set to change the behavior of the error message function.
	`CRSP_TRACE` – can be used to modify the output behavior of the function. The default behavior is to add only the formatted string to the message output, and use the CRSP_`NULL` printflag option. If CRSP_TRACE is defined it must have one or more of the following one-letter codes in a string. Each code present changes the output. The possible codes are:
	`m =` `CRSP_MSGNUMBER` – add the message number to message output
	`e =` `CRSP_ERRNUMBER` – add the error number to message output
	`n =` `CRSP_MSGNAME` – add the message header name to message output
	`f =` `CRSP_MSGFORMAT` – do not add formatted string to message output
	`s =` `CRSP_SEVERITY` – add severity name to output
	`t =` `CRSP_ERRTYPE` – add error type to output
	`c =` `CRSP_CALLTRACE` – print call error type messages
	`o =` `CRSP_NULLOUT` – overrides `CRSP_NULL printflag` option in library functions. Print message directly to standard output and clear `errmsg`
	`r =` `CRSP_NULLERR` – overrides `CRSP_NULL printflag` option in library functions. Print message directly to standard error and clear `errmsg`
	`w =` `CRSP_NOWARN` – warning messages are ignored
	`i =` `CRSP_NOINFO` – informational messages are ignored
	`a =` `CRSP_NOFATAL` – fatal messages are ignored
	`CRSP_MSGFILE` – can be used to use messages from one or more alternate message files. If this environment variable is set to a comma-delimited set of files, the function will search these files in order for messages. The message files must have a leading line with the lowest and highest message numbers allowed in the file, followed by lines with three text fields delimited by pipes (|), containing in order a message number, a header name, and a format string compatible with the C `printf` function. The message lines must be sorted by message number. Header names are limited to 20 characters, and the format cannot produce a string of more than 500 characters. Message numbers must be 100,000 or higher to avoid possible conflict with standard CRSPAccess messages
	The standard CRSP message file is named `crsp_error_msg.dat`. It is found in the directory set by environment variables $CRSP_LIB on Unix, `%crsp_lib%` on Windows, and `CRSP_LIB`: on OpenVMS. It can contain message numbers in the range of 1 to 99,999. If an alternate message file contains a message number also in the CRSP message file, the alternate definition is used, and therefore must have a compatible format.

ARGUMENTS:	variable argument list, including: `int errnum` – number assigned to the specific error `int msgnum` – number assigned to the specific message, must be defined in the error message file `int errorflag` – flag for the type and severity of the error. It must be a constant of the following form: `CRSP_severity_type` where severity is one of: `INFO` – informational `WARN` – warning `FATAL` – fatal and `type` is one of: `USER` – error in user argument or usage `SYS` – error returned by a system or external function `CALL` – function call returned an error `PRINT` – print global error messages `int printflag` – flag for the method of handling the output. It must be one of the following constants: `CRSP_ERROUT_STDERR` – write messages directly to standard error `CRSP_ERROUT_STDOUT` – write messages directly to standard output `CRSP_ERROUT_FILE` – write messages to a file pointer (previously opened with `fopen`) given in the next argument `CRSP_ERROUT_STRING` – write messages to string given in the next argument. The string must have enough space allocated to store the message. `CRSP_NULL` – append messages to a global string err_msg. If messages extend past the length of the string, previously stored messages are printed to the screen. This is used by all CRSP library functions. `FILE * errfileptr` – optional argument present only if `printflag` is `CRSP_ERROUT_FILE`. If present, error messages are written to this file. `Errfileptr` is the file handler returned by fopen. `char * msgstring` – optional argument present only if `printflag` is `CRSP_ERROUT_STRING`. If present, error messages are copied to the string `int msgstringlen` – optional argument present only if `printflag` is `CRSP_ERROUT_STRING`. If present, `msgstringlen` is the length allocated to msgstring. ... – list of variables to be embedded in the error message. There can be zero or more variables. There must be a one to one correspondence between the number and types of variables in this list and the format string in the CRSP error message file for the specified `msgnum`.
RETURN VALUES:	`CRSP_SUCCESS`: if error handled successfully `CRSP_FAIL`: if error in parameters or in opening or reading the error message file
SIDE EFFECTS:	The `crsp_init` initialization function is called to initialize access. The `crsp_error_msg.dat` file in the initialization directory is opened the first time the function is called and closed when the program exits.
PRECONDITIONS:	CRSP functions always place errors in a global string named `err_msg`. CRSP environment variables must be set properly so the file `crsp_error_msg.dat` is found in the `CRSP_LIB` directory. See the description for optional environment variables that affect the results.

C Portable File Functions

These functions call standard I/O functions in the C run time library, but can be used on Windows, Unix, and Open- VMS systems without changes and incorporate the CRSP error handling function.

FUNCTION	DESCRIPTION	PAGE
`crsp_file_append`	generic file `append` for multiple platforms	page 135
`crsp_file_close`	generic file `close` for multiple platforms	page 135
`crsp_file_fopen`	generic file `fopen` for multiple platforms	page 135
`crsp_file_lseek`	generic file `lseek` for multiple platforms	page 135
`crsp_file_open`	generic file `open` for multiple platforms	page 136
`crsp_file_read`	generic file `read` for multiple platforms	page 129
`crsp_file_remove`	generic file delete for multiple platforms	page 137
`crsp_file_rename`	generic file rename for multiple platforms	page 137
`crsp_file_search`	generic check for existence of a file on multiple platforms	page 137
`crsp_file_stamp`	generic generation of a unique file name for multiple platforms	page 141

FUNCTION	DESCRIPTION	PAGE
crsp_file_write	generic file write for multiple platforms	page 137
crsp_free	generic memory free for multiple platforms	page 138

crsp_file_append Generic File Append for Multiple Platforms

PROTOTYPE:	int crsp_file_append (char *origfile, char *newfile)
DESCRIPTION:	append a file by adding the data from the second file at the end of the first file
ARGUMENTS:	char *origfile – pointer to the original file char *newfile – pointer to the new file to be appended to the first file
RETURN VALUES:	CRSP_SUCCESS: if appended successfully CRSP_FAIL: error in parameters or error in open or write operation
SIDE EFFECTS:	both files are opened with fopen, data from the second file is copied to the first, and then both files are closed.
PRECONDITIONS:	both files must exist and contain character data with no records 500 characters or longer.

crsp_file_close Generic File Close for Multiple Platforms

PROTOTYPE:	int crsp_file_close (int file_desc)
DESCRIPTION:	calls the C close function
ARGUMENTS:	int file_desc – file handler of file to close, as returned from open function
RETURN VALUES:	CRSP_SUCCESS: if closed successfully CRSP_FAIL: file not opened or error in close
SIDE EFFECTS:	file described by file_desc is closed
PRECONDITIONS:	file must be previously opened, with file_desc returned from open

crsp_file_fopen Generic File fopen for Multiple Platforms

PROTOTYPE:	(FILE *)crsp_file_fopen (va_alist)
DESCRIPTION:	platform-independent version of fopen with support for extra OpenVMS options
ARGUMENTS:	variable argument list: char *path – mandatory argument containing path of file to open char *mode – mandatory argument containing mode passed to fopen, "r" to open read-only, "rw" to read and write. See fopen for all options. 0-6 char *rmsflags – up to 6 optional RMS flags passed to fopen on OpenVMS systems, and ignored on other systems
RETURN VALUES:	File pointer on success NULL if error opening file
SIDE EFFECTS:	the file specified in the first parameter is opened. The default setting for OpenVMS systems is "mbc=127" unless overridden by one of the rmsflags options.
PRECONDITIONS:	See C documentation on this function for more details

`crsp_file_lseek` Generic C lseek for Multiple Platforms

PROTOTYPE:	`int crsp_file_lseek (int file_desc, int offset, int direction)`
DESCRIPTION:	Generic file `lseek` for multiple platforms. This function positions a file to an arbitrary byte position and returns the new position. Parameters are passed directly to the C `lseek` function. See C documentation on this function for more details
ARGUMENTS:	`int file_desc` – the file descriptor of the current file returned by C `open` `int offset` – the offset specified in bytes `int direction` – an integer measuring whether the offset is to be measured: forward from the beginning of the file (`direction = SEEK_SET`) forward from the current position (`direction = SEEK_CUR`) forward from the `end` of the file (`direction = SEEK_END`)
RETURN VALUES:	the new file position if successful `CRSP_FAIL`: error if file descriptor unidentified, or a seek was attempted before the beginning of the file.
SIDE EFFECTS:	the current position in the file is set for further operations
PRECONDITIONS:	file must be previously opened with the open function

`crsp_file_open` Generic C Open for Multiple Platforms

PROTOTYPE:	`int crsp_file_open (char *file_spec, int flags, unsigned int mode, int platflags, int pmode, int allocate, int mbc, int extend)`
DESCRIPTION:	Generic file open for multiple platforms. Parameters for OpenVMS, Unix, and Windows versions are passed, and only the ones needed for the current platform are passed to the C open function. See C documentation on this function for more details.
ARGUMENTS:	`char *file_spec` – character string containing a valid file specification of a file to be opened. `int flags` – flags for permitted usage of opened file `int mode` – the file protection of a new file `int platflags` – additional flags bitwise `or`'ed with `flags` if Windows, ignored if another system `int pmode` – additional protection modes `or`'ed with `mode` if Windows, ignored if another system `int allocate` – blocks to allocate for a new file on OpenVMS, ignored if another system `int mbc` – block count per I/O on OpenVMS, ignored if another system `int extend` – blocks to allocate if additional space is needed on OpenVMS, ignored if another system
RETURN VALUES:	file descriptor if opened successfully, to be used in other file operations with this file `CRSP_FAIL`: error if file could not be opened
SIDE EFFECTS:	the file is opened and the file pointer is returned for further access
PRECONDITIONS:	the file existence and protections must agree with flags and modes passed to open

`crsp_file_read` Generic C Read for Multiple Platforms

PROTOTYPE:	`int crsp_file_read (int file_desc, void *buffer, int nbytes)`
DESCRIPTION:	Generic file read for multiple platforms
ARGUMENTS:	`int file_desc` – the file descriptor of the current file returned by C `open` `void *buffer` – address of contiguous storage where data will be loaded `int nbytes` – the maximum number of bytes to read
RETURN VALUES:	the number of bytes read. The return value does not necessarily equal nbytes since the function does not read beyond the end of the file or input terminal line `CRSP_FAIL`: if error in parameters or read
SIDE EFFECTS:	the current position in the file is set to the end of the `read`
PRECONDITIONS:	file must be previously opened with the `open` function

`crsp_file_remove` Generic File Delete for Multiple Platforms

PROTOTYPE:	`int crsp_file_remove (char *file_spec)`
DESCRIPTION:	calls the C remove function to delete a file
ARGUMENTS:	`char *file_spec` – specification of file to remove
RETURN VALUES:	`CRSP_SUCCESS`: if removed successfully `CRSP_FAIL`: error in parameters or error in remove operation
SIDE EFFECTS:	file is removed
PRECONDITIONS:	file must exist and user must have delete permissions

`crsp_file_rename` Generic File Rename for Multiple Platforms

PROTOTYPE:	`int crsp_file_rename (char *old_file_spec, char *new_file_spec)`
DESCRIPTION:	Calls the C rename function to change the name of a file
ARGUMENTS:	`char *old_file_spec` – specification of the file to rename `char *new_file_spec` – new specification of the file
RETURN VALUES:	`CRSP_SUCCESS`: if renamed `CRSP_FAIL`: error in parameters or error in file operation or permissions
SIDE EFFECTS:	the file is renamed
PRECONDITIONS:	the old file must exist, the second must be a valid specification, and the rename operation must be valid on the system between the two files.

`crsp_file_search` Generic Check for the Existence of a File

PROTOTYPE:	`int crsp_file_search (char *file_spec)`
DESCRIPTION:	Checks for the existence of a file
ARGUMENTS:	`char *file_spec` – specification of file to check
RETURN VALUES:	`CRSP_SUCCESS`: if the file exists `CRSP_FAIL`: if the file does not exist or cannot be opened for read access
SIDE EFFECTS:	file is opened and closed
PRECONDITIONS:	file must have read permissions

`crsp_file_stamp` Create a Unique File Name

PROTOTYPE:	`char *crsp_file_stamp ( )`
DESCRIPTION:	Creates a string that can be built into a unique file name based on system time and user ID. The string contains the process ID returned from the C `getpid` function, an underscore, and the system time in seconds returned from the C `time` function.
ARGUMENTS:	none
RETURN VALUES:	pointer to a string with a file specification if successful
	`NULL`: if failure getting system time or user ID
SIDE EFFECTS:	memory is allocated up to 80 characters to store the new file name
PRECONDITIONS:	none

`crsp_file_write` Generic C Write for Multiple Platforms

PROTOTYPE:	`int crsp_file_write (int file_desc, void *buffer, int nbytes)`
DESCRIPTION:	Generic file write for multiple platforms
ARGUMENTS:	`int file_desc` – the file descriptor of the current file returned by C `open` `void *buffer` – address of contiguous storage where data will be retrieved `int nbytes` – the maximum number of bytes to write

RETURN VALUES:	the number of bytes written. `CRSP_FAIL`: if error in parameters or write
SIDE EFFECTS:	the current position in the file is set to the `end` of the written data
PRECONDITIONS:	file must be previously opened with the `open` function

`crsp_free` Generic Memory Free for Multiple Platforms

PROTOTYPE:	`int crsp_free (void *ptr)`
DESCRIPTION:	Calls the C `free` function
ARGUMENTS:	`void *ptr` – pointer to the memory to be freed
RETURN VALUES:	`CRSP_SUCCESS`: if successfully freed. Always true on Windows and Unix where C `free` function is void. `CRSP_FAIL`: error freeing memory on OpenVMS systems
SIDE EFFECTS:	memory pointed to by `ptr` is deallocated
PRECONDITIONS:	none

GENERAL UTILITY FUNCTIONS

The utility functions operate on the base CRSPAccess data structures and are not specific to a type of data. They include operations on calendars CRSP object structures and general utilities.

FUNCTION GROUP	DESCRIPTION	PAGE
Calendar Utility Functions	Functions used to manipulate CRSP calendars	page 145
Calendar Access Functions	Functions used to access CRSP calendars	page 141
Compare Functions	Functions used to compare data in two structures	page 151
Object Functions	Functions used to manipulate base object structures	page 145
String Functions	Functions used to manipulate character strings	page 156
C Structure Copy Functions	Functions used to copy data from one like CRSPAccess structure to another	page 152
C Structure Generic Clear Functions	Functions used to load missing data to object structures	page 154
Data to Time Series Mapping Functions	Functions used to map a subset of fields to a `CRSP_TIMESERIES`	page 174
CRSPAccess C Database Information Function	Function used to retrieve information about a database	page 155

Calendar Utility Functions

These functions are used to manipulate calendar data in CRSPAccess databases.

FUNCTION	DESCRIPTION	PAGE
`crsp_cal_datecmp`	CALDT Date Search	page 138
`crsp_cal_dt2lin`	Transforms YYYYMMDD Format into Linear Date	page 146
`crsp_cal_dt2parts`	Separates the YYYYMMDD Format into Year, Month and Day	page 145
`crsp_cal_lin2dt`	Transfers Linear Date into YYYYMMDD Format	page 139
`crsp_cal_middt`	Finds the Mid-Point Date of a Range	page 139
`crsp_cal_diffdays`	Finds the Number of Calendar Days Between two Dates	page 139
`crsp_cal_link`	Creates a Link to Map Periods of two Calendars	page 140
`crsp_cal_search`	Generic Calendar Date Search	page 140
`crsp_cal_incr`	Increments an Integer Date to the Next Date	page 148
`crsp_cal_decr`	Decrements an Integer Date to the Previous Date	page 141

`crsp_cal_datecmp CALDT` Date Search

PROTOTYPE:	`int crsp_cal_datecmp (int *calelem, int *caldates, int beg, int end, int flag)`
DESCRIPTION:	Searches for an array of `caldates` to find the matching date for `calelem` and return the array index.
ARGUMENTS:	`int *calelem` – pointer to the date in YYYYMMDD format `int *caldates` – pointer to the array of calendar dates, usually the caldt pointer in a `CRSP_CAL` structure `int beg` – Index of the first calendar date range in the first calendar dates array, usually 1 `int end` – Index of the last calendar date in the last calendar dates array, usually the ndays element of the `CRSP_CAL` structure `int flag` – flag for handling inexact matches (see crsp_cal_search)
RETURN VALUES:	index of date found if a date found according to `flag` `CRSP_FAIL`: if not acceptable match according to `flag`
SIDE EFFECTS:	none

`crsp_cal_dt2lin` Transforms YYYYMMDD Format into Linear Date

PROTOTYPE:	`int crsp_cal_dt2lin (int idate)`
DESCRIPTION:	Transforms the YYYYMMDD format of date into a linear date (number of days since 19000101)
ARGUMENTS:	`int idate` – date to be transformed
RETURN VALUES:	linear date `CRSP_FAIL`: if error
SIDE EFFECTS:	none

`crsp_cal_dt2parts` Separates the YYYYMMDD Format into Year, Month, and Day

PROTOTYPE:	`void crsp_cal_dt2parts (int idate, int *year, int *month, int *day)`
DESCRIPTION:	Separates the YYYYMMDD formatted date into year, month, day.
ARGUMENTS:	`int idate` – date to be separated `int *year` – pointer to be loaded with YYYY year `int *month` – pointer to be loaded with MM month `int *day` – pointer to be loaded with DD day
RETURN VALUES:	none

`crsp_cal_lin2dt` Transfers Linear Dates into YYYYMMDD Format

PROTOTYPE:	`int crsp_cal_lin2dt (int linear_date)`
DESCRIPTION:	Transfers the linear date (number of days since 19000101) into YYYYMMDD format date.
ARGUMENTS:	`int linear_date` – the date in linear format
RETURN VALUES:	translated YYYYMMDD date `CRSP_FAIL`: if error
SIDE EFFECTS:	None

`crsp_cal_middt` Finds the Mid-Point Date of a Range

PROTOTYPE:	`int crsp_cal_middt (int idate1, int idate2)`
DESCRIPTION:	Finds a date in the middle of first date and second date
ARGUMENTS:	`int idate1` – first date, in YYYYMMDD format `int idate2` – second date, in YYYYMMDD format
RETURN VALUES:	middt: middle date between idate1 and idate2 `CRSP_FAIL`: if error
SIDE EFFECTS:	none

crsp_cal_diffdays Finds the Number of Calendar Days Between Two Dates

PROTOTYPE:	`int crsp_cal_diffdays (int idate1, int idate2)`
DESCRIPTION:	Finds the number of days between two YYYYMMDD dates
ARGUMENTS:	`int idate1` – the first date `int idate2` – the `end` date
RETURN VALUES:	number of days `CRSP_FAIL`: if error
SIDE EFFECTS:	none

crsp_cal_link Maps from One Calendar to Another

PROTOTYPE:	`int crsp_cal_link (CRSP_CAL *calbase, CRSP_CAL *calsub, int wanted, int flag)`
DESCRIPTION:	Finds mapping between a base and subset calendar. The map will have each period in the subset calendar in terms of period index ranges of the base.
ARGUMENTS:	`CRSP_CAL *calbase` – pointer to base calendar structure `CRSP_CAL *calsub` – pointer to subset calendar structure `int wanted` – the type of calendar period identification to link in the source calendar. Possible values are: `CAL_TYPE_ID` – wanted callist `CAL_TYPE_DATE` – wanted caldt `CAL_TYPE_DATERANGE` – wanted date range `CAL_TYPE_TIME` – wanted date + time `CAL_TYPE_TIMERANGE` – wanted date range + time `int flags` – flags for mapping when subset date/range is not applicable to the base. Possible values are: `CRSP_CAL_EXACT` – (= 0) non-exact matches are not mapped `CRSP_CAL_BACK` – (= -1) if not found use previous `CRSP_CAL_NEXT` – (=-1) if not found use next
RETURN VALUES:	`CRSP_SUCCESS`: calmap successfully loaded `CRSP_FAIL`: error
SIDE EFFECTS:	This function allocates space for the `calmap CRSP_CAL` structure element. It loops through `calsub` and for each date, finds the `cal` index at `calbase` for the same date and stores them in `calsub->calmap`. The `calsub basecal` pointer is set to basecal.

crsp_cal_search Date Range Search

PROTOTYPE:	`int crsp_cal_search (CRSP_CAL *cal, int wanted, void *calelem, int flag, int rangeflag)`
DESCRIPTION:	Finds the relevant calendar index number for a given calendar period. The element type may be any of the types supported by `CRSP_CAL`. `crsp_cal_search` will use only the calendar type that matches the element type. The flag is used depending on type to handle inexact matches.

ARGUMENTS:	`CRSP_CAL *cal` – pointer to the calendar structure to search `int wanted` – type of calendar element that will be located, one of: `CAL_TYPE_ID` (1) = callist `CAL_TYPE_DATE` (2) = caldt `CAL_TYPE_DATERANGE` (4) = date range `CAL_TYPE_TIME` (8) = date and time `CAL_TYPE_TIMERANGE` (16) = date+time range `int calelem` – calendar element to find. This is a pointer to a structure that must agree with the `wanted` parameter, either an `int` for `CAL_TYPE_ID` or `CAL_TYPE_DATE` or, a `CRSP_CAL_TIME`, a `CRSP_CAL_DATERANGE`, or a `CRSP_CAL_TIMERANGE` structure. `int flag` – flag for handling inexact matches – `CRSP_CAL_EXACT (0)` – only exact matches are acceptable `CRSP_CAL_BACK (-1)` – if not found use previous `CRSP_CAL_NEXT (1)` – if not found use next `int rangeflag` – option if calendar type and elements are date or time ranges: `0` = not applicable `1` = use beginning of ranges `2` = use `end` of range `3` = use middle of beginning and `end`
RETURN VALUES:	index of date if a date found according to `flag` `CRSP_NOMATCH` if no acceptable match according to `flag` `CRSP_FAIL` if invalid `flag` or data variable
SIDE EFFECTS:	none

`crsp_cal_incr` Increments an Integer Date to the Next Date

PROTOTYPE:	`int crsp_cal_incr (int_date)`
DESCRIPTION:	Increments an integer date to the next date
ARGUMENTS:	`int date` – date increment must be in YYYYMMDD format, a 0, or 99999999.
RETURN VALUES:	The next integer date in YYYYMMDD format. If date was 0 or 99999999, that value is returned.
SIDE EFFECTS:	none

`crsp_cal_decr` Decrements an Integer Date to the Previous Date

PROTOTYPE:	`int crsp_cal_decr (int_date)`
DESCRIPTION:	Decrements an integer date to the previous date
ARGUMENTS:	`int date` - date decrement must be in YYYYMMDD format, a 0, or 99999999.
RETURN VALUES:	The previous integer date in YYYYMMDD format. If date was 0 or 99999999, that value is returned.
SIDE EFFECTS:	none

Calendar Access Functions

These functions can be used to load stock data with additional options.

FUNCTION	DESCRIPTION	PAGE
crsp_obj_copy_cal	Copy data from one CRSP calendar structure to another	page 142
crsp_obj_free_cal	Free memory allocated for a CRSP calendar structure	page 147
crsp_obj_init_cal	Initialize and allocate a CRSP calendar structure	page 143
crsp_cal_load	Load a calendar available in a CRSPAccess database	page 143

`crsp_obj_copy_cal` Copies a CRSP Calendar Structure

PROTOTYPE:	`int crsp_obj_copy_cal (CRSP_CAL *trgcal, CRSP_CAL *srccal, int caltype, int appendflag, int begind, int endind)`
DESCRIPTION:	Copies a CRSP calendar structure. Can be used to copy all calendar fields or just selected period arrays over a selected range.
ARGUMENTS:	`CRSP_CAL *trgcal` – pointer to target calendar structure to load. `CRSP_CAL *srccal` – pointer to source calendar structure. `int caltype` – integer binary code indicating which calendar types to copy. The sum of codes can be used to all copy multiple types. The codes are: `CRSP_CAL_ID` (=1) – calendar list `CRSP_CAL_DATE` (=2) – calendar dates `CRSP_CAL_DATERANGE` (=4) – calendar range `CRSP_CAL_TIME`(=8) – times `CRSP_CAL_TIMERANGE` (=16) – time range `int appendflag` – integer code determining whether to overlay new data or copy the entire structure. Valid code values are: `CRSP_COPY_RESET`. -The source calendar is copied entirely to the target. All header fields are copied directly and all calendar types selected are copied directly `CRSP_COPY_OVERLAY` – Only the period arrays selected are copied to the target calendar `int begind` – index of first calendar period to copy `int endind` – index of second calendar period to copy
RETURN VALUES:	`CRSP_SUCCESS`: if the target calendar is loaded successfully. `CRSP_FAIL`: if bad parameters on incompatible calendars.
SIDE EFFECTS:	Data are copied to the target calendar according to parameters. No memory is allocated. Calmap and callink data are not copied.
PRECONDITIONS:	Memory must be allocated for all selected `caltype` fields in the target calendar. The target `maxarr` must be greater than or equal to the source `maxarr`. If `CRSP_COPY_OVERLAY` is used and the `loadflag` is not 0, the `ndays` must agree.

`crsp_obj_free_cal` Frees a CRSP Calendar Structure

PROTOTYPE:	`int crsp_obj_free_cal (CRSP_CAL **calptr, int free_flag)`
DESCRIPTION:	Frees memory allocated for a CRSP calendar structure. Can be used to free memory allocated to period arrays or the entire structure.
ARGUMENTS:	`CRSP_CAL **calptr` – pointer to pointer to calendar pointer to free. `int free_flag` – integer code indicating which parts of the structure to free. Valid code values are: `CRSP_FREE_ARR_ONLY` (=0) – free only period arrays in the calendar structure. `CRSP_FREE_OBJ_ALL` (=1) – free all periods and the structure itself
RETURN VALUES:	`CRSP_SUCCESS`: if the desired arrays are freed successfully. `CRSP_FAIL`: if wrong structure type, error freeing memory, or invalid flags
SIDE EFFECTS:	All calendar period types allocated are freed, and `arrtype` and `maxarr` are set to 0. If the `calmap` pointer is not `NULL` it is also freed. If `free_flag` is `CRSP_OBJ_FREE_ALL`, the structure itself is freed. All freed pointers are set to `NULL`. If `calptr` is initially `NULL` the function does nothing and returns `CRSP_SUCCESS`.
PRECONDITIONS:	`calptr` must be either `NULL` or point to a pointer to a calendar structure with accurate `loadflag` settings. The `calmap` pointer must be `NULL` if never allocated. Never use this function on a calendar allocated directly with a CRSPAccess open function.

`crsp_obj_init_cal` Initializes a CRSP Calendar Structure

PROTOTYPE:	`int crsp_obj_init_cal (CRSP_CAL **calptr, int maxarr, int caltype, int initflag)`
DESCRIPTION:	Initializes a CRSP calendar structure. Can be used to allocate the structure itself, allocate calendar period type arrays, and initialize values within the structure.
ARGUMENTS:	`CRSP_CAL **calptr` – pointer to pointer to calendar structure pointer to initialize. `int maxarr` – number of periods to allocate in each calendar type array. `int caltype` – integer binary code indicating which calendar types to allocate. The sum of codes can be used to allocate multiple types. The codes are `CRSP_CAL_ID` (=1) – calendar list `CRSP_CAL_DATE` (=2) – calendar dates `CRSP_CAL_DATERANGE` (=4) – calendar range `CRSP_CAL_TIME`(=8) – times `CRSP_CAL_TIMERANGE` (=16) – time range `int initflag` – integer code determining the type of initialization. Valid code values are: `CRSP_CLEAR_INIT` (=1) – initialize all fields in the structure `CRSP_CLEAR_RANGE` (=2) – add additional calendar types to the loaded structures only
RETURN VALUES:	`CRSP_SUCCESS`: if the structure is initialized and desired arrays are allocated successfully. `CRSP_FAIL`: if bad parameters, error allocating memory, or inconsistent `maxarr`
SIDE EFFECTS:	If `calptr` is initially `NULL`, it is allocated for `maxarr` periods with all `wanted` caltypes. If `calptr` is already allocated, the behavior is determined by initflag. If `initflag` is `CRSP_CLEAR_INIT`, all fields are initialized and `wanted` caltypes are allocated. Any previous information is overwritten. If `initflag` is `CRSP_CLEAR_RANGE`, only `wanted` caltypes not already loaded are allocated. `Loadflag` is set to reflect the allocated period types.
PRECONDITIONS:	`calptr` must be either `NULL` or point to a pointer to a calendar structure with accurate `loadflag` settings.

`crsp_cal_load` Loads an Existing Calendar

PROTOTYPE:	`CRSP_CAL * crsp_cal_load(int crspnum, int calid, int loadflag)`
DESCRIPTION:	Returns a pointer to a CRSPAccess calendar available in a database. The database must be previously opened with one of the `crsp_stk_open`, or `crsp_ind_open`, or `crsp_cst_open` functions. All time series accessed in a set automatically have their matching calendars loaded, so this function is only needed to access a calendar not already available in the set.
ARGUMENTS:	`int crspnum` – database handle returned by a CRSPAccess open function `int calid` – identifier of the calendar. Currently available calendars are: `100 (CRSP_CALID_DAILY)` = CRSP Daily Stock Calendar `101 (CRSP_CALID_MONTHLY)` = CRSP Monthly Stock Calendar `300 (CRSP_CALID_ANNUAL)` = CRSP Annual Stock Calendar `310 (CRSP_CALID_QUARTERLY)` = CRSP Quarterly Stock Calendar `500 (CRSP_CALID_WEEKLY)` = CRSP Weekly Stock Calendar `int loadflag` – the types of calendar period data to load. Values can be added to load multiple types: `1 (CAL_TYPE_ID)` = Calendar ID Lists `2 (CAL_TYPE_DATE)` = Calendar Dates (yyyymmdd) `4 (CAL_TYPE_DATERNG` = Calendar Date Ranges `8 (CAL_TYPE_TIME)` = Calendar Date and Time `16 (CAL_TYPE_TIMERNG)` = Calendar Date and Time Ranges
RETURN VALUES:	A pointer to a loaded calendar: if successful. The calendar found is shared by all time series of that frequency in the database. If changing values in the calendar, use `crsp_obj_init_cal` and `crsp_obj_copy_cal` to make a local copy. `NULL`: if bad parameter, unopened database, or unknown `calid`
SIDE EFFECTS:	The calendar header data and requested calendar period arrays are allocated and loaded only if the calendar is not loaded already. `Loadflag` in the calendar structure is changed if additional data is loaded.
PRECONDITIONS:	The database must be opened with a CRSPAccess open function and the `calid` must be present in the database.

Compare Functions

These functions are used to compare data.

`crsp_cmp_int` Compares Two Integers

PROTOTYPE:	`int crsp_cmp_int(const void *elem1, const void *elem2)`
DESCRIPTION:	Compares two integers. Can be used as input functions to C search and sort functions.
ARGUMENTS:	`const void* – elem1` – pointer to the first element `const void* – elem2` – pointer to the second element
RETURN VALUES:	`int: <0 if elem1 < elem2, 0 if elem1 = elem2,>1 if elem1 > elem2`. Based on standard integer comparisons

`crsp_cmp_string` Compares Two Strings

PROTOTYPE:	`int crsp_cmp_string(const void *elem1, const void *elem2)`
DESCRIPTION:	Compares two strings. Can be used as input functions to C search and sort functions.
ARGUMENTS:	`const void* – elem1` – pointer to the first terminated string
	`const void* – elem2` – pointer to the second terminated string
RETURN VALUES:	`int: <0 if elem1 < elem2, 0 if elem1 = elem2, >1 if elem1 > elem2`. Based on standard string comparisons

CRSP Object Functions

These functions are used to manipulate base CRSPAccess object structures.

FUNCTION	DESCRIPTION	PAGE
`crsp_obj_verify_ts`	Verifies a CRSP Time Series Object	page 144
`crsp_obj_verify_arr`	Verifies a CRSP Array Object	page 145
`crsp_obj_verify_row`	Verifies a CRSP Row Object	page 145
`crsp_obj_init_ts`	Initializes a CRSP Time Series Object	page 150
`crsp_obj_init_arr`	Initializes a CRSP Array Object	page 146
`crsp_obj_init_row`	Initializes a CRSP Row Object	page 146
`crsp_obj_comp_ts`	Compares two CRSP Time Series Objects	page 151
`crsp_obj_comp_arr`	Compares two CRSP Array Objects	page 151
`crsp_obj_comp_row`	Compares two CRSP Row Objects	page 147
`crsp_obj_free_ts`	Frees a CRSP Time Series Object	page 147
`crsp_obj_free_arr`	Frees a CRSP Array Object	page 155
`crsp_obj_free_row`	Frees a CRSP Row Object	page 148
`crsp_obj_free`	Frees a CRSP Object Element Link List	page 148

`crsp_obj_verify_ts` Verifies a CRSP Time Series Object

PROTOTYPE:	`int crsp_obj_verify_ts(CRSP_TIMESERIES *ptr, int arrtype, int subtype, int maxarr, int caltypes)`
DESCRIPTION:	Verifies a time series object, by comparing array type, size, calendar, and data characteristics against expected values
ARGUMENTS:	`CRSP_TIMESERIES *ptr` – pointer to a CRSP time series object `int arrtype` – constant for the structure type of the array in the object arr. Constants are defined in crsp_const.h `int subtype` – constant for the subcategory of data in the array. Constants are defined in crsp_const.h `int maxarr` – maximum elements in the array `int caltype` – expected calendar type in the time series

RETURN VALUES:	CRSP_SUCCESS: if verification is correct 1251: object type does not verify in CRSP_TIMESERIES structures 1252: array type does not verify in time series 1253: subtype does not verify in time series 1254: maxarr does not verify in time series 1255: caltype does not verify in time series 1256: beg and end do not verify in time series 1257: end cannot be greater than maxarr in time series 1 258: cal pointer cannot be NULL in time series 1259: ndays cannot be greater than maxarr in time series 1260: arr pointer cannot be NULL in time series

crsp_obj_verify_arr Verifies a CRSP Array Object

PROTOTYPE:	int crsp_obj_verify_arr (CRSP_ARRAY *crsp_array_ptr, int arrtype, int subtype, int maxarr)
DESCRIPTION:	Verifies a CRSP array object, by comparing array type, size, and data characteristics against expected values
ARGUMENTS:	CRSP_ARRAY *crsp_array_ptr – pointer to a CRSP array object int arrtype – constant for the structure type of the array in the object arr. Constants are defined in *crsp_const.h* int subtype – constant for the subcategory of data in the array. Constants are defined in *crsp_const.h* int maxarr – maximum elements in the array
RETURN VALUES:	CRSP_SUCCESS: if verification is correct 1271:object type does not verify in CRSP_ARRAY 1272: array type does not verify in CRSP_ARRAY 1273: subtype does not verify in CRSP_ARRAY 1274: maxarr does not verify in CRSP_ARRAY 1275: num cannot be greater than maxarr in CRSP_ARRAY 1276: arr pointer cannot be NULL in CRSP_ARRAY

crsp_obj_verify_row Verifies a CRSP Row Object

PROTOTYPE:	int crsp_obj_verify_row (CRSP_ROW *crsp_row_ptr, int arrtype, int subtype)
DESCRIPTION:	Verifies a CRSP row object, by comparing array type and data characteristics against expected values
ARGUMENTS:	CRSP_ROW *crsp_row_ptr – pointer to a CRSP row object to verify int arrtype – constant for the structure type of the array in the object arr. Constants are defined in *crsp_const.h* int subtype – constant for the subcategory of data in the array. Constants are defined in *crsp_const.h*
RETURN VALUES:	CRSP_SUCCESS: if verification is correct 1282: object type does not verify in CRSP_ROW 1283: array type does not verify in CRSP_ROW 1284: subtype does not verify in CRSP_ROW 1285: arr pointer cannot be NULL in CRSP_ROW

crsp_obj_init_ts Initializes a CRSP Time Series Object

PROTOTYPE:	int crsp_obj_init_ts (CRSP_TIMESERIES **crsp_timser_ptr, int arrtype, int subtype, int maxarr, int caltype, int size_of_array, CRSP_CAL *calptr, void *init_ptr
DESCRIPTION:	Initializes a time series object. If the crsp_timeser_ptr pointer passed is NULL, the function allocates space for the object. If the array within the object is not allocated, the function allocates space for the array. Object header values are set and the calendar is attached to the time series. Each element in the object's array is initialized with the value in init_ptr.

ARGUMENTS:	CRSP_TIMESERIES **crsp_timser_ptr – pointer to a CRSP time series pointer int arrtype – constant for the structure type of the array in the object arr. Constants are defined in *crsp_const.h* int subtype – constant for the subcategory of data in the array. Constants are defined in *crsp_const.h* int maxarr – maximum elements in the array int caltype – calendar type to allocate, =2 for caldts int size_of_array – size of the structure for each array element CRSP_CAL *calptr – pointer to a calendar that will be attached to the time series object void *init_ptr – a pointer to a structure of size size_of_array with missing values to load to each element in the array. Can be NULL.
RETURN VALUES:	CRSP_SUCCESS: if successfully initialized and space allocated CRSP_FAIL: if error allocating memory, error in parameters

crsp_obj_init_arr Initializes a CRSP Array Object

PROTOTYPE:	int crsp_obj_init_arr(CRSP_ARRAY **crsp_array_ptr, int arrtype, int subtype, int maxarr, int size_of_array, void *init_ptr)
DESCRIPTION:	Initializes an array object. If the crsp_array_ptr pointer passed is NULL, the function allocates space for the object. If the array within the object is not allocated, the function allocates space for the array. Object header values are set and each element in the object's array is initialized with the value in init_ptr.
ARGUMENTS:	CRSP_ARRAY **crsp_array_ptr – pointer to a CRSP array structure pointer int arrtype – constant for the structure type of the array in the object arr. Constants are defined in *crsp_const.h* int subtype – constant for the subcategory of data in the array. Constants are defined in *crsp_const.h* int maxarr – maximum elements in the array int size_of_array – size of the structure for each array element void *init_ptr – a pointer to a structure of size size_of_array with missing values to load to each element in the array. Can be NULL.
RETURN VALUES:	CRSP_SUCCESS: if successfully initialized and space allocated CRSP_FAIL: if error allocating memory, error in parameters

crsp_obj_init_row Initializes a CRSP Row Object

PROTOTYPE:	int crsp_obj_init_row(CRSP_ROW **crsp_row_ptr, int arrtype, int subtype, int size_of_array, void *init_ptr)
DESCRIPTION:	Initializes a row object. If the crsp_row_ptr pointer passed is NULL, the function allocates space for the object. If the array within the object is not allocated, the function allocates space for the array. Object header values are set and the object's array element is initialized with the value in init_ptr.
ARGUMENTS:	CRSP_ROW **crsp_row_ptr – pointer to a CRSP row pointer int arrtype – constant for the structure type of the array in the object arr. Constants are defined in crsp_const.h int subtype – constant for the subcategory of data in the array. Constants are defined in crsp_const.h int size_of_array – size of the structure for the array element void *init_ptr – a pointer to a structure of size size_of_array with missing values to load to the row. Can be NULL.
RETURN VALUES:	CRSP_SUCCESS: if successfully initialized and space allocated CRSP_FAIL: if error allocating memory, error in parameters

crsp_obj_comp_ts Compares Two CRSP Time Series Objects

PROTOTYPE:	int crsp_obj_comp_ts(CRSP_TIMESERIES *crsp_timser_ptr1, CRSP_TIMESERIES *crsp_timser_ptr2)
DESCRIPTION:	Compares two time series objects, by comparing array types, data characteristics, array sizes, and associated calendars
ARGUMENTS:	CRSP_TIMESERIES *crsp_timser_ptr1 CRSP_TIMESERIES *crsp_timser_ptr2

RETURN VALUES:	CRSP_SUCCESS: if comparison is correct 1261: the two CRSP_TIMESERIES have different object types 1262: the two CRSP_TIMESERIES have different array types 1263: the two CRSP_TIMESERIES have different subtypes 1264: the two CRSP_TIMESERIES have different array widths 1265: the two CRSP_TIMESERIES have different maximum arrays 1266: the two CRSP_TIMESERIES have different calendar types 1267: the two CRSP_TIMESERIES calendar pointers do not compare

crsp_obj_comp_arr Compares Two CRSP Array Objects

PROTOTYPE:	int crsp_obj_comp_arr(CRSP_ARRAY *crsp_array_ptr1, CRSP_ARRAY *crsp_array_ptr2)
DESCRIPTION:	Compares two CRSP_ARRAY objects, by comparing data array type, size, and data characteristics
ARGUMENTS:	CRSP_ARRAY *crsp_array_ptr1 CRSP_ARRAY *crsp_array_ptr2
RETURN VALUES:	CRSP_SUCCESS: if array objects match 1277: the two CRSP_ARRAYs have different object types 1278: the two CRSP_ARRAYs have different array types 1279: the two CRSP_ARRAYs have different subtypes 1280: the two CRSP_ARRAYs have different array widths 1281: the two CRSP_ARRAYs have different maximum array types

crsp_obj_comp_row Compares Two CRSP Row Objects

PROTOTYPE:	int crsp_obj_comp_row(CRSP_ROW *crsp_row_ptr1, CRSP_ROW *crsp_row_ptr2)
DESCRIPTION:	Compares two CRSP row objects, by comparing data array type and data characteristics
ARGUMENTS:	CRSP_ROW *crsp_row_ptr1 CRSP_ROW *crsp_row_ptr2
RETURN VALUES:	CRSP_SUCCESS: if row objects match 1286: The two CRSP_ROW objects have different object types 1287: The two CRSP_ROW objects have different array types 1288: The two CRSP_ROW objects have different subtypes 1289: The two CRSP_ROW objects have different array widths

crsp_obj_free_ts Frees a CRSP Time Series Object

PROTOTYPE:	int crsp_obj_free_ts (CRSP_TIMESERIES **crsp timeser_ptr, int free_flag)
DESCRIPTION:	Frees a CRSP time series object by deallocating memory for just the data array or the entire object
ARGUMENTS:	CRSP_TIMESERIES **crsp_timser_ptr – points to a CRSP time series object pointer int free_flag – frees only the arr part or all. Valid values to be freed are: CRSP_FREE_ARR_ONLY CRSP_FREE_OBJ_ALL
RETURN VALUES:	CRSP_SUCCESS: if free is successful CRSP_FAIL: if error freeing memory or bad pointer or flag
SIDE EFFECTS:	Frees part or whole of the CRSP_TIMESERIES, depending on the free_flag set.

crsp_obj_free_arr Frees a CRSP Array Object

PROTOTYPE:	`int crsp_obj_free_arr(CRSP_ARRAY **crsp_array_ptr, int free_flag)`
DESCRIPTION:	Frees a CRSP array object by deallocating memory for just the data array or the entire object

ARGUMENTS:	`CRSP_ARRAY **crsp_array_ptr` – points to a CRSP array object pointer
	`int free_flag` – frees only the arr part or all. Valid values to be freed are:
	`CRSP_FREE_ARR_ONLY`
	`CRSP_FREE_OBJ_ALL`

RETURN VALUES:	`CRSP_SUCCESS`: if free is successful `CRSP_FAIL`: if error freeing memory or bad pointer or flag
SIDE EFFECTS:	Frees part or whole of the `CRSP_ARRAY`, depending on the `free_flag` set.

crsp_obj_free_row Frees a CRSP Row Object

PROTOTYPE:	`int crsp_obj_free_row( CRSP_ROW **crsp_row_ptr, int free_flag )`
DESCRIPTION:	Frees a CRSP row object by deallocating memory for just the data array or the entire object

ARGUMENTS:	`CRSP_ROW **crsp_row_ptr` – points to a CRSP row object pointer
	`int free_flag` – frees only the `arr` part or all. Valid values to be freed are:
	`CRSP_FREE_ARR_ONLY`
	`CRSP_FREE_OBJ_ALL`

RETURN VALUES:	`CRSP_SUCCESS`: if free is successful `CRSP_FAIL`: if error freeing memory or bad pointer or flag
SIDE EFFECTS:	Frees part or whole of the `CRSP_ROW`, depending on the `free_flag` set.

crsp_obj_free Frees a CRSP Object Element Link List

PROTOTYPE:	`CRSP_OBJECT_ELEMENT *objlist`
DESCRIPTION:	Frees a CRSP object element link list.
ARGUMENTS:	`CRSP_OBJECT_ELEMENT *objlist` - object element list pointer
RETURN VALUES:	`CRSP_SUCCESS`: if free is successful `CRSP_FAIL`: if free fails

String Functions

These functions can be used to manipulate strings.

FUNCTION	DESCRIPTION	PAGE
`crsp_util_convtype`	Converts CRSP Constant Names to Integers	page 153
`crsp_util_lowercase`	Converts Strings to All Lowercase Letters	page 149
`crsp_util_strtrim`	Removes Trailing Blanks from Strings	page 157
`crsp_util_uppercase`	Converts Strings to All Uppercase Letters	page 149
`crsp_util_squeeze`	Removes White Space from Character Strings	page 149
`crsp_util_strtoken`	Locates the First Delimiter in a String	page 150
`crsp_util_cvt_date_mmddyy_i`	Converts Character Date String YYMMDD into a Y-2K Compliant Date	page 150
`crsp_util_cvt_t_i`	Converts a Text String to an Integer	page 150
`crsp_util_cvt_t_l`	Converts a Numeric Text String into a Long Integer	page 150
`crsp_util_cvt_t_f`	Converts a Text String to a Floating Point Number	page 164
`crsp_util_cvt_t_d`	Converts a Text String to a Double Floating Point Number	page 157
`crsp_util_cvt_cdate_i`	Convert a Character Date String into an Integer	page 154

FUNCTION	DESCRIPTION	PAGE
crsp_util_cvt_i_cdate	Convert Integer Date to Character Date String	page 158
crsp_util_cvt_i_ingdate	Convert an Integer Date (YYYYMMDD) into an Date-Field-Compatible Character String Date	page 152

crsp_util_convtype Converts CRSP Constant Names to Integers

PROTOTYPE:	int crsp_util_convtype (char *typestring)
DESCRIPTION:	converts a CRSP constant name string to an integer. All CRSP defined _NUM constants defined in *crsp_const.h* are supported.
ARGUMENTS:	char * typename – string to convert
RETURN VALUES:	integer code found CRSP_FAIL: if string not supported
SIDE EFFECTS:	none
PRECONDITIONS:	none

crsp_util_lowercase Converts Strings to All Lowercase Letters

PROTOTYPE:	void crsp_util_lowercase (char *string)
DESCRIPTION:	converts a string to all lowercase letters.
ARGUMENTS:	char *string – string to convert
RETURN VALUES:	none
SIDE EFFECTS:	string may be changed. If the string is a string of spaces, the routine leaves one leading space.
PRECONDITIONS:	string must be a NULL-terminated character string

crsp_util_strtrim Removes Trailing Blanks From Strings

PROTOTYPE:	void crsp_util_strtrim (char *string)
DESCRIPTION:	converts a string by moving the string termination to after the last nonblank character.
ARGUMENTS:	char *string – string to convert
RETURN VALUES:	none
SIDE EFFECTS:	string may be changed
PRECONDITIONS:	string must be a NULL-terminated character string

crsp_util_uppercase Converts Strings to All Uppercase Letters

PROTOTYPE:	void crsp_util_uppercase (char *string)
DESCRIPTION:	converts a string to all uppercase letters.
ARGUMENTS:	char *string – string to convert
RETURN VALUES:	none
SIDE EFFECTS:	string may be changed
PRECONDITIONS:	string must be a NULL-terminated character string

`crsp_util_squeeze` Removes White Space from Character Strings

PROTOTYPE:	`int crsp_stk_clear (CRSP_STK_STRUCT *stk, int clearflag)`
DESCRIPTION:	converts a string by removing white space. All leading and trailing tabs or spaces are removed, and multiple tabs and spaces are replaced with a single space.
ARGUMENTS:	`char *string` – string to convert `int clearflag` – constant identifying the level of clearing. Supported values are: `CRSP_CLEAR_INIT` – only reset num `CRSP_CLEAR_ALL` – set num to 0 and set missing values for all elements in the object arrays `CRSP_CLEAR_RANGE` – set missing values for all array elements between 0 and num-1 `CRSP_CLEAR_SET` – set ranges in the `maxarr-1'th` element of the `CRSP_ARRAY` to missing values specific to the array type.
RETURN VALUES:	none
SIDE EFFECTS:	`string` may be changed
PRECONDITIONS:	`string` must be a `NULL`-terminated character string
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to a stock structure pointer to be cleared

`crsp_util_strtoken` Locates the First Delimiter in a String

PROTOTYPE:	`char * crsp_util_strtoken(char *ptr, char *delimiters)`
DESCRIPTION:	Locate the first delimiter in a string. Find the first terminator character, replace that character with a `NULL` and update the pointer to the remaining string. Unlike the standard library function, strtok, this function can handle consecutive delimiters.
ARGUMENTS:	`char *ptr` - string to parse
	`char *delimiters` - delimiter characters in a string
RETURN VALUES:	pointer to the remainder of the string, or `NULL` if no delimiter character was found
SIDE EFFECTS:	`string` may be changed
PRECONDITIONS:	`string`s must be `NULL` terminated character strings

`crsp_util_cvt_date_mmddyy_i` Converts Character Date String YYMMDD into a Y-2K Compliant Date

PROTOTYPE:	`int crsp_util_cvt_date_mmddyy_i(char *text_ptr, int *date_value)`
DESCRIPTION:	Converts a character date string of the format YYMMDD into a year 2000 compliant integer based on a 1950 cutoff. Year values < 50 are assumed to be +2000.
ARGUMENTS:	`char *text_ptr` - string to convert `int *date_value` - pointer to location into which will be put the integer value
RETURN VALUES:	`CRSP_SUCCESS` Normal successful completion `CRSP_FAIL` One or more fields could not be converted to integer values
SIDE EFFECTS:	date value is loaded

`crsp_util_cvt_t_i_` Converts a Text String to an Integer

PROTOTYPE:	`int crsp_util_cvt_t_i(char *text, int text_size, int *output)`
DESCRIPTION:	Convert a text string to an integer. Cannot convert a string larger than 11 characters. The string is assumed to NOT be `NULL` terminated.
ARGUMENTS:	`char *text` - Pointer to integer string `int text_size` - Number of digits to convert `int *output` - Pointer to location into which is written the results of the conversion
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion Anything else - system error, no value `ERANGE` - Number is too big to convert
SIDE EFFECTS:	none

`crsp_util_cvt_t_l` Converts a Numeric Text String into a Long Integer

PROTOTYPE:	`int crsp_util_cvt_t_l(char *text, int text_size, long *output)`
DESCRIPTION:	Converts a numeric text string into a long integer
ARGUMENTS:	`char *text` - Pointer to integer string `int text_size` - Number of digits to convert
	`int *output` - Pointer to long integer location into which is written the results of the conversion
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion Anything else - system error, no value `ERANGE` - Number is too big to convert
SIDE EFFECTS:	output is loaded
PRECONDITIONS:	Text string must be terminated numeric value. Output must minimally point to size (long) bytes of accessible memory.

`crsp_util_cvt_t_f_` Converts a Text String to a Floating Point Number

PROTOTYPE:	`int crsp_util_cvt_t_f(char *text, int text_size, int precision, float *output)`
DESCRIPTION:	Converts a text string to a floating point number. The string is assumed to not be `NULL`-terminated and to contain no decimal points. This routine does not handle scientific notation.
ARGUMENTS:	`char *text` - Pointer to character string to be converted `int text_size` - Number of characters in the string `int precision` - Number of characters to the right of the implied decimal point `float *output` - Pointer to floating point variable where the results of the conversion are written
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion Other - system error, no value
SIDE EFFECTS:	Output is loaded
PRECONDITIONS:	Output must point to at least size of (float) bytes of accessible memory.

`crsp_util_cvt_t_d_` Converts a Text String to a Double Floating Point Number

PROTOTYPE:	`int crsp_util_cvt_t_d(char *text, int text_size, int precision, double *output)`
DESCRIPTION:	Converts a text string to a double precision floating point number. The string is assumed to not be to `NULL` terminated and contain no decimal points. This routine does not handle scientific notation.
ARGUMENTS:	`char *text` - Pointer to character string to be converted `int text_size` - Number of characters in the string `int precision` - Number of characters to the right of the implied decimal point `float *output` - Pointer to floating point variable where the results of the conversion are written
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion Other - system error, no value
SIDE EFFECTS:	Output is loaded.
PRECONDITIONS:	Output must point to at least size (double) bytes of accessible memory.

`crsp_util_cvt_cdate_i` Convert a Character Date String into an Integer

PROTOTYPE:	`int crsp_util_cvt_cdate_i(char *date_str, int *date_int)`
DESCRIPTION:	Convert a character date string into an integer. Date format: “Mon May 19 18:05:12 1996” Integer format:19960519
ARGUMENTS:	`char *date_str` - Pointer to the `NULL` terminated string to be converted
	`int *date_int` - Pointer to the integer into which is written the converted date value
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion
	`CRSP_FAIL` - Conversion failed. Character string was possibly not the valid date format

SIDE EFFECTS:	`String` may be changed.
PRECONDITIONS:	`String` must be `NULL` terminated character string. Date-integer must point to at least size of (integer) bytes of accessible memory.

`crsp_util_cvt_i_cdate` Converts an Integer Date to a Character Date String

PROTOTYPE:	`int crsp_util_cvt_i_cdate(int int_date, char *char_buffer)`
DESCRIPTION:	Convert integer date to character date string Integer format: 19960519 Date format: "Mon May 19 18:05:12 1996"
ARGUMENTS:	`int int_date` - Value to be converted to date string `char *char_buffer` - Pointer to the character buffer into which is written the converted text string
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion `CRSP_FAIL` - Conversion failed. Integer value was possibly not a valid date
SIDE EFFECTS:	Character buffer is loaded.
PRECONDITIONS:	Character buffer must point to at least 25 bytes of accessible memory.

`crsp_util_cvt_i_ingdate` Convert an Integer Date (YYYYMMDD) into an Date-Field-Compatible Character String Date

PROTOTYPE:	`int crsp_util_cvt_i_ingdate(int date_int, char *date_str)`
DESCRIPTION:	Convert an integer date (YYYYMMDD) into a date field compatible character string date. Note: Integer value '99999999' converted to 31-dec-2299 Integer value '0' converted to blank
ARGUMENTS:	`int date_int` Integer date to be converted to character string `char *date_str` - Pointer to character string where new date is output
RETURN VALUES:	`CRSP_SUCCESS` - Normal successful completion `CRSP_FAIL` - Conversion failed. Integer value was possibly not a valid date
SIDE EFFECTS:	Date string is loaded with resultant string.
PRECONDITIONS:	Date string must point to at least 12 bytes of accessible memory.

C Structure Copy Functions

These functions are used to copy data from one like CRSPAccess structure to another.

FUNCTION	DESCRIPTION	PAGE
`crsp_util_copy_ts`	Copy Time Series Data to Another Time Series	page 152
`crsp_util_copy_arr`	Copy CRSP Array Data to Another CRSP Array	page 153
`crsp_util_copy_cal2ts`	Copy a Calendar to a Time Series	page 153

`crsp_util_copy_ts` Copy Time Series Data in a Given Range to Another Time Series

PROTOTYPE:	`int crsp_util_copy_ts(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES **trg_ts, int beg, int end, int appendflag)`
DESCRIPTION:	Copy time series data in a given range to another time series. Copies data from one time series to another within a given range. It is optional whether to overlay the source data on top of existing target data or replace the target with only the source data in the range.

ARGUMENTS:	CRSP_TIMESERIES *src_ts - pointer to existing source time series CRSP_TIMESERIES **trg_ts - pointer to pointer to target time series to be loaded. This pointer can be changed to point to a new time series if it is NULL or a different time series type int beg - beginning index to copy from source time series int end - ending index to copy from source time series int appendflag - option on whether to overlay or reset the target time series. Possible values include: CRSP_COPY_RESET - The target time series is reset. If NULL, it is initialized, and if not NULL but a different time series from the source, it is freed and re-initialized. Beg and end will become the new beg and end for the target, and data for that range will be copied from the source. CRSP_COPY_OVERLAY - The source data in the range is overlaid on top of the existing target time series. The target time series must be allocated and must compare to the source time series. The new data in the range is copied into the target, the ranges are changed accordingly.
RETURN VALUES:	CRSP_SUCCESS - if successfully CRSP_FAIL - if bad parameter, mismatched time series on overlay, unable to initialize target if needed
SIDE EFFECTS:	The target time series is initialized if different from source and loaded with data in the range copied from the source time series. The 0'th array element of the source time series (assumed containing the missing value for that time series) is copied to the target time series.
PRECONDITIONS:	The source time series must exist and have beg and end such that beg >= source time series beg, end <= source time series end, and beg >= end. If appendflag is CRSP_COPY_OVERLAY, the target time series must be allocated and compare with the source time series.

crsp_util_copy_arr Copy CRSP Array Data to Another CRSP Array

PROTOTYPE:	int crsp_util_copy_arr(CRSP_ARRAY *src_arr, CRSP_ARRAY *trg_arr)
DESCRIPTION:	Copy CRSP array data to another CRSP array
ARGUMENTS:	CRSP_ARRAY *src_arr - pointer to an existing source CRSP array CRSP_ARRAY *trg_arr - pointer to an existing target CRSP array
RETURN VALUES:	CRSP_SUCCESS - if successful CRSP_FAIL - if bad parameter, mismatched time series on overlay, unable to initialize target if needed
SIDE EFFECTS:	The source CRSP array is copied to target CRSP array. All data in array is copied and target num is set.
PRECONDITIONS:	The source and target CRSP array must exist and must be compatible.

crsp_util_copy_cal2ts Copy a Calendar to a Time Series

PROTOTYPE:	int crsp_util_copy_cal2ts(CRSP_CONFIG_CAL *cal, CRSP_TIMESERIES **ts, int cal_type)
DESCRIPTION:	Copy a calendar to a time shares
ARGUMENTS:	CRSP_CONFIG_CAL *cal - pointer to a calendar in the internal config structure. CRSP_TIMESERIES **ts - pointer to a pointer to CRSP_TIMERSERIES to store the result in the internal CRSP_ARRAY config[crspnum]->cal or equivalent. Time Series will be initialized if NULL. int caltype - determines which calendar new is copied. It must be one of: CAL_TYPE_ID - copy callist array CAL_TYPE_DATE - copy caldt array CAL_TYPE_DATERANGE - copy date range array CAL_TYPE_TIME - copy time array CAL_TYPE_TIMERANGE - copy time range array
RETURN VALUES:	CRSP_SUCCESS - if successful CRSP_FAIL - if failure
SIDE EFFECTS:	Time series will be allocated if necessary. See crsp_obj_init_ts on page 150 for expected allocation.
PRECONDITIONS:	Database must be opened with crsp_openroot or one of the crsp_*_open functions. If initialized, time series must be NULL.

C Structure Generic Clear Functions

These functions are used to load missing data to CRSPAccess object structures. CRSPAccess access functions may be used when the set type is not known ahead of time.

FUNCTION	DESCRIPTION	PAGE
crsp_util_clear_arr	Sets a CRSP_ARRAY to missing values	page 154
crsp_util_clear_elem	Sets one structure to missing values	page 154
crsp_util_clear_row	Sets a CRSP_ROW to missing values	page 155
crsp_util_clear_ts	Sets a CRSP_TIMESERIES to missing values	page 155
crsp_util_delete_ts	Deletes ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 155
crsp_util_insert_ts	Inserts ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 172
crsp_util_update_ts	Updates ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 158
crsp_util_is_missing	Handle missing value problem in CRSP_TIMESERIES structure parameters	page 163
crsp_util_reset_enddts	Resets end date for an array structure	page 164
crsp_util_merge_arr	Merges two array structures to a third, single array	page 159
crsp_util_merge_ts	Merges two time series to a third, single time series	page 160

crsp_util_clear_arr Load Missing Values to an Array

PROTOTYPE:	int crsp_util_clear_arr (CRSP_ARRAY *arr, int clearflag)
DESCRIPTION:	Loads missing values into a CRSP_ARRAY on a range level or array level.
ARGUMENTS:	CRSP_ARRAY *arr – pointer to a CRSP_ARRAY int clearflag – constant identifying the level of clearing. Supported values are: CRSP_CLEAR_INIT – only reset num to 0 CRSP_CLEAR_ALL – set num to 0 and set missing values for all elements in the object arrays CRSP_CLEAR_RANGE – set missing values for elements between 0 and num-1 CRSP_CLEAR_SET – set ranges in the maxarr-1'th element of the CRSP_ARRAY to missing values specific to the array type.
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters
SIDE EFFECTS:	The array pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT only num is set to 0. If clearflag is CRSP_CLEAR_RANGE all elements between 0 and num-1 are set to missing values. If clearflag is CRSP_CLEAR_ALL num is set to 0 and missing values are set for all elements in the object arrays. If clearflag is CRSP_CLEAR_SET, the maxarr-1'th element of the array is set to the missing value for the arrtype and subtype.
PRECONDITIONS:	The array pointer must be NULL or initialized with a valid arrtype and subtype.

crsp_util_clear_elem Load Missing Values to One Array Element or Structure

PROTOTYPE:	int crsp_util_clear_elem (void *elem, int arrtype, int subtype)
DESCRIPTION:	Loads missing values into one structure identified by array type and subtype.
ARGUMENTS:	void *elem – pointer to structure to be loaded with missing values int arrtype – integer code identifying the structure or simple data type of the element int subtype – integer code identifying the subcategory of data loaded in the element
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters or unknown arrtype or subtype
SIDE EFFECTS:	The proper missing values are loaded to the element
PRECONDITIONS:	arrtype and subtype must be valid

crsp_util_clear_row Load Missing Values to a Row

PROTOTYPE:	int crsp_util_clear_row (CRSP_ROW *row, int clearflag)
DESCRIPTION:	Loads missing values into a CRSP_ROW
ARGUMENTS:	CRSP_ROW *row – pointer to a CRSP_ROW. int clearflag – constant identifying the level of clearing. Supported values are: CRSP_CLEAR_INIT – just return success CRSP_CLEAR_ALL – set missing values for the array element CRSP_CLEAR_RANGE – set missing values for the array element CRSP_CLEAR_SET – set missing values for the array element
RETURN VALUES:	CRSP_SUCCESS: if successfully cleared or NULL row or row array CRSP_FAIL: if unknown arrtype or subtype
SIDE EFFECTS:	The array pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT it doesn't do anything. If any other flag is passed it set missing value in the in the arr part
PRECONDITIONS:	The row pointer must be NULL or initialized with a valid arrtype and subtype.

crsp_util_clear_ts Loads Missing Values to a Time Series

PROTOTYPE:	int crsp_util_clear_ts (CRSP_TIMESERIES *ts, int clearflag)
DESCRIPTION:	Loads missing values into a time series on a range level or array level.
ARGUMENTS:	CRSP_TIMESERIES *ts – pointer to a time series to be loaded.
	int clearflag – constant identifying the level of clearing. Supported values are:
	CRSP_CLEAR_INIT – only reset beg and end to 0
	CRSP_CLEAR_ALL – set beg and end to 0 and set missing values for all elements in the time series.
	CRSP_CLEAR_RANGE – set missing values for elements between beg and end of the time shares
	CRSP_CLEAR_SET – set ranges in the 0'th element of the CRSP_TIMESERIES to missing values specific to the array type.
RETURN VALUES:	CRSP_SUCCESS: if success
	CRSP_FAIL: if bad parameters
SIDE EFFECTS:	The time series pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT only beg and end are set to 0. If clearflag is CRSP_CLEAR_RANGE all elements between beg and end are set to missing values. If clearflag is CRSP_CLEAR_ALL beg and end are set to 0 and missing values are set for all elements in the time series. If clearflag is CRSP_CLEAR_SET, the 0'th element of the time series is set to the missing value for the arrtype and subtype.
PRECONDITIONS:	The time series pointer must be NULL or initialized with valid arrtype and subtype.

crsp_util_clear_arr_user Load Missing Values to an Array Based on User Function

PROTOTYPE:	int crsp_util_clear_arr_user (CRSP_ARRAY *arr, void (*clear_fnct) void *elem, int clearflag)
DESCRIPTION:	Loads missing values into a CRSP_ARRAY on a range level or array level.
ARGUMENTS:	CRSP_ARRAY *arr – pointer to a CRSP_ARRAY void (*clear_fnct) void *elem – pointer to user's function int clearflag – constant identifying the level of clearing. Supported values are: CRSP_CLEAR_INIT – only reset num to 0 CRSP_CLEAR_ALL – set num to 0 and set missing values for all elements in the object arrays CRSP_CLEAR_RANGE – set missing values for elements between 0 and num-1 CRSP_CLEAR_SET – set ranges in the maxarr-1'th element of the CRSP_ARRAY to missing values specific to the array type.
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters

SIDE EFFECTS:	The array pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT only num is set to 0. If clearflag is CRSP_CLEAR_RANGE all elements between 0 and num-1 are set to missing values. If clearflag is CRSP_CLEAR_ALL num is set to 0 and missing values are set for all elements in the object arrays. If clearflag is CRSP_CLEAR_SET, the maxarr-1'th element of the array is set to the missing value for the arrtype and subtype.
PRECONDITIONS:	The array pointer must be NULL or initialized with a valid arrtype and subtype.User's function must exist with one void pointer argument. This function must be able to clear one element of the user's array.

crsp_util_clear_row_user Load Missing Values to a Row

PROTOTYPE:	int crsp_util_clear_row_user (CRSP_ROW *row, oid (*clear_fnct) void *elem, int clearflag)
DESCRIPTION:	Loads missing values into a CRSP_ROW
ARGUMENTS:	CRSP_ROW *row – pointer to a CRSP_ROW. void (*clear_fnct) void *elem - pointer to user's function int clearflag – constant identifying the level of clearing. Supported values are: CRSP_CLEAR_INIT – just return success CRSP_CLEAR_ALL – set missing values for the array element CRSP_CLEAR_RANGE – set missing values for the array element CRSP_CLEAR_SET – set missing values for the array element
RETURN VALUES:	CRSP_SUCCESS: if successfully cleared or NULL row or row array CRSP_FAIL: if unknown arrtype or subtype
SIDE EFFECTS:	The array pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT it doesn't do anything. If any other flag is passed it set missing value in the in the arr part
PRECONDITIONS:	The array pointer must be NULL or initialized with a valid arrtype and subtype.User's function must exist with one void pointer argument. This function must be able to clear one element of the user's array.

crsp_util_clear_ts_user Loads Missing Values to a Time Series

PROTOTYPE:	int crsp_util_clear_ts_user (CRSP_TIMESERIES *ts, void (*clear_fnct) void *elem, int clearflag)
DESCRIPTION:	Loads missing values into a time series on a range level or array level.
ARGUMENTS:	CRSP_TIMESERIES *ts – pointer to a time series to be loaded. void (*clear_fnct) void *elem int clearflag – constant identifying the level of clearing. Supported values are: CRSP_CLEAR_INIT – only reset beg and end to 0 CRSP_CLEAR_ALL – set beg and end to 0 and set missing values for all elements in the time series. CRSP_CLEAR_RANGE – set missing values for elements between beg and end of the time shares CRSP_CLEAR_SET – set ranges in the 0'th element of the CRSP_TIMESERIES to missing values specific to the array type.
RETURN VALUES:	CRSP_SUCCESS: if success CRSP_FAIL: if bad parameters
SIDE EFFECTS:	The time series pointer has all allocated fields initialized according to the clearflag. If clearflag is CRSP_CLEAR_INIT only beg and end are set to 0. If clearflag is CRSP_CLEAR_RANGE all elements between beg and end are set to missing values. If clearflag is CRSP_CLEAR_ALL beg and end are set to 0 and missing values are set for all elements in the time series. If clearflag is CRSP_CLEAR_SET, the 0'th element of the time series is set to the missing value for the array type and subtype.
PRECONDITIONS:	The array pointer must be NULL or initialized with a valid arrtype and subtype.User's function must exist with one void pointer argument. This function must be able to clear one element of the user's array.

`crsp_util_cmp_float` Compares two floating point values at different levels of precision

PROTOTYPE:	`int crsp_util_cmp_float (float *item1, float *item2, int epsflag, double epsilon)`
DESCRIPTION:	Compares two floating point values at different levels of precision.
FORMAL PARAMETERS:	`float *item1` - pointer to the first floating point number `float *item2` - pointer to the second floating point number `int epsflag` - code used to determine precision in comparisons. -1 make an absolute comparison to the value in epsilon. If the difference between item1 and item2 is less than or equal to epsilon, they are reported as equal 0 make an exact comparison between the two numbers >0 the number of significant digits to compare. Format to a decimal scientific notation at that many digits, and then compare exactly mantissa, sign, and magnitude if global int `flt_compare_mode` is 1 then positive epsflag is done as a relative comparison. double epsilon the minimum difference between the two numbers before they are considered as equal if epsflag is -1. Ignored if epsflag is not -1.
RETURN VALUE:	1 if item1 is greater than item2 0 if item1 is equal to item2 within epsilon parameters -1 if item1 is less than item2

`crsp_util_cmp_string` Compares two character strings using preprocessing options.

PROTOTYPE:	`int crsp_util_cmp_string(char *item1, char *item2, int cepsflag)`
DESCRIPTION:	Compares two character strings using preprocessing options.
ARGUMENTS:	`char *item1`- pointer to the first string `char *item2`– pointer to the second string `int cepsflag` - code used to determine preprocessing in comparison 0 - exact string comparison using strcmp 1 - trim strings (remove trailing white space) before comparing them 2 - squeeze strings (remove leading and trailing white space and replace repeated white space with a single space) before comparing them
RETURN VALUES:	1 if item1 is greater than item2 0 if item1 is equal to item2 after preprocessing -1 if item1 is less than item2
SIDE EFFECTS:	If trim or squeeze options are used, the string may be modified
PRECONDITIONS:	Both strings must be NULL terminated

`crsp_util_delete_ts` Deletes Ranges from a CRSP_TIMESERIES Based on a Second CRSP_TIMESERIES

PROTOTYPE:	`int crsp_util_delete_ts(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *del_ts, int exactflag, int rangeflag, int cepsflag, int epsflag, double epsilon)`
DESCRIPTION:	Deletes ranges from a `CRSP_TIMESERIES` given a second `CRSP_TIMESERIES`. It is optional whether the structure must match an existing row exactly or if only key fields identify the structure to delete

ARGUMENTS:	CRSP_TIMESERIES *src_ts - pointer to existing CRSP_TIMESERIES to be modified CRSP_TIMESERIES *del_ts - pointer to existing CRSP_TIMESERIES to be removed from the source int exactflag - option on whether the element to be deleted must be an exact much or if a match on the keys fields only is sufficient. Possible values are: CRSP_MATCH_EXACT the function reports CRSP_NOT_FOUND if any overlapping rows in the source and delete time series do not match CRSP_MATCH_IGNORE the function only considers the ranges of the time series, not the values within the time series. int rangeflag - option on which types of overlapping ranges are accepted. Possible values are: CRSP_RANGE_NONE no restrictions are made on input ranges; all overlapping ranges are erased CRSP_RANGE_BEG the begin ranges must match between source and delete time series CRSP_RANGE_END the end ranges must match between source and delete time series CRSP_RANGE_ONE at least one of the begin or end ranges must match between source and delete time series int cepsflag - flag used to compare string fields within structure. “crsp_util_cmp_string” on page 156 . int epsflag - flag used to compare float fields within structure. See “crsp_util_cmp_float” on page 156 for values. double epsilon - the maximum difference between two float fields in the structures before they are considered different, used only if epsflag is -1.
RETURN VALUES:	CRSP_SUCCESS - if successfully deleted CRSP_NOT_FOUND - if del_ts values not found in the src_ts values according to exactflag CRSP_FAIL - if bad parameter or fail function calls or mismatched ranges according to rangeflag
SIDE EFFECTS:	beg and end of source time series will be changed
PRECONDITIONS:	The time series must be allocated, and elem must be correct type with valid data in at least key fields.

crsp_util_insert_ts Data Into a CRSP_TIMESERIES from a Second CRSP_TIMESERIES

PROTOTYPE:	int crsp_util_insert_ts(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *ins_ts, int rangeflag)
DESCRIPTION:	Inserts ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES. Options govern handling of overlapping data
ARGUMENTS:	CRSP_TIMESERIES *src_ts - pointer to existing CRSP_TIMESERIES to be modified CRSP_TIMESERIES *ins_ts - pointer to existing CRSP_TIMESERIES to be inserted to the source int rangeflag - option on which types of overlapping ranges are accepted. Possible values are: CRSP_RANGE_OVER no restrictions are made on input ranges; all overlapping ranges are replaced with the insert ts values CRSP_RANGE_KEEP no restrictions are on input ranges; keep existing values in all overlapping ranges CRSP_RANGE_BEG the insert end must be one less than the source begin CRSP_RANGE_END the insert begin must be one higher than the source end CRSP_RANGE_ONE at least one of the previous two conditions must be true
RETURN VALUES:	CRSP_SUCCESS - if successfully inserted CRSP_NOT_FOUND - if del_ts values not found in the src_ts values according to exactflag
SIDE EFFECTS:	beg and end of source time series will be changed
PRECONDITIONS:	The time series must be allocated, and elements must agree on type with valid data in at least key fields

crsp_util_update_ts Updates Data in a CRSP_TIMESERIES From Data in a Second CRSP_TIMESERIES

PROTOTYPE:	int crsp_util_update_ts(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *new_ts, CRSP_TIMESERIES *old_ts, int exactflag, int cepsflag, int epsflag, double epsilon)
DESCRIPTION:	Updates ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES. It is optional whether the structure must match an existing row exactly or if only key fields identify the structure to delete.

ARGUMENTS:	CRSP_TIMESERIES *src_ts - pointer to existing CRSP_TIMESERIES to be modified CRSP_TIMESERIES *new_ts - pointer to existing CRSP_TIMESERIES to be updated into the source CRSP_TIMESERIES *old_ts - pointer to existing CRSP_TIMESERIES to be compared to the existing source. Used only for exact matches (exactflag=CRSP_MATCH_EXACT) int exactflag - option on whether the element to be updated must be an exact match or if a match on the keys fields only is sufficient. Possible values are: CRSP_MATCH_EXACT the function reports CRSP_NOT_FOUND if any overlapping rows in the source and old time series do not match CRSP_MATCH_IGNORE the function only considers the ranges of the time series, not the values within the time series. int* code - pointer to location used to store structure specific results of a comparison of all fields. If code is -1, then only the key-based comparison is made. Otherwise, code is set to a positive number containing information of the fields that are different int cepsflag - flag used to compare string fields within structure. See "crsp_util_cmp_string" on page 156 for values. int epsflag - flag used to compare float fields within structure. See "crsp_util_cmp_float" on page 156 for values. double epsilon - the maximum difference between two float fields in the structures before they are considered different, only used if epsflag is -1
RETURN VALUES:	CRSP_SUCCESS - if successfully updated CRSP_NOT_FOUND - if old_ts values not found in the src_ts values according to exactflag CRSP_FAIL - if bad parameter or fail function calls or mismatched ranges according to rangeflag
SIDE EFFECTS:	beg and end of source time series will be changed and data will be loaded if successful.
PRECONDITIONS:	The time series must be allocated, and array types and calendars must agree, with valid data in at least key fields.

crsp_util_is_missing Determine Whether One Array Element Contains Missing Data

PROTOTYPE:	int crsp_util_is_missing(void *elem, int arrtype, int subtype)
DESCRIPTION:	Determines whether one passed array element contains missing data according to arrtype and subtype. This function is useful if there are multiple missing values for a type of data. Normally the first element of a CRSP time series array, or the last element of a CRSP_ARRAY contains the primary missing data for that type of data. This function supports all primary and secondary missing values.
ARGUMENTS:	void *elem - a pointer to element to be checked int arrtype - a CRSP-defined array type constant identifying the structures int subtype - a CRSP-defined subtype constant identifying possible subcategories of data loaded in the element
RETURN VALUES:	0 - CRSP_NOT_MISSING - value present 1 - CRSP_IS_MISSING - value missing -1 - unknown or unsupported arrtype or subtype
SIDE EFFECTS:	none
PRECONDITIONS:	elem must point to valid data for the structure indicated by arrtype.

crsp_util_reset_enddts Resets End Date for the CRSP_ARRAY Histories

PROTOTYPE:	int crsp_util_reset_enddts(CRSP_ARRAY *array, int lastenddt, int begdt_offset, int enddt_offset)
DESCRIPTION:	Resets end date to the next begin date minus 1 for the CRSP_ARRAY structure
ARGUMENTS:	CRSP_ARRAY *array – source array structure lastenddt - date in CCYYMMDD format to be used for the end date of the last event. Resets end dates for CRSP_ARRAY event histories. Sets end dates in array structure to one day before the following event's effective date. The end date of the last event must be provided as a parameter. Only valid for arrays containing contiguous increasing effective dates. int begdt_offset - offset of begin date field of the specified array stucture int enddt_offset - offset of end date field of the specified array structure
RETURN VALUES:	CRSP_SUCCESS - if successfully set CRSP_FAIL - if error in parameters or loading process
SIDE EFFECTS:	enddt - offset for each event from - to num-1 is updated.
PRECONDITIONS:	Array must be allocated and loaded. begdt_offset with each event structure must be an integer date feed in YYYYMMDD format.

crsp_util_merge_arr Compare Two Source Arrays; if They are Equal, Copy Main Array Data into Target Array

PROTOTYPE:	`int crsp_util_merge_arr(CRSP_ARRAY *trg_arr, CRSP_ARRAY *main_arr, CRSP_ARRAY *sub_arr, int *status, int cepsflag, int epsflag, double epsilon)`
DESCRIPTION:	Compare two source array, put two records in order into target array, if they are equal, copy main array data into target, where the main array takes the precedence.

ARGUMENTS:	`CRSP_ARRAY *trg_arr` - output, pointer to `CRSP_ARRAY`
	`CRSP_ARRAY *main_arr` - input, pointer to `CRSP_ARRAY`
	`CRSP_ARRAY *sub_arr` - input, pointer to `CRSP_ARRAY`
	`int *status` - flag to indicates the status of the data

RETURN VALUES:	`CRSP_SUCCESS` - successfully ran `CRSP_FAIL` - failed to run
SIDE EFFECTS:	Any previously stored data in target array will be overwritten

crsp_util_merge_ts Merges Two Source Time Series to One Target Time Series

PROTOTYPE:	`int crsp_util_merge_ts(CRSP_TIMESERIES *trg_ts, CRSP_TIMESERIES *src1_ts, CRSP_TIMESERIES *src2_ts)`
DESCRIPTION:	Merges two source `ts(src1_ts, src2_ts)` to target `ts(trg_ts)`, where `trg_ts` takes the precedence

ARGUMENTS:	`trg_ts` - output, pointer to `CRSP_TIMESERIES`
	`src1_ts` – input, pointer to `CRSP_TIMESERIES`
	`src2_ts` - input, point to `CRSP_TIMESERIES`

RETURN VALUES:	`CRSP_SUCCESS` - successfully ran
	`CRSP_FAIL` - failed to run

Data to Time Series Mapping Utility Functions

FUNCTION	DESCRIPTION	PAGE
`crsp_util_map_arr2ts`	Maps a subset of fields from a `CRSP_ARRAY` to a `CRSP_TIMESERIES`	page 160
`crsp_util_map_row2ts`	Maps a subset of fields from a `CRSP_ROW` to a `CRSP_TIMESERIES`	page 161
`crsp_util_map_ts2ts`	Maps a subset of fields from one `CRSP_TIMESERIES` to another	page 170

crsp_util_map_arr2ts Maps Selected Fields in a CRSP_ARRAY into a CRSP_TIMESERIES

PROTOTYPE:	`int crsp_util_map_arr2ts (CRSP_ARRAY *src_arr, CRSP_TIMESERIES *trg_ts, int flags, int rangflag, int offset, int length, int begdt_offset, int enddt_offset)`
DESCRIPTION:	Loads selected fields in a `CRSP_ARRAY` into a `CRSP_TIMESERIES`. The specific fields are identified with the offset within the array structure and the length of the field. Date range fields in the array used to map to the time series calendar are specified with their offsets. This function only works with status change event arrays where each event refers to the status values until the next event.

ARGUMENTS:	CRSP_ARRAY *src_arr – pointer to source array. The array must be allocated and loaded with the data to map. CRSP_TIMESERIES *trg_ts – pointer to target time series with desired calendar loaded. int flags – flags used to interpret date ranges CRSP_ACTUAL – target is loaded with source at the end of target period, trg[i] = f(src[i]) CRSP_EFFECTIVE – target is loaded with source at the end of the previous target period, trg[i+1] = f(src[i]) CRSP_NLAST – last data from the source is moved to all periods on target int rangflag – flags used to interpret time series ranges outside of explicit source ranges. Flags are: CRSP_RANGE_AS_IS – as it is, target set to missing outside of explicit source range CRSP_RANGE_FIRST – assume first source event is valid back to beginning of target range CRSP_RANGE_LAST – assume last source event is good forever CRSP_RANGE_FIRST_LAST – both first and last int offset – the offset in bytes of the target field from the beginning of the structure in the source array. int length – the number of bytes of the target field int begdt_offset – the offset in bytes of the effective date field of the source structure from the beginning of the structure in the source array. int enddt_offset – the offset in bytes of the last effective date field of the source structure from the beginning of the structure in the source array.
RETURN VALUES:	CRSP_SUCCESS: if successful CRSP_FAIL: if bad parameter, mismatched time series size or uninitialized source or target, or unmatched parameters.
SIDE EFFECTS:	The target time series is loaded with data from the source array according to flags.
PRECONDITIONS:	The source array must be allocated and loaded with the data to copy. The target time series and calendar must be allocated. The target size_of_array_width must match the length parameter and target object fields arrtype and subtype must be set according to the data to be loaded. No offsets can extend past the size of the array structure.

crsp_util_map_row2ts Maps Selected Fields in One CRSP_ROW into a CRSP_TIMESERIES

PROTOTYPE:	int crsp_util_map_row2ts (CRSP_ROW *row_ts, CRSP_TIMESERIES *trg_ts, int offset)
DESCRIPTION:	Loads selected fields in a CRSP_ROW into a CRSP_TIMESERIES. The specific fields are identified by the offset within the source structure and the size_of_array_width of the target. The row field value is copied to every period in the target time series.
ARGUMENTS:	CRSP_ROW *src_row – pointer to source row. It must be allocated and loaded with the data to map. CRSP_TIMESERIES *trg_ts – pointer to target time series with desired calendar loaded and desired beg and end set. int offset – the offset in bytes of the target field from the beginning of the structure in the source row.
RETURN VALUES:	CRSP_SUCCESS: if successful CRSP_FAIL: if bad parameter, uninitialized source or target, or unmatched parameters.
SIDE EFFECTS:	The target time series is loaded with data from the source. Data is copied to each period between the target beg and end.
PRECONDITIONS:	The source time series must be allocated and loaded with the data to copy. The target time series and calendar must be allocated and the desired beg and end must be set. Target object fields arrtype and subtype must be set according to the data to be loaded.

crsp_util_map_ts2ts Maps Selected Fields in One CRSP_TIMESERIES into Another

PROTOTYPE:	int crsp_util_map_ts2ts (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int offset)
DESCRIPTION:	Loads selected fields in a CRSP_TIMESERIES into another CRSP_TIMESERIES. The specific fields are identified bythe offset within the source structure and the size_of_array_width of the target. The two time series must have identical calendars.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – pointer to source time series. It must be allocated and loaded with the data to map. CRSP_TIMESERIES *trg_ts – pointer to target time series. int offset – the offset in bytes of the target field from the beginning of the structure in the source array.
RETURN VALUES:	CRSP_SUCCESS: if successful CRSP_FAIL: if bad parameter, mismatched time series size or uninitialized source or target, or unmatched parameters.
SIDE EFFECTS:	The target time series is loaded with data from the source array according to flags. Data is copied on a period by period basis and the target beg and end are copied from the source.
PRECONDITIONS:	The source time series must be allocated and loaded with the data to copy. The target time series and calendar must be allocated. Target object fields arrtype and subtype must be set according to the data to be loaded. The two time series must have identical calendars.

CRSPAccess C Database Information Function

This function is used to retrieve information about a database.

`crsp_root_info_get` Load CRSPAccess Database Information

PROTOTYPE:	`int crsp_root_info_get (int crspnum, CRSP_ROOT_INFO *info)`
DESCRIPTION:	Loads database information from a CRSPAccess database into a structure. CRSP_ROOT_INFO is defined in *crsp_objects.h.* The following fields are available: `crt_date` – 25-character string containing the time the database was created, in the format "Dow Mon DD HH:MM:SS YYYY" `mod_date` – 25-character string containing the time the database was last modified, in the format "Dow Mon DD HH:MM:SS YYYY" `cut_date` – 25-character string containing the last date of data in the database, currently loaded as YYYYMM `binary_type` – L if IEEE Little-Endian, and B if IEEE Big-Endian `code version` – 19-character string containing the CRSPAccess version used to create the database `product_code` – 11-character CRSP Product Code product_name – 47-character Product name of the database version – integer version number of the database `settypes` – an array of up to eight integer settypes available in the database setids – an array of up to eight integer setids available in the database setnames – an array of up to eight names of the sets in the database `numsets` – the number of data sets in the database `calids` – an array of up to eight integer calids of calendars available in the database calavail – an array of up to eight integer caltypes of the calendars available in the database calnames – an array of up to eight names of the calendars in the database `numcals` – the number of calendars in the database
ARGUMENTS:	`int crspnum` – database identifier returned by a CRSPAccess database open function `CRSP_ROOT_INFO *info` – structure that will be loaded with database information
RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if database is not open or error loading information structure
SIDE EFFECTS:	none
PRECONDITIONS:	the database must be opened with one of the CRSPAccess open functions.

DATA UTILITY FUNCTIONS

The CRSP library contains several groups of data functions described in the following table. Subsections in this sec- tion contain the descriptions of the individual functions within each of the function groups.

FUNCTIONS GROUP	DESCRIPTION	PAGE
Adjust Functions	Functions to Adjust Prices or Other Data	page 162
Excess Returns Functions	Functions to Make Excess Returns Calculations	page 179
Name Array Functions	Functions to Map Name History Fields to Time Series	page 180
NASDAQ Information Mapping Functions	Functions to Map NASDAQ Information Elements to Time Series	page 164
Returns Functions	Functions to Calculate Returns	page 166
Shares Outstanding Functions	Functions to Manipulate Shares Data	page 174
Subset Functions	Functions to Print Specialized Stock Data	page 178
Translation Functions	Functions to Translate Data to New Time Series	page 191

Adjust Functions

These functions adjust prices, dividends, volumes, and shares for splits or other price factors.

FUNCTION	DESCRIPTION	PAGE
`crsp_adj_load`	Builds a Price Adjustment Structure Array	page 163
`crsp_adj_map_ts`	Adjusts a Source `CRSP_TIMESERIES` According to an Adjustment Array	page 167
`crsp_adj_map_arr`	Adjusts a Source `CRSP_ARRAY` According to an Adjustment Array	page 165
`crsp_adj_stk`	Adjusts all relevant fields in a Source Stock Structure	page 164

crsp_adj_load Builds a Price Adjustment Structure Array

PROTOTYPE:	int crsp_adj_load (CRSP_STK_STRUCT *stk, CRSP_ARRAY *adj_arr, int adjdt, int factyp, int gapflg, int knowexch)
DESCRIPTION:	loads a crsp_array of crsp_adj_struct structures with cumulative adjustment factors and effective dates.
ARGUMENTS:	CRSP_STK_STRUCT *stk – stk structure with at least events and prices loaded. CRSP_ARRAY *adj_arr – adj array that will be loaded. It must exist with enough space to store completed array of adjustment events int adjdt – base anchor date guaranteed to have 1.0 factor int factyp – code of adjustment type: 0 = stock splits and dividends only 1 = all dists with facpr int gapflg – 0 carry adjustments over a gap 1 adjustments stop when trading on unknown exchange int knowexch – unused, always set to 0.
RETURN VALUES:	CRSP_SUCCESS: if adjustment structure successfully loaded CRSP_FAIL: if error in parameters or structures
SIDE EFFECTS:	The adj_arr will be loaded. The subtype in the adj_arr is set to the adjust base date.
PRECONDITIONS:	It is assumed that events and prices have been loaded. The adj_arr must have arrtype CRSP_ADJ_STRUCT_NUM

crsp_adj_map_ts Adjusts a Source CRSP_TIMESERIES According to an Adjustment Array

PROTOTYPE:	int crsp_adj_map_ts(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, CRSP_ARRAY *adj_arr, int begrng, int endrng, int endflg, int direct)
DESCRIPTION:	adjusts a source time series according to an adjust array and put the results in a target time series. The adjust array must exist.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – pointer to source time series, already loaded by crsp_adj_load
	CRSP_TIMESERIES *trg_ts – pointer to preexisting target time series, empty
	CRSP_ARRAY *adj_arr – pointer to adjustment array already loaded
	int begrng – begin date index of the date range adjustment
	int endrng – end date index of the date range adjustment
	int endflg – determines whether adjustments can be made after the last day of prices. If set to 1 missing values are set for the range after the end date; otherwise the last adjustment value is used
	int direct – direction flag multiply or divide with adj factor 1= multiply with adjustment factor
	-1= divide with adjustment factor
RETURN VALUES:	CRSP_SUCCESS: (integer) if successfully adjusted
	CRSP_FAIL: if error in parameters or adjustment
SIDE EFFECTS:	The target time series is loaded with the adjusted data items from the source time series, for the date range specified by begrng and endrng.
PRECONDITIONS:	The src_ts, trg_ts, and adj_arr must exist. src_ts and trg_ts must have the same arrtype and subtype and the same calendar. The src_ts subtype cannot be any of these: CRSP_RETURN_NUM or CRSP_PRICE_ADJ_NUM or CRSP_VOLUME_ADJ_NUM. The wanted date range must be a subset of the data date range and the adj_arr date range.

crsp_adj_map_arr Adjusts a Source CRSP_ARRAY According to an Adjustment Array

PROTOTYPE:	`int crsp_adj_map_arr (CRSP_ARRAY *src_arr, CRSP_ARRAY *trg_arr, CRSP_ARRAY *adj_arr, int begdt, int enddt, int endflg, int direct)`
DESCRIPTION:	adjusts a source `CRSP_ARRAY` according to an adjust array and put the results in a target `CRSP_ARRAY`. The adjust array must exist.
ARGUMENTS:	`CRSP_ARRAY *src_arr` – pointer to source time series, already loaded `CRSP_ARRAY *trg_arr` – pointer to preexisting target time series, empty `CRSP_ARRAY *adj_arr` – pointer to adjustment array already loaded by crsp_adj_load `int` begdt – begin date index of the date range adjustment `int enddt` – `end` date index of the date range adjustment `int endflg` – determines whether adjustments can be made after the last day of prices. If set to 1 missing values are set for the range after the `end` date; otherwise the last adjustment value is used. `int direct` – direction flag multiply or divide with adj factor `1`= multiply by adjustment factor (prices) `-1`= divided by adjustment factor (shares and values)
RETURN VALUES:	`CRSP_SUCCESS`: if successfully adjusted `CRSP_FAIL`: if error in parameters or adjustment
SIDE EFFECTS:	The target `CRSP_ARRAY` is loaded with the adjusted data items from the source `CRSP_ARRAY`, for the date range specified by begdt and `enddt`.
PRECONDITIONS:	The `src_arr`, `trg_arr` and `adj_arr` must exist. `src_arr` and `trg_arr` must have the same `arrtype` and `subtype`. The `src_arr subtype` can not be any of these: `CRSP_SHARES_ADJ_NUM` or `CRSP_DISTS_ADJ_NUM` or `CRSP_DELIST_ADJ_NUM`. The `wanted` date range must be a subset of the data date range and the `adj_arr` date range.

crsp_adj_stk Adjusts All Relevant Fields in a Source Stock Structure

PROTOTYPE:	`int crsp_adj_stk(CRSP_STK_STRUCT *src_stk, CRSP_STK_STRUCT *trg_stk, int adjdt, int factyp, int gapflg, int endflg, int knownexch)`
DESCRIPTION:	adjusts a source stk structure according to an adjust array and put the results in a target stk structure. The adjust array is initialized and loaded inside this function.
ARGUMENTS:	`CRSP_STK_STRUCT *src_stk` – pointer to source stk structure `CRSP_STK_STRUCT *trg_stk` – pointer to target stk structure `int adjdt` – base anchor adjustment date guaranteed to have 1.0 factor `int factyp` – code of adjustment type: 0 = stock splits and dividends only 1 = all dists with `facpr` `int gapflg` – take into account gaps in the date range or not (values: 1,0) and set the `adjfac` accordingly if `adjdt` < begin of gap and `gapflg` is set then zero out all `adjfac` after the gap if `adjdt` > `end` of gap and `gapflg` is set then zero out all `adjfac` before the gap if `adjdt` between the gap and `gapflg` is set then zero out all `adjfac` `int endflg` – determines whether adjustments can be made after the last day of prices. If set to 1 missing values are set for the range after the `end` date; otherwise the last adjustment value is used. `int knownexch` – unused. Always set to 0, no restriction
RETURN VALUES:	`CRSP_SUCCESS`: if successfully adjusted `CRSP_FAIL`: if error in parameters or adjustment
SIDE EFFECTS:	The target stk structure is loaded with the adjusted data items from the source stk structure. The `subtypes` of objects loaded with adjusted data are changed to reflect the adjusted data. See crsp_const for `*_NUM subtype` constants.
PRECONDITIONS:	The source `src_stk` must be already loaded with all the modules `wanted` to be adjusted. The target `trg_stk` must already be initialized. Use `crsp_stk_open` to initialize a new structure. If an object `subtype` indicates adjusted data is already loaded, no adjustment will be made. Use the `crsp_stk_clear` function to reset stock structures to unadjusted `subtypes`.

Excess Returns Functions

CRSP excess returns compare two returns time series, and produce a series of returns with the amounts a source time series is in excess of a base time series.

`crsp_xs_calc` CRSP Stock Excess Returns Calculation

PROTOTYPE:	`int crsp_xs_calc (CRSP_TIMESERIES *bas_ts, CRSP_TIMESERIES *ind_ts, CRSP_TIMESERIES *trg_ts, int beg, int end, int missflag)`
DESCRIPTION:	general CRSP stock excess returns calculation given a base return series, a reference return series, and a date range, loads excess returns for each date in the series.
ARGUMENTS:	`CRSP_TIMESERIES *bas_ts` – time series of issue returns `CRSP_TIMESERIES *ind_ts` – time series of index returns `CRSP_TIMESERIES *trg_ts` – target output of excess returns `int beg, end` – index range to calculate excess returns `int missflag` – flag for handling missing returns `CRSP_KEEP` – base missing returns are copied to target, index returns are compounded over gap `CRSP_SMOOTH` – first return after gap is geometrically averaged so entire gap has the same amount `CRSP_IGNORE` – missing returns are treated as 0's; missing returns in index always generate a missing excess return It is assumed that targ, base, and ind have been allocated and have the same calendar. `0 < start <=end < maxarr` must be true for each time series
RETURN VALUES:	`CRSP_SUCCESS`: if returns successfully loaded `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	The target time series object is loaded with excess returns data. The range is set to the min of current `beg` and passed start, and max of current `end` and passed `end`. Any excess returns already loaded are kept only if they are outside of start/end. If there is a gap between existing range and new range the returns are loaded with missing values.
PRECONDITIONS:	The `subtype` of `bas_st` and `ind_ts` is `CRSP_RETURN_NUM` The `subtype` of `trg_ts` is `CRSP_RETURN_XS_NUM` or `CRSP_RETURN_CUM_NUM`

`crsp_xs_port` Builds Portfolio Returns into One Series

PROTOTYPE:	`int crsp_xs_port (CRSP_TIMESERIES **ind_ts, int indtypes, CRSP_TIMESERIES port_ts, int porttype, CRSP_TIMESERIES *trg_ts)`
DESCRIPTION:	builds a time series of index returns by mapping from an array of index returns time series based on a portfolio time series
ARGUMENTS:	`CRSP_TIMESERIES **ind_ts` – pointer to indexes returns time series `int indtypes` – total number of indexes types `CRSP_TIMESERIES` **port_ts – time series array of portfolio assignments `int porttype` – portfolio type index of interest `CRSP_TIMESERIES *trg_ts` – target index based on portfolio `trg_ts` and all indexes must be allocated and have the same calendar
RETURN VALUES:	`CRSP_SUCCESS`: if returns successfully loaded `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	The `trg_ts` time series object is loaded with index data by mapping to an index based on a portfolio time series.
PRECONDITIONS:	The target time series and all the indexes time series must exist prior calling the function and must all verify (see crsp_obj_verify_ts on page 144) and have the same calendar

Name Array Functions

These functions map elements in the names event array to time series.

FUNCTION	DESCRIPTION	PAGE
`crsp_map_shrcd`	Map name history share codes to a time series	page 166
`crsp_map_exchcd`	Map name history exchange codes to a time series	page 170
`crsp_map_siccd`	Map name history siccd codes to a time series	page 170
`crsp_map_ncusip`	Map name history name CUSIPs to a time series	page 167

FUNCTION	DESCRIPTION	PAGE
crsp_map_ticker	Map name history tickers to a time series	page 182
crsp_map_comnam	Map name history company names to a time series	page 168
crsp_map_shrcls	Map name history share classes to a time series	page 168
crsp_cur_name	Finds index of name structure on a select date	page 168

crsp_map_shrcd Map Share Codes to a Time Series

PROTOTYPE:	int crsp_map_shrcd (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)
DESCRIPTION:	loads a target time series from a source CRSP_ARRAY by copying the share type code of the stock event's names structure over each restricted period according to the target calendar file.
ARGUMENTS:	CRSP_ARRAY *names_arr – source CRSP_ARRAY stock event's names histories CRSP_TIMESERIES *trg_ts – target time series int flags – flags passed to the function. One of: CRSP_ACTUAL means data from the source is moved to the same period on target trg[i] = f(src[i]) CRSP_EFFECTIVE means data from the source is moved to the next period on target trg[i+1] = f(src[i]) CRSP_NLAST means last data from the source is moved to all periods on target
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	Source CRSP_ARRAY must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The names_arr arrtype must be CRSP_STK_NAME_NUM The trg_ts subtype must be CRSP_SUB_SHRCD_NUM

crsp_map_exchcd Map Exchange Codes to a Time Series

PROTOTYPE:	int crsp_map_exchcd (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)
DESCRIPTION:	loads a target time series from a source CRSP_ARRAY by copying the exchange code of the stock event's names structure over each restricted period according to the target calendar file.
ARGUMENTS:	CRSP_ARRAY *names_arr – source CRSP_ARRAY stock event's names histories CRSP_TIMESERIES *trg_ts – target time series int flags – flags passed to the function. One of: CRSP_ACTUAL means data from the source is moved to the same period on target trg[i] = f(src[i]) CRSP_EFFECTIVE means data from the source is moved to the next period on target trg[i+1] = f(src[i]) CRSP_NLAST means last data from the source is moved to all periods on target
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	Source CRSP_ARRAY must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The names_arr arrtype must be CRSP_STK_NAME_NUM The trg_ts arrtype must be CRSP_INTEGER_NUM and subtype must be CRSP_SUB_EXCHCD_NUM

crsp_map_siccd Map SIC Codes to a Time Series

PROTOTYPE:	int crsp_map_siccd (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)
DESCRIPTION:	loads a target time series from a source CRSP_ARRAY by copying the SIC code of the stock event's names structure over each restricted period according to the target calendar file.

ARGUMENTS:	CRSP_ARRAY *names_arr – source CRSP_ARRAY stock event's names histories CRSP_TIMESERIES *trg_ts – target time series int flags – flags passed to the function. One of: CRSP_ACTUAL means data from the source is moved to the same period on target trg[i] = f(src[i]) CRSP_EFFECTIVE means data from the source is moved to the next period on target trg[i+1] = f(src[i]) CRSP_NLAST means last data from the source is moved to all periods on target
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	Source CRSP_ARRAY must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The names_arr arrtype must be CRSP_STK_NAME_NUM The trg_ts subtype must be CRSP_SUB_SICCD_NUM

crsp_map_ncusip Map CUSIPs to a Time Series

PROTOTYPE:	int crsp_map_ncusip (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)
DESCRIPTION:	loads a target time series from a source CRSP_ARRAY by copying the cusip of the stock event's names structure over each restricted period according to the target calendar file.
ARGUMENTS:	CRSP_ARRAY *names_arr – source CRSP_ARRAY stock event's name histories CRSP_TIMESERIES *trg_ts – target time series int flags – flags passed to the function. One of: CRSP_ACTUAL means data from the source is moved to the same period on target trg[i] = f(src[i]) CRSP_EFFECTIVE means data from the source is moved to the next period on target trg[i+1] = f(src[i]) CRSP_NLAST means last data from the source is moved to all periods on target
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	Source CRSP_ARRAY must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The names_arr arrtype must be CRSP_STK_NAME_NUM The trg_ts subtype must be CRSP_SUB_NCUSIP_NUM

crsp_map_ticker Map Tickers to a Time Series

PROTOTYPE:	int crsp_map_ticker (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)
DESCRIPTION:	loads a target time series from a source CRSP_ARRAY by copying the ticker of the stock event's names structure over each restricted period according to the target calendar file.
ARGUMENTS:	CRSP_ARRAY *names_arr – source CRSP_ARRAY stock event's name histories CRSP_TIMESERIES *trg_ts – target time series int flags – flags passed to the function. One of: CRSP_ACTUAL means data from the source is moved to the same period on target trg[i] = f(src[i]) CRSP_EFFECTIVE means data from the source is moved to the next period on target trg[i+1] = f(src[i]) CRSP_NLAST means last data from the source is moved to all periods on target
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	Source CRSP_ARRAY must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The names_arr arrtype must be CRSP_STK_NAME_NUM The trg_ts subtype must be CRSP_SUB_TICKER_NUM

`crsp_map_comnam` Map Company Names to a Time Series

PROTOTYPE:	`int crsp_map_comnam (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flag)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the company name of the stock event's names structure over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *names_arr` – source `CRSP_ARRAY` stock event's names histories
	`CRSP_TIMESERIES *trg_ts` – target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `names_arr arrtype` must be `CRSP_STK_NAME_NUM` The `trg_ts subtype` must be `CRSP_SUB_COMNAM_NUM`

`crsp_map_shrcls` Map Share Classes to a Time Series

PROTOTYPE:	`int crsp_map_shrcls (CRSP_ARRAY *names_arr, CRSP_TIMESERIES *trg_ts, int flags)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the shares class of the stock event's names structure over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *names_arr` – source `CRSP_ARRAY` stock event's name histories
	`CRSP_TIMESERIES *trg_ts` – target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `names_arr arrtype` must be `CRSP_STK_NAME_NUM` The `trg_ts subtype` must be `CRSP_SUB_SHRCLS_NUM`

`crsp_cur_name` Finds Index of Name Structure on a Selected Date

PROTOTYPE:	`int crsp_cur_name (CRSP_ARRAY *names_arr, int ndate, int code)`
DESCRIPTION:	finds the index of the name structure given a date. If the name is earlier than the first name date it returns a value passed as a parameter.
ARGUMENTS:	`CRSP_ARRAY *names_arr` – pointer to a `CRSP_ARRAY` with stock names data loaded. `int ndate` – date in yyyymmdd format to find `int code` – value to return if date earlier than first name
RETURN VALUES:	`name index` – index of last name structure effective on or before date passed \ `code` – if date is before first name structure or names array not initialized.
SIDE EFFECTS:	None
PRECONDITIONS:	Source names array must exist and be allocated

NASDAQ Information Mapping Functions

These functions map data in the NASDAQ Information event arrays to time series.

FUNCTION	DESCRIPTION	PAGE
crsp_map_trtscd	Map NASDAQ status codes to a time series	page 169
crsp_map_nmsind	Map NASDAQ National Market indicator to a time series	page 169
crsp_map_mmcnt	Map NASDAQ Market Maker count to a time series	page 174
crsp_map_nsdinx	Map NASDAQ index code to a time series	page 174

`crsp_map_trtscd` Map NASDAQ Status Codes to a Time Series

PROTOTYPE:	`int crsp_map_trtscd (CRSP_ARRAY *nasdin_arr, CRSP_TIMESERIES *trg_ts, int flags)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the NASDAQ status code of the stock event's `nasdin` structure over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *nasdin_arr` – source `CRSP_ARRAY` stock event's `nasdin` NASDAQ information history
	`CRSP_TIMESERIES *trg_ts` – target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `nasdin_arr arrtype` must be `CRSP_STK_NASDIN_NUM` The `trg_ts subtype` must be `CRSP_SUB_TRTSCD_NUM`

`crsp_map_nmsind` Map NASDAQ National Market Indicator to a Time Series

PROTOTYPE:	`int crsp_map_nmsind (CRSP_ARRAY *nasdin_arr, CRSP_TIMESERIES *trg_ts, int flags)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the National Market indicator of the stock event's `nasdin` stucture over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *nasdin_arr` – source `CRSP_ARRAY` stk events `nasdin` NASDAQ information history
	`CRSP_TIMESERIES *trg_ts` – target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `nasdin_arr arrtype` must be `CRSP_STK_NASDIN_NUM` The `trg_ts subtype` must be `CRSP_SUB_NMSIND_NUM`

crsp_map_mmcnt Map NASDAQ Market Maker Count to a Time Series

PROTOTYPE:	`int crsp_map_mmcnt (CRSP_ARRAY *nasdin_arr, CRSP_TIMESERIES *trg_ts, int flags)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the market maker count of the stock event's `nasdin` structure over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *nasdin_arr` – source `CRSP_ARRAY` stk events `nasdin` NASDAQ Information History
	`CRSP_TIMESERIES *trg_ts` – target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `nasdin_arr arrtype` must be `CRSP_STK_NASDIN_NUM` The `trg_ts subtype` must be `CRSP_SUB_MMCNT_NUM`

crsp_map_nsdinx Map NASDAQ Index Code to a Time Series

PROTOTYPE:	`int crsp_map_nsdinx (CRSP_ARRAY *nasdin_arr, CRSP_TIMESERIES *trg_ts, int flags)`
DESCRIPTION:	loads a target time series from a source `CRSP_ARRAY` by copying the NASDAQ index code of the stock event's `nasdin` structure over each restricted period according to the target calendar file.

ARGUMENTS:	`CRSP_ARRAY *nasdin_arr` – source `CRSP_ARRAY` stk events `nasdin` NASDAQ information history
	`CRSP_TIMESERIES *trg_ts` target time series
	`int flags` – flags passed to the function. One of:
	`CRSP_ACTUAL` means data from the source is moved to the same period on target `trg[i] = f(src[i])`
	`CRSP_EFFECTIVE` means data from the source is moved to the next period on target `trg[i+1] = f(src[i])`
	`CRSP_NLAST` means last data from the source is moved to all periods on target

RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	Source `CRSP_ARRAY` must be loaded with stock event's name histories data. The target time series must exist. Also, a calendar must be associated with the target time series. The `nasdin_arr arrtype` must be `CRSP_STK_NASDIN_NUM` The `trg_ts subtype` must be `CRSP_SUB_NSDINX_NUM`

Returns Functions

These functions make various CRSP returns calculations.

FUNCTION	DESCRIPTION	PAGE
`crsp_ret_calc`	Stock Returns Calculations	page 170
`crsp_ret_calc_del`	CRSP Delisting Returns Calculations	page 180
`crsp_ret_calc_one`	Returns Calculation for One Return	page 171
`crsp_ret_off_exch`	Marks Returns when it is Not Traded on the Exchange	page 172
`crsp_ret_ordinary`	Determines if a Distribution Is Considered Ordinary	page 172
`crsp_ret_payments`	Calculates Price Factor and Cash Dividend Amounts	page 172
`crsp_stk_ret_append_ts`	Appends Return to the `End` of the Returns Time Series	page 173
`crsp_stk_ret_append_dlret`	Appends Delisting Return to the Returns Time Series	page 173
`crsp_stk_delret_params`	Parses a Delisting Parameter File	page 173

crsp_ret_calc Stock Returns Calculations

PROTOTYPE:	`int crsp_ret_calc (CRSP_STK_STRUCT *stk, CRSP_TIMESERIES *p1, CRSP_TIMESERIES *p2, CRSP_TIMESERIES *r, CRSP_TIMESERIES *rn, int start, int end, int gapwindow, int validexch)`
DESCRIPTION:	general CRSP stock returns calculations, with and without dividends, allowing one or two price series for before/after, options on gap limits before considered missing, and valid exchanges.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure with names, distributions, and price data loaded `CRSP_TIMESERIES *p1` – time series of primary prices `CRSP_TIMESERIES *p2` – time series of secondary prices (`NULL` if unused) `CRSP_TIMESERIES *r` – time series to load total returns `CRSP_TIMESERIES *rn` – time series to load returns without dividends `int start, end` – index range to calculate returns `int gapwindow` – gap in periods before considered missing, use 0 for default (10 periods) `int validexch` – binary code for valid exchange codes 1=nyse, 2=NYSEMKT, 4=nasd, 8=arca, 0 = no restriction
RETURN VALUES:	`CRSP_SUCCESS`: if returns successfully loaded `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	Return time series objects are loaded with returns data. The `subtype` of these time series will be set to `CRSP_RETURN_NUM`. The `beg` and `end` ranges will be set according to start and `end` parameters and price range, so all previous returns ranges and data loaded will be erased. If start and `end` are outside of price ranges missing returns will be generated for the range outside of prices.
PRECONDITIONS:	It is assumed that `r` and `rn` have been allocated and have the same calendar as the price time series. One can be `NULL` if that type is not `wanted`. Prices, names and distribution histories must be loaded.

crsp_ret_calc_del CRSP Delisting Returns Calculations

PROTOTYPE:	`int crsp_ret_calc_del (CRSP_STK_STRUCT *stk, float *delret, float *delretx, float *effnewprc, int *effnewdt, int gapwindow, int crspnum, int setnum)`
DESCRIPTION:	CRSP stock delisting returns calculations. The delisting return is the return between the last price and the value of the stock after delisting, either based on the value given for the stock or the price on a new exchange. Returns are calculated with these steps: find if sufficient delisting information exists to calculate a return; if not, use the correct missing value. find a payment date and payment amount. The amount will either be the `dlprc`, the sum of final distributions, or the sum of both. The date will be the delist date + 1 period, or the `nextdt` if one is available. calculate a normal CRSP return between `endprc` and payment date, using `lastprc` and payment, using all distributions.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure `float *delret` – delisting return `float *delretx` – delisting return without dividends `float *effnewprc` – value after delisting `int *effnewdt` – date of value after delisting `int gapwindow` – gap in periods before considered missing `int crspnum, setnum` – database and set identifiers to load prices, these can be set to -99 if prices are loaded
RETURN VALUES:	`CRSP_SUCCESS`: if returns successfully loaded, `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	A delisting return with dividends is placed in `dlret`. A delisting return without dividends is placed in `dlretx`. An effective last payment is placed in `effnewprc`. The effective date of the last payment is placed in `effnewdt`. This will load the prices time series if prices are needed and they are not already loaded.
EXCEPTION CODES:	Exception codes (in order of precedence): `STK_RMISSR` – issue still active, `STK_RMISSD` – no sources to establish value after `delist`, `STK_RMISSG` – no acceptable previous price to calculate return, `STK_RMISSP` – trades on new exchange, but no price available.

`crsp_ret_calc_one` Returns Calculation for One Period

PROTOTYPE:	`float crsp_ret_calc_one (CRSP_ARRAY *di, float p1, float p2, float *rn, int start, int end)`
DESCRIPTION:	General CRSP stock returns calculation for one period given two prices, the dates of the two prices, and a distributions array. Total return is returned; return without dividends can be loaded by reference.
ARGUMENTS:	`CRSP_ARRAY *di` – stock distributions structure `float p1` – previous price `float p2` – current price `float *rn` – place to load returns without dividends (`NULL` if unwanted) `int start, end` – actual YYYYMMDD dates of `p1` and `p2` `int gapwindow` – gap in periods before considered missing
RETURN VALUES:	Total return `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	Returns without Dividends is loaded to `rn` if not `NULL`

`crsp_ret_off_exch` Marks Returns when Security is Not Traded on Valid Exchange

PROTOTYPE:	`int crsp_ret_off_exch (CRSP_ARRAY *nam, CRSP_TIMESERIES *r1, CRSP_TIMESERIES *r2, int start, int end, int validexch)`
DESCRIPTION:	uses the names history to mark returns from a time period when not on the desired exchange. Returns are marked as off exchange: during the effective range of a name structure that overlaps the returns range when the exchange code of that name structure is: `0` = (unknown) `1` = known but not one of CRSP-supported exchanges (NYSE, NYSEMKT, NASDAQ, ARCA) `2` = On one of these valid exchanges but not one part of the validexch binary code
ARGUMENTS:	`CRSP_ARRAY *nam` – names array `CRSP_TIMESERIES *r1, *r2` – returns and returns without dividends `int start, end` – effective range of returns to check `int validexch` – binary code of valid exchanges: 1 = NYSE, 2 = NYSEMKT, 4 = NASDAQ, 8 = ARCA, sum for combinations
RETURN VALUES:	`CRSP_SUCCESS`: `CRSP_FAIL`: if bad or missing parameters
PRECONDITIONS:	The two returns time series must be loaded or set to `NULL`.

`crsp_ret_ordinary` Determines if a Distribution Is Considered Ordinary

PROTOTYPE:	`int crsp_ret_ordinary (int code, float facpr)`
DESCRIPTION:	uses the distribution code and price factor to determine whether a distribution is considered ordinary for the purposes of the returns without dividends calculation
ARGUMENTS:	`int code` – 4-digit CRSP distribution code `float facpr` – CRSP distribution price factor
RETURN VALUES:	`1` if ordinary `0` if non-ordinary `2` to use factor
SIDE EFFECTS:	None

`crsp_ret_payments` Calculates Price Factor and Cash Dividend Amounts

PROTOTYPE:	`int crsp_ret_payments (double *t_fp, double *t_odiv, double *t_ndiv, CRSP_ARRAY *di, int dp, int date)`
DESCRIPTION:	calculates price factor and cash dividend amounts for a period using the distribution events array. It is passed a distribution array, a current event, and an ending date of the period. It cumulates information for all distributions in the period and returns the number of the distribution after the period.

ARGUMENTS:	`double *t_fp` – price factor for period `*t_odiv` – ordinary cash dividends for period `*t_ndiv` – non-ordinary cash dividends for period `CRSP_ARRAY *di` – distributions array `int dp` – current distribution event in array `int date` – ending calendar date of period the first three parameters are passed as pointers so they can be loaded with the result values
RETURN VALUES:	integer: current location in distributions array, this will be the first distribution after date
SIDE EFFECTS:	the parameters `t_fp`, `t_odiv`, and `t_ndiv` are set with period price factor, ordinary amount, and non-ordinary amount
CALL SEQUENCE:	Assumes `exdt`, `distcd` order

`crsp_stk_ret_append_ts` Appends Return to the End of the Returns Time Series

PROTOTYPE:	`int crsp_stk_ret_append_ts (CRSP_TIMESERIES *ret_ts, float ret, int date)`
DESCRIPTION:	appends return to the `end` of the returns time series
ARGUMENTS:	`CRSP_TIMESERIES *ret_ts` – pointer to return time series `float ret` – return to be appended to the `end` of return time series `int date` – date (YYYYMMDD) that the return is associated with
RETURN VALUES:	`CRSP_SUCCESS`: if return successfully appended `CRSP_FAIL`: if date does not follow existing returns range
SIDE EFFECTS:	The return is added to the returns time series on date. All periods between the previous `end` of returns and the date are loaded with missing values.
PRECONDITIONS:	`ret_ts` must be previously opened. Date must be at least as large as the last day when the return is not missing

`crsp_stk_ret_append_dlret` Appends Delisting Returns to the Returns Time Series

PROTOTYPE:	`int crsp_stk_ret_append_dlret (CRSP_STK_STRUCT *stk, CRSP_STK_DLSTCD_LIST *list)`
DESCRIPTION:	appends delisting returns to the returns time series
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure `CRSP_STK_DLSTCD_LIST *list` – user linked list of values to use as approximations for missing delisting returns of specified delist code ranges or exchanges.
RETURN VALUES:	`CRSP_SUCCESS`: if delist return successfully added `CRSP_FAIL`: if needed data not available or error in parameters
SIDE EFFECTS:	The delisting return is appended to the `end` of the returns time series and the delisting return without dividends is appended to returns without dividends time series. If the delisting returns are missing or contain partial month returns, the value can be adjusted from a user list of values. If the security matches the exchange code and delist code from the list and the delisting return is missing, the value from the list is used. If the security matches and the delisting return is a partial month return, the value from the list is compounded with the partial month return.
PRECONDITIONS:	The stock set must be previously loaded with events and returns arrays. The list can be loaded from a user file with the `crsp_stk_delret_params` function.

`crsp_stk_delret_params` Parses a Delisting Parameter File

PROTOTYPE:	`int crsp_stk_dlret_params (CRSP_STK_DLSTCD_LIST **itemlist, char *filename)`
DESCRIPTION:	Parses a delisting parameter file with information on user replacement values for missing delisting returns based on exchange or delist code. Each different exchange or delist code is represented in this file with a space delimited line with six fields. The fields are `beg` delisting code, `end` delisting code, `beg` exchange code, `end` exchange code, delisting return, and delisting return without dividends.
ARGUMENTS:	`CRSP_STK_DLSTCD_LIST **itemlist` – pointer to linked list that will be loaded in with replacement delisting returns information. `char *filename` – pointer to string containing path of delist returns parameter file.
RETURN VALUES:	`CRSP_SUCCESS`: if list is created successfully `CRSP_FAIL`: if an error in parsing arguments opening or reading file, or space allocation

SIDE EFFECTS:	*filename is opened for read, loaded, and then closed. Itemlist now points to a loaded linked list with delist parameters loaded.
PRECONDITIONS:	Itemlist should be set to NULL before starting. Filename must exist with read access in the format described above.

crsp_ret_map_payments Maps Adjustment Factors and Payments to a Time Series

PROTOTYPE:	int crsp_ret_map_payments(CRSP_STK_STRUCT *stk, CRSP_TIMESERIES *fp_ts, CRSP_TIMESERIES *odiv_ts, CRSP_TIMESERIES *ndiv_ts)
DESCRIPTION:	calculates payments over range based on distribution events array
ARGUMENTS:	CRSP_STK_STRUCT *stk – stock structure with events loaded CRSP_TIMESERIES *fp_ts – target ts of factor of adjust prices CRSP_TIMESERIES *odiv_ts – target ts of total ordinary dividend amount CRSP_TIMESERIES *ndiv_ts – target ts of total dividend amount It is assumed that at least one of the three target is not NULL and if more than one target exist then they have the same calendar
RETURN VALUES:	CRSP_SUCCESS: (integer) if returns successfully loaded CRSP_FAIL: if error in parameters or structures

Shares Outstanding Functions

FUNCTION	DESCRIPTION	PAGE
crsp_shr_imp	Converts Raw Shares to Imputed Shares to a CRSP Array	page 174
crsp_shr_reimp	Converts Raw Shares to Imputed Shares in Place	page 175
crsp_shr_num	Returns Shares Outstanding on a Given Date	page 179
crsp_shr_map	Maps The Imputed Shares Array to a Time Series	page 179
crsp_shr_raw	Converts Imputed Shares to Raw Shares	page 177

crsp_shr_imp Converts Raw Shares to Imputed Shares

PROTOTYPE:	int crsp_shr_imp (CRSP_STK_STRUCT *stk, CRSP_ARRAY *impshrs,int uniqflag, int skipflag, int firstflag)
DESCRIPTION:	general imputed CRSP stock shares function: given a standard stock structure with header and events structures, a pre-initialized CRSP_ARRAY is loaded with shares observations, including those imputed from distribution events. CRSPAccess stock databases are delivered with imputed shares already loaded. There are two options: the first is a flag that supports collapsing duplicate events so there is only one share observation on a given date; the second supports screening of certain types of distributions such as rights from affecting the shares outstanding results. This only uses ex-date of distributions.
ARGUMENTS:	CRSP_STK_STRUCT *stk – source data must have EVENTS loaded CRSP_ARRAY *impshrs – array that will be loaded. It must exist with enough space to store completed array of share events int uniqflag – flag for dates with multiple observations 0 – collapse structure so only the last observation on a date is left in the structure. Raw shares observations take precedence over derived ones 1 – allow multiple shares events on the same day. The last will be used by crsp_shr_map int skipflag – flag for skipping certain types of dists 1 – ignore facshr from rights 0 – use all facshrs int firstflag – flag for creating a dummy first observation 0 – do not create a dummy first observation 1 – copy first share structure up to begdt if available.
RETURN VALUES:	CRSP_SUCCESS: if shares array successfully loaded, CRSP_FAIL: if error in parameters or structures
SIDE EFFECTS:	The impshrs array is loaded with imputed shares structures and num is set to the number of shares observations found. shrflg is set with the following conventions: > 0 distribution event # (index-1 into dists array of facshr) 0 raw shares observation -1 implied 1st shrs observation (if dist with facshr precedes all raw shares observations, the first shrflg is –1 and the second > 0. -2 implied leading shares observation, where second is copied forward and shrsdt set to begdt. A value of 2 indicates an observation generated from a name change event. The shares outstanding for effective observation on the date of the name change is copied to the new observation and the observation is marked with a share flag of 2.
PRECONDITIONS:	The impshrs array must have arrtype CRP_STK_SHARE_NUM and subtype STK_SHARES_IMP

crsp_shr_reimp Converts Raw Shares to Imputed Shares in Place

PROTOTYPE:	`int crsp_shr_imp (CRSP_STK_STRUCT *stk, int skipflag)`
DESCRIPTION:	This function is similar to CRSP_SHR_IMP, but converts raw shares array to imputed shares in place instead of to a `CRSP_ARRAY` as the CRSP_SHR_IMP does. Stock structure must loaded with header and events structures.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – source data must have EVENTS loaded `int skipflag` – flag for skipping certain types of distributions 0: ignore `facshr` from rights 1: use all factors to adjust shares 2:The shares outstanding for effective observation on the date of the name change is copied to the new observation
RETURN VALUES:	`CRSP_SUCCESS`: if shares array successfully loaded, `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	The shares array is loaded with imputed shares structures and num is set to the number of shares observations found. `shrflg` is set to one digit number with the following conventions: 0: raw shares observation 1: shares observation implied by distribution events 2: shares observation implied by names change events.
PRECONDITIONS:	Shares must be loaded. The subflag of shares_arr must reflect whether raw (`CRSP_SHARES_RAW_NUM=20`) or imputed (`CRSP_SHARES_IMP_NUM=0`) shares are currently loaded.

crsp_shr_num Returns Shares Outstanding on a Given Date

PROTOTYPE:	`int crsp_shr_num (CRSP_STK_STRUCT *stk, int date, int skipflag, CRSP_STK_SHARE *share)`
DESCRIPTION:	returns the shares outstanding on a given date. There is an optional parameter that can return the actual observation date of the shares outstanding result. Uses `crsp_shr_imp` to build a static array of imputed shares. If the PERMNO is the same, the array is not rebuilt.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – source data must have `EVENTS` and `HEADER` loaded `int date` – yymmdd or yyyymmdd date to find shares out `int skipflag` – flag for skipping certain types of `dists` 1 – ignore `facshr` from rights 0 – use all `facshrs` `CRSP_STK_SHARE *shares_obs` – shares info of actual observation used if set to `NULL` will not be loaded
RETURN VALUES:	`CRSP_SUCCESS`: number of shares outstanding effective on date, 0 if no shares structures or date out of data range `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	If the fourth parameter is passed, it is loaded with the information from the effective shares event.

crsp_shr_map Maps The Imputed Shares Array to a Time Series

PROTOTYPE:	`int crsp_shr_map (CRSP_STK_STRUCT *stk, CRSP_TIMESERIES *shr_ts, int begind, int endind, int skipflag)`
DESCRIPTION:	maps the imputed shares to a time series. Uses `crsp_shr_imp` to load an imputed shares events array if necessary, then maps the observations by finding the effective shares outstanding for each date in the calendar.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure loaded with HEADER and EVENTS `CRSP_TIMESERIES *shr_ts` – pre-initialized time series that will be loaded. Must have array allocated at least up to `endind` and calendar set. `int begind, endind` – range of indexes into calendar that will be loaded to shrs `int skipflag` – flag for skipping certain types of `dists` 1 – ignore `facshr` from rights 0 – use `facshr` loaded with shares
RETURN VALUES:	`CRSP_SUCCESS`: (integer) if shares successfully loaded `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	shrs time series is loaded. arr is filled with shares outstanding values and `beg` and `end` are set. If there are no shares, structures `beg` and `end` are set to 0; otherwise they inherit parameters `begind` and `endind`.
PRECONDITIONS:	The `shr_ts` time series must have `arrtype` `CRSP_INTEGER_NUM` and `subtype` `CRSP_SHARES_IMP_NUM`

`crsp_shr_raw` Converts Imputed Shares Into Raw Shares Observations

PROTOTYPE:	`int crsp_shr_raw (CRSP_ARRAY *shr_arr)`
DESCRIPTION:	converts imputed shares outstanding events in raw shares. Imputed shares are observations directly derived from CRSP distribution events. These are removed from the shares outstanding observation array.
ARGUMENTS:	`CRSP_ARRAY *shr_arr` – `CRSP_ARRAY` of imputed shares already loaded
RETURN VALUES:	`CRSP_SUCCESS`: if shares successfully modified `CRSP_FAIL`: if error in parameters or structures
SIDE EFFECTS:	Imputed shares array is converted to raw shares, from `subtype STK_SHARES_IMP` to `STK_SHARES_RAW`

SUBSET FUNCTIONS

These functions are used to perform subsetting of stock data based on exchange, share type, NASDAQ market listing, or when-issued status.

FUNCTION	DESCRIPTION	PAGE
`crsp_stk_subset_all`	calls the indicated restriction functions	page 176
`crsp_stk_subset_exch`	restricts a stock structure by exchange	page 181
`crsp_stk_subset_shrcd`	restricts a stock structure by share code	page 195
`crsp_stk_subset_range`	restricts a stock structure by date range	page 178
`crsp_stk_subset_nmsind`	restricts a stock structure by NASDAQ National Market status	page 187
`crsp_stk_subset_wi`	restricts a stock structure by when-issued status	page 179
`crsp_stk_subset_freq`	maps data with a new frequency into a new stock structure	page 189
`crsp_stk_subset_parload`	loads a structure of subset parameters (a CRSP_UNIV_PARAM_LOAD structure) used by other subset functions	page 181
`crsp_stk_gen_sum_nasdin`	summarizes NASDAQ market maker count	page 181

`crsp_stk_subset_all` Calls the Indicated Restriction Functions

PROTOTYPE:	`int crsp_stk_subset_all (CRSP_STK_STRUCT *stk, int crspnum, int setid, CRSP_UNIV_PARAM_LOAD *subpar, char *stat)`
DESCRIPTION:	Calls other stock restriction functions based on a parameter structure loaded with desired subsetting options.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure to restrict `int crspnum` – database handle returned by `crsp_stk_open`. `int setid` – set identifier used in call to open and read the stock structure. `CRSP_UNIV_PARAM_LOAD *subpar` – pointer to structure containing restriction parameters. See `crsp_stk_subset_parload` on page 181 for details of this structure. `char *stat` – pointer to location to store two-letter code indicating the return status of the restriction. The codes are: `DR` if restricted or eliminated because of date restriction `EX` if restricted or eliminated because of exchange restriction `SH` if restricted or eliminated because of share code restriction `NM` if restricted or eliminated because of NMS code restriction `W1` if restricted or eliminated because of when-issued type 1 restriction `W2` if restricted or eliminated because of when-issued type 2 restriction `W3` if restricted or eliminated because of when-issued type 3 restriction `OK` if return 1 and no header variables changed `O#` if return 1 and header variables have changed
RETURN VALUES:	`1`: if stock structure successfully restricted and valid data remains `0`: if success but issue is totally erased by some restriction `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The `stk` structure is modified if partially restricted by one of the subset functions. Price time series data may be loaded if needed to identify ranges of data to delete. The `stat` character string will be set to a string based on the changes made to the security data.

PRECONDITIONS:	The `subpar` structure must be loaded with the parameters specifying the restrictions to make. The `stk` structure must be opened with at least header, events, and price modules, and header and events modules must be loaded. The `stat` pointer must point to at least three bytes of allocated memory.

`crsp_stk_subset_exch` Restricts Stock Data by Exchange Code

PROTOTYPE:	`int crsp_stk_subset_exch (CRSP_STK_STRUCT *stk, int crspnum, int setid, int nameflag, int shareflag, int wantexch, int subflag)`
DESCRIPTION:	Restricts stock data based on exchange code. This function uses the Exchange Code in the name structures to decide which exchange the issue is listed on, at what time. The `wanted` exchanges are specified with a binary code: 1=NYSE, 2=NYSEMKT, 4=NASDAQ, 8=ARCA. When-issued time periods with 3 prefixes are treated as the base exchange for purposes of this function. Suspends and halts are treated as the previous exchange. `Wanted` exchange is the exchange(s) that data will be restricted to. This restricts by delist date before using the names. It moves price data back to the last delist structure if prices exist after delist. It moves delist date back to prices if prices end before delisting. It adjusts delistings – it creates a 500 delist if there is any invalid name after the last valid name and before the old delist date.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure to restrict `int crspnum` – database handle returned by `crsp_stk_open`. `int setid` – set identifier used in call to open and read the stock structure. `int nameflag` – code that determines how name records are handled in the restricted structure. 0 = keep all name structures 1 = delete name structures out of range `int shareflag` – code that determines how shares outstanding observations out of range are handled: 0 = keep no shares observations out of range 1 = keep shares out of range that are applicable to the range 2 = keep shares out of range that are applicable to range only if there are no shares in the range, this shares structure precedes the range, and the range begins with the first name structure for the issue with a valid exchange `int wantexch` – code of exchanges to keep. The values below can be added together to select multiple exchanges. 1 = NYSE 2 = NYSEMKT 4 = NASDAQ 8 = ARCA `int subflag` – subset flag 0 = subset data during range 1 = if ever not valid, delete entire issue 2 = if ever valid make no restrictions
RETURN VALUES:	`CRSP_SUCCESS`: if stock structure successfully restricted and valid data remains `CRSP_NOT_FOUND`: if excluded because never on valid exchanges `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The `stk` structure is modified according to flags if partially restricted. Price time series data may be loaded if needed to identify ranges of data to delete.
PRECONDITIONS:	The stk structure must be opened with at least header, events, and price modules, and header and events modules must be loaded.

`crsp_stk_subset_shrcd` Restricts Stock Data by Share Code

PROTOTYPE:	`int crsp_stk_subset_shrcd (CRSP_STK_STRUCT *stk, CRSP_UNIV_SHRCD *scs, int nameflag, int shareflag, int subflag)`
DESCRIPTION:	Restricts stock data based on share code. This function uses the `shrcd` in the name structures to decide the issue's share code over time. The share code is a two-digit number where each digit separately contains information classifying the type of share. The function allows specification of one or more valid first digits and one or more valid second digits in deciding which share codes are valid. This function adjusts delistings. It creates a 500 delist if there is any invalid name after the last valid name and before the old delist date.

ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure to restrict `CRSP_UNIV_SHRCD *scs` – pointer to share code restriction structure. There are two required fields in the structure that must be set to define the restriction. The fields are: `fstdig` – bit map of valid first digits of share code. If the n'th bit of `fstdig` is a 1, the share code `n*` is considered valid. Bit positions used in the bit map are the right-most 10 bits, numbered left to right, beginning at 0. `secdig` – bit map of valid second digits of share code. If the n'th bit of `secdig` is a 1, the share code `*n` is considered valid. Bit positions used in the bit map are the right-most 10 bits, numbered left to right, beginning at 0. `int nameflag` – code that determines how name records are handled in the restricted structure. 0 = keep all name structures 1 = delete name structures out of range `int shareflag` – code that determines how shares outstanding observations out of range are handled: 0 = keep no shares observations out of range 1 = keep shares out of range that are applicable to the range 2 = keep shares out of range that are applicable to range only if there are no shares in the range, this shares structure precedes the range, and the range begins with the first valid exchcd name structure for the issue `int subflag` – subset flag 0 = subset data during range 1 = if ever not valid, delete entire issue 2 = if ever valid make no restrictions
RETURN VALUES:	`CRSP_SUCCESS`: if stock structure successfully restricted and valid data remains `CRSP_NOT_FOUND`: if excluded because never had valid share code `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The `stk` structure is modified according to flags if partially restricted.
PRECONDITIONS:	The `stk` structure must be opened with at least header and events modules, and header and events modules must be loaded.

`crsp_stk_subset_range` Restricts Stock Data by Date Range

PROTOTYPE:	`int crsp_stk_subset_range (CRSP_STK_STRUCT *stk, int begdata, int enddata, int nameflag, int shareflag)`
DESCRIPTION:	Restricts stock data based on date ranges.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure to restrict `int begdata` – beginning date in YYYYMMDD format of restricted data. `int enddata` – ending date in YYYYMMDD format of restricted data. `int nameflag` – code that determines how name records are handled in the restricted structure: 0 = keep all name structures 1 = delete name structures out of range `int shareflag` – code that determines how shares outstanding observations out of range are handled: 0 = keep no shares observations out of range 1 = keep shares out of range that are applicable to the range 2 = keep shares out of range that are applicable to range only if there are no shares in the range, this shares structure precedes the range, and the range begins with the first valid exchcd name structure for the issue
RETURN VALUES:	`CRSP_SUCCESS`: if included and stock structure successfully restricted `CRSP_NOT_FOUND`: if excluded because never had data within range `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The `stk` structure is modified according to flags if partially restricted.
PRECONDITIONS:	The `stk` structure must be opened with at least header and events modules, and header and events modules must be loaded.

`crsp_stk_subset_nmsind` Restricts Stock Data by NASDAQ Market

PROTOTYPE:	`int crsp_stk_subset_nmsind (CRSP_STK_STRUCT *stk, int crspnum, int setid, int nmsflag, int shareflag, int subflag)`
DESCRIPTION:	Restricts stock data based on NASDAQ market listing. This function uses Exchange Code and NASDAQ National Market Indicator to decide whether the issue is listed on NASDAQ, and if so, which NASDAQ market it is listed on. Only NASDAQ issues are affected by this function. The NASDAQ National Market and SmallCap designations were introduced in 1992. The NASDAQ National Market, originally called the National Market System, was introduced in 1984. Before June 15, 1992, issues not listed on the National Market System were not required to report trades. NASDAQ introduced a 3-tier market initiative in July 2006. As a result, the CRSP NASDAQ National Market Indicator (NMSIND) coding scheme was changed. After July 1, 2006, SmallCap is renamed to Capital Market. National Market is split into two: Global Market and Global Select Market.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – stock structure to restrict. `int crspnum` – database handle returned by `crsp_stk_open`. `int setid` – set identifier used in call to open and read the stock structure. `int nmsflag` – code used to specify valid NASDAQ markets: 1 = erase data if nmsind is not 2, 5 or 6 (keep National Market and Global and Global Select Markets only) 2 = erase data if nmsind is 2, 5 or 6 (keep SmallCap and Capital Market only) 3 = erase data if nmsind is 1 (keep all NASDAQ markets with price reporting) 4 = erase data if nmsind is not 1 (keep SmallCap before June 15, 1992) 5 = erase data if nmsind is not 2 or 6 (keep National Market and Global Select Market) 6 = erase data if nmsind is not 2 or 5 (keep National Market and Global Market only) 7 = erase data if nmsind is not 6 (keep Global Select Market only) `int shareflag` – code that determines how shares outstanding observations out of range are handled: 0 = keep no shares observations out of range 1 = keep shares out of range that are applicable to the range 2 = keep shares out of range that are applicable to range only if there are no shares in the range, this shares structure precedes the range, and the range begins with the first valid exchcd name structure for the issue `int subflag` – subset flag 0 = subset data during range 1 = if ever not valid, delete entire issue 2 = if ever valid make no restrictions
RETURN VALUES:	`CRSP_SUCCESS`: if stock structure successfully restricted and valid data remains `CRSP_NOT_FOUND`: if excluded because never on valid exchanges `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The `stk` structure is modified according to flags if partially restricted. Price time series data may be loaded if needed to identify ranges of data to delete.
PRECONDITIONS:	The `stk` structure must be opened with at least header, events, and price modules, and header and events modules must be loaded.

`crsp_stk_subset_wi` Restricts Stock Data by When-Issued Status

PROTOTYPE:	`int crsp_stk_subset_wi (CRSP_STK_STRUCT *stk, int wiflag, int shareflag)`
DESCRIPTION:	Restricts stock data based on when-issued status of an issue. CRSP classifies when-issued trading into three categories: Type 1 = when-issued trading for new issues before regular-way trading. Type 2 = ex-distribution – simultaneous trading of post-distribution shares before the distribution is official. Type 3 = when-issued trading during a reorganization or bankruptcy proceedings when the market expects the security to return to regular status. On type 3 cases, names are not erased, but modified. NASDAQ 5th character V's are dropped and the exchange code has 30 subtracted. They cannot be dropped because they are usually accompanied by a CUSIP change. Only type 3 cases are present on CRSP subscriber files.

ARGUMENTS:	CRSP_STK_STRUCT *stk – pointer to stock structure to restrict int wiflag – code that determines which restrictions are made. Possible codes are: 1 = ignore type 1 when-issued cases, erase range, erase name structures 2 = ignore type 1 when-issued cases, erase range, keep name structures 3 = ignore type 2 when-issued cases, delete entire issue 4 = ignore type 3 when-issued cases, erase range, keep name structures 5 = ignore type 3 when-issued cases, keep range, erase name structure 6 = ignore type 3 when-issued cases, erase range, erase name structure int shareflag – code that determines how shares outstanding observations out of range are handled: 0 = keep no shares observations out of range, 1 = keep shares out of range that are applicable to the range
RETURN VALUES:	CRSP_SUCCESS: if stock structure successfully restricted and valid data remains CRSP_NOT_FOUND: if excluded because never had valid data within range CRSP_FAIL: if error in parameters or processing
SIDE EFFECTS:	The stk structure is modified according to flags if partially restricted.
PRECONDITIONS:	The stk structure must be opened with at least header and events modules, and header and events modules must be loaded.

crsp_stk_subset_freq Converts Stock Data to a Different Time Series Frequency

PROTOTYPE:	int crsp_stk_subset_freq (CRSP_STK_STRUCT *dstk, CRSP_STK_STRUCT *mstk, CRSP_UNIV_SUM *summ)
DESCRIPTION:	Copies stock data for one security into a new structure with converted time series calendar frequencies. The rules used are based on an input structure of summary specifications. Event data are copied as is.
ARGUMENTS:	CRSP_STK_STRUCT *dstk – pointer to input stock structure CRSP_STK_STRUCT *mstk – pointer to output stock structure CRSP_UNIV_SUM *summ – pointer to structure with summary rules for conversion. The following fields in the summary structure are used: sum_prc – specifications for loading Closing Price or Bid/Ask Average 0 = last price or bid/ask average of source in period 1 = average price or bid/ask average of source over period 2 = median price or bid/ask average of source over period. 3 = no prices are loaded 4 = nonmissing price or bid/ask average on the day closest to the last date of the period, within the range of the target period. sum_sp – specifications for loading Bid or Low and Ask or High 0 = last bid or low and last ask or high 1 = lowest bid or low and highest ask or high 2 = lowest price or bid/ask average and highest price or bid/ask average 3 = no bid or low or ask or high data sum_vol – specifications for loading Volume 0 = last volume in period 1 = sum of all volumes in period divided by the sum_volume factor constant. 2 = average of volumes in period 3 = median of volumes in period 4 = no volumes sum_ret – specifications for loading returns 0 = no returns loaded 1 = compound Total Returns in period 2 = compound Total Returns and Returns without Dividends in period sum_spread – specifications for loading spread or other secondary time series 0 = spread on last day of period, calculated from bid and ask prices if last date has no trading price 1 = load no spread, alternate price, bid, or ask time series 2 = set Spread, Bid, and Ask based on last day of period, Number of Trades to total number of trades in period, and Price Alternate to last nonmissing price or bid-ask average in period 3 = set Price Alternate to last nonmissing price in period, and Number of Trades to the Price Alternate Date. 4 = set Bid and Ask to the last value in the period, and set the Number of Trades to the sum of trades in the period.
RETURN VALUES:	number of periods in the resultant price time series for the converted security CRSP_FAIL: if error in parameters or processing

SIDE EFFECTS:	The `mstk` structure is loaded with converted data.
PRECONDITIONS:	The `dstk` and `mstk` structures must be opened with at least header, events, and prices modules, and at least header, events, and prices modules must be loaded in the input stock structure. The summary structure must be loaded with valid specifications. If adjusted results are desired, the input stock structure must be adjusted before calling this function.

`crsp_stk_subset_parload` Loads Subsetting Parameters from a File

PROTOTYPE:	`int crsp_stk_subset_parload (CRSP_UNIV_PARAM_LOAD *subpar, char *parfile)`
DESCRIPTION:	Loads a subsetting parameter structure from an input file containing subsetting options. See below for the available options and format of the input file.
ARGUMENTS:	`CRSP_UNIV_PARAM_LOAD *subpar` – pointer to subset parameter structure to be loaded. `char *parfile` – pointer to string containing the path of the parameter input file. The input file must contain text with one or more rows of specifications. Each row must contain one parameter keyword and a corresponding value, separated by spaces. (see Parameter Options Specifications for crsp_stk_subset utility program CUPL Guide, for description of Parameter options file)
RETURN VALUES:	`CRSP_SUCCESS`: if parameters successfully loaded `CRSP_FAIL`: if error in parameters or processing
SIDE EFFECTS:	The subpar structure is loaded with parameter data. The input file is opened, loaded, and closed.
PRECONDITIONS:	The input file must exist in the proper format. The subpar pointer must point to an allocated `CRSP_UNIV_PARAM_LOAD` structure.

CRSPAccess C Stock General Data Utility Functions

These functions are used to make general data summaries of stock data.

`crsp_stk_gen_sum_nasdin` Summarizes NASDAQ Information Events

PROTOTYPE:	`int crsp_stk_gen_sum_nasdin (CRSP_ARRAY *nasdin_arr, int pct)`
DESCRIPTION:	Summarizes NASDAQ Information histories by eliminating events when the only change is the number of market makers and the change is smaller than a certain amount. The limit of change is passed as an integer percentage.
ARGUMENTS:	`CRSP_ARRAY *nasdin_arr` – pointer to NASDAQ Information array to restrict. `int pct` – minimum percentage change in Market Maker Count compared to previous before observation is kept.
RETURN VALUES:	`CRSP_SUCCESS`: if array successfully summarized `CRSP_FAIL`: if error in parameters
SIDE EFFECTS:	The `nasdin_arr` structure is modified according to the percentage parameter. The kept rows are shifted up and the num counter is adjusted to reflect the remaining number of observations.
PRECONDITIONS:	The `nasdin_arr` array must be allocated with `arrtype` = 55 and loaded data.

`crsp_stk_gen_hdr_fromnam` Resets Header Identification Information

PROTOTYPE:	`int crsp_stk_gen_hdr_fromnam (CRSP_STK_STRUCT *stk)`
DESCRIPTION:	Resets header identification information in a stock structure using the names array.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure to modify
RETURN VALUES:	`CRSP_SUCCESS`: if stock structure successfully summarized `CRSP_FAIL`: if error in parameters or structure not loaded
SIDE EFFECTS:	The stock structure header structure is modified by the names array
PRECONDITIONS:	The stock structure must be allocated, opened with at least headers and events, and loaded with at least headers and events.

CRSPAccess C Stock Delete Range Data Utility Functions

These functions are used to delete ranges of stock data.

FUNCTION	DESCRIPTION	PAGE
crsp_stk_delrng_all	Delete ranges of data from stock structure	page 200
crsp_stk_delrng_names	Delete ranges of data from names array	page 182
crsp_stk_delrng_dists	Delete ranges of data from distribution array	page 192
crsp_stk_delrng_nasdin	Delete ranges of data from NASDAQ Information array	page 183
crsp_stk_delrng_groups	Delete ranges of data from groups array	page 184
crsp_stk_delrng_delists	Delete ranges of data from delisting array	page 183
crsp_stk_delrng_shares	Delete ranges of data from shares array	page 184
crsp_stk_delrng_resetdt	Reset the header beginning and ending dates	page 184

crsp_stk_delrng_all Deletes Ranges of Stock Data

PROTOTYPE:	`int crsp_stk_delrng_all (CRSP_STK_STRUCT *stk, int beg_date, int end_date, int data_beg, int namflg, int shrflg, int ndiflg)`
DESCRIPTION:	Deletes ranges of data from a stock structure by calling other delete range functions.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure to restrict `int beg_date` – beginning date of delete range, in YYYYMMDD format. `beg_date` is not deleted. `int end_date` – ending date of delete range, in YYYYMMDD format. `end_date` is not deleted. `int data_beg` – first date of prices before restriction, in YYYYMMDD format `int namflg` – code to determine how name history is modified by restrictions 0 = names events are not deleted 1 = names events are to be deleted, and data exists after the last name 2 = names events are to be deleted, and data does not exist after the last name `int shrflg` – code to determine how shares observations are modified by restrictions 0 = delete any shares observations in the range 1 = keep any shares observations in the range that apply outside the range `int ndiflg` – code to determine how the NASDAQ Information history is modified by restrictions 0 = no NASDAQ Information event deletions 1 = NASDAQ Information events can be deleted
RETURN VALUES:	0 = if there are no data after the deletion 1 = if there are events data after the deletion 2 = if there are time series data after the deletion 3 = if there are time series and events data after the deletion `CRSP_FAIL`: if error in parameters
SIDE EFFECTS:	The `stk` structure is modified according to the other parameters.
PRECONDITIONS:	The `stk` structure must be allocated and loaded with at least header, events, and price data.

crsp_stk_delrng_names Deletes Ranges of Stock Names Data

PROTOTYPE:	`int crsp_stk_delrng_names (CRSP_ARRAY *names_arr, int beg_date, int end_date, int namflg)`
DESCRIPTION:	Deletes ranges of stock names data
ARGUMENTS:	`CRSP_ARRAY *names_arr` – pointer to names array to restrict `int beg_date` – beginning date of delete range, in YYYYMMDD format. `beg_date` is not deleted. `int end_date` – ending date of delete range, in YYYYMMDD format. `end_date` is not deleted. `int namflg` – code to determine how name history is modified by restrictions 0 = names events are not deleted 1 = names events are to be deleted, and data exists after the last name 2 = names events are to be deleted, and data does not exist after the last name

RETURN VALUES:	0 = if there are no data after the deletion 1 = if there are names data after the deletion CRSP_FAIL: if error in parameters
SIDE EFFECTS:	The names_arr array is modified according to the other parameters.
PRECONDITIONS:	The names_arr array must be allocated and loaded with arrtype = 54.

crsp_stk_delrng_dists Deletes Ranges of Stock Distribution Data

PROTOTYPE:	int crsp_stk_delrng_dists (CRSP_ARRAY *dists_arr, CRSP_ARRAY *delist_arr, int beg_date, int end_date)
DESCRIPTION:	Deletes ranges of stock distribution data. If the delisting date is in the range to delete, all final distributions are also removed. Ex-Distribution date of distributions is used in restrictions.
ARGUMENTS:	CRSP_ARRAY *dists_arr – pointer to loaded distributions array to restrict CRSP_ARRAY *delist_arr – pointer to loaded delisting array int beg_date – beginning date of delete range, in YYYYMMDD format. beg_date is not deleted. int end_date – ending date of delete range, in YYYYMMDD format. end_date is not deleted.
RETURN VALUES:	0 = if there are no distributions data after the deletion 1 = if there are distributions data after the deletion CRSP_FAIL: if error in parameters
SIDE EFFECTS:	The dists_arr array is modified according to the other parameters.
PRECONDITIONS:	The dists_arr array must be allocated and loaded with arrtype = 52. The delist_arr array must be allocated and loaded with arrtype = 54.

crsp_stk_delrng_delists Deletes Ranges of Stock Delisting Data

PROTOTYPE:	int crsp_stk_delrng_delists (CRSP_ARRAY *delist_arr, int beg_date, int end_date)
DESCRIPTION:	Deletes ranges of stock delisting data. If no delisting events remain, one is added and coded as active.
ARGUMENTS:	CRSP_ARRAY *delist_arr – pointer to loaded delisting array to modify. int beg_date – beginning date of delete range, in YYYYMMDD format. beg_date is not deleted. int end_date – ending date of delete range, in YYYYMMDD format. end_date is not deleted.
RETURN VALUES:	1 = if there is delisting data after the deletion CRSP_FAIL: if error in parameters
SIDE EFFECTS:	The delist_arr array is modified according to the other parameters.
PRECONDITIONS:	The delist_arr array must be allocated and loaded with arrtype = 54

crsp_stk_delrng_nasdin Deletes Ranges of Stock NASDAQ Information Data

PROTOTYPE:	int crsp_stk_delrng_nasdin (CRSP_ARRAY *nasdin_arr, int beg_date, int end_date)
DESCRIPTION:	Deletes ranges of NASDAQ Information data.
ARGUMENTS:	CRSP_ARRAY *nasdin_arr – pointer to loaded NASDAQ Information array to restrict. int beg_date – beginning date of delete range, in YYYYMMDD format. beg_date is not deleted. int end_date – ending date of delete range, in YYYYMMDD format. end_date is not deleted.
RETURN VALUES:	0 = if there is no NASDAQ Information data after the deletion 1 = if there is NASDAQ Information data after the deletion CRSP_FAIL: if error in parameters
SIDE EFFECTS:	The nasdin_arr array is modified according to the other parameters.
PRECONDITIONS:	The nasdin_arr array must be allocated and loaded with arrtype = 55.

crsp_stk_delrng_shares Deletes Ranges of Stock Shares Outstanding Data

PROTOTYPE:	`int crsp_stk_delrng_shares (CRSP_ARRAY *shares_arr, int beg_date, int end_date, int data_beg int keepflg)`
DESCRIPTION:	Deletes ranges of shares outstanding observation data.
ARGUMENTS:	`CRSP_ARRAY *shares_arr` – pointer to loaded shares array to restrict `int beg_date` – beginning date of delete range, in YYYYMMDD format. `beg_date` is not deleted. `int end_date` – ending date of delete range, in YYYYMMDD format. `end_date` is not deleted. `int data_beg` – first date of prices before restriction, in YYYYMMDD format `int keepflg`: 0 will delete all shares within delete range based on observation date only – the first will be kept if it applies to outside the range 1 will keep any observations that apply to data before and after the delete range
RETURN VALUES:	0 = if there are no shares data after the deletion 1 = if there are shares data after the deletion `CRSP_FAIL`: if error in parameters
SIDE EFFECTS:	The `shares_arr` array is modified according to the other parameters. The shares array is modified by removing rows to beginning and end and splitting or setting existing rows to shares outstanding = 0 when a range is removed not on an edge. This may change the observation dates based on the subset dates. All adjacent shares outstanding 0 gaps are consolidated.
PRECONDITIONS:	The `shares_arr` array must be allocated and loaded with `arrtype` = 53. The function expects input of shares loaded with shsenddts (shares `end` dates) set.

crsp_stk_delrng_groups Deletes Ranges of Stock Group

PROTOTYPE:	`int crsp_stk_delrng_groups (CRSP_ARRAY **groups_arr, int grouptypes, int beg_date, int end_date)`
DESCRIPTION:	Deletes ranges of stock group data for all types.
ARGUMENTS:	`CRSP_ARRAY **groups_arr` – pointer to array of pointers to loaded group arrays to restrict `int grouptypes` – the number of group arrays in the array of pointers `int beg_date` – beginning date of delete range, in YYYYMMDD format. `beg_date` is not deleted. `int end_date` – ending date of delete range, in YYYYMMDD format. `end_date` is not deleted.
RETURN VALUES:	0 = if there are no group data after the deletion 1 = if there are group data after the deletion `CRSP_FAIL`: if error in parameters
SIDE EFFECTS:	All the `group_arr` arrays are modified according to the other parameters.
PRECONDITIONS:	The `group_arr` arrays must be allocated and loaded with `arrtype` = 57.

crsp_stk_delrng_resetdt Resets Header Date Ranges from Time Series

PROTOTYPE:	`int crsp_stk_delrng_resetdt (CRSP_STK_STRUCT *stk)`
DESCRIPTION:	Resets header begdt and `enddt` fields from available time series in the stock structure.
ARGUMENTS:	`CRSP_STK_STRUCT *stk` – pointer to stock structure
RETURN VALUES:	0 = if there are no time series data and ranges are set to 0 1 = if there are time series data after the deletion
SIDE EFFECTS:	The `stk` structure begdt and `enddt` are modified.
PRECONDITIONS:	The `stk` structure must be allocated with at least header data opened. The ranges will only be set based on the time series loaded in the stock structure.

CRSPAccess C Stock Valid Data Utility Functions

These functions are used to determine whether data are valid for different filtering criteria.

FUNCTION	DESCRIPTION	PAGE
`crsp_stk_valid_exchcd`	Determines if exchange code is valid	page 185
`crsp_stk_valid_nmsind`	Determines if NASDAQ National Market Indicator is valid	page 185
`crsp_stk_valid_shrcd`	Determines if Share Code is valid	page 193
`crsp_stk_valid_shrcd_ld`	Sets up a `CRSP_UNIV_SHRCD` structure for checking valid share codes	page 186

`crsp_stk_valid_exchcd` Determines if Exchange Code is Valid

PROTOTYPE:	`int crsp_stk_valid_exchcd (int exhave, int exwant)`
DESCRIPTION:	Determines if a given exchange code is valid based on a set of `wanted` exchanges. When-issued trading is not differentiated from regular-way trading.
ARGUMENTS:	`int exhave` – Exchange Code to validate. Codes are standard CRSP stock Exchange Codes: 1=NYSE 2=NYSEMKT 3=NASDAQ 4=ARCA 31=NYSE when-issued 32=NYSEMKT when-issued 33=NASDAQ when-issued 34=ARCA when-issued `int exwant` – acceptable Exchange Code or codes. If multiple exchanges are valid, `exwant` is the sum of the individual codes below: 1=NYSE 2=NYSEMKT 4=NASDAQ 8=ARCA
RETURN VALUES:	0 = if exhave is valid according to exwant -1 = if exhave is not valid according to exwant
SIDE EFFECTS:	none
PRECONDITIONS:	none

`crsp_stk_valid_nmsind` Determines if NASDAQ National Market Indicator is Valid

PROTOTYPE:	`int crsp_stk_valid_exchcd (int nmscode, int nmsind)`
DESCRIPTION:	Determines if a given NASDAQ National Market Indicator code is valid based on a set of valid codes.
ARGUMENTS:	`int nmscode` – acceptable NASDAQ National Market Indicator Code. Codes are: 1 = invalid if NASDAQ National Market Indicator Code is not 2, 5 or 6 (only National Market and Global and Global Select Markets are valid) 2 = invalid if NASDAQ National Market Indicator Code is 2, 5 or 6 (only SmallCap and Capital Market are valid) 3 = invalid if NASDAQ National Market Indicator Code is 1 (all NASDAQ markets with price reporting are valid) 4 = invalid if NASDAQ National Market Indicator Code is not 1 (only SmallCap before June 15, 1992 is valid) 5 = invalid if NASDAQ National Market Indicator Code is not 2 or 6 (only National Market and Global Select Market are valid) 6 = invalid if NASDAQ National Market Indicator Code is not 2 or 5 (only National Market and Global Market are valid) 7 = invalid if NASDAQ National Market Indicator Code is not 6 (only Global Select Market is valid) `int nmsind` – actual NASDAQ National Market Indicator to validate.
RETURN VALUES:	0 = if `nmsind` is valid according to nmscode
	-1 = if `nmsind` is not valid according to nmscode
SIDE EFFECTS:	none
PRECONDITIONS:	none

crsp_stk_valid_shrcd Determines if Share Code is Valid

PROTOTYPE:	`int crsp_stk_valid_shrcd (CRSP_UNIV_SHRCD *scs, int shrcd)`
DESCRIPTION:	Determines if a given share code is valid based on a map of acceptable first and second digits of the CRSP Share Code.
ARGUMENTS:	`CRSP_UNIV_SHRCD *scs` – loaded structure containing information of valid Share Codes. Desired codes are loaded as bit maps into two fields in the structure, `fstdig` for the first Share Code digit, and scddig for the second Share Code digit. The bit map fields are loaded so that the right-most 10 bits n to n+9 are set. If the nth bit is set to 1 then the Share Code digit n is valid. If the nth bit is set to 0 then the Share code digit n is invalid. See the function `crsp_stk_valid_shrcd_ld` to load this structure. `int shrcd` – actual Share Code to validate.
RETURN VALUES:	0 = if `shrcd` is valid according to the scs structure -1 = if `shrcd` is not valid according to the scs structure
SIDE EFFECTS:	none
PRECONDITIONS:	none

crsp_stk_valid_shrcd_ld Loads a Structure Used to Specify Valid Share Codes

PROTOTYPE:	`int crsp_stk_valid_shrcd (CRSP_UNIV_SHRCD *scs, int sc_code, char *leftdig, char *rightdig)`
DESCRIPTION:	This function sets up a `CRSP_UNIV_SHRCD` structure used by `crsp_stk_valid_shrcd`. It is passed a pointer to the structure, a code of possible subsets, and two strings of flags to specify subsets by digits. Certain codes are supported automatically. These are described below.
ARGUMENTS:	`CRSP_UNIV_SHRCD *scs` – pointer to structure to load with valid share code criteria `int sc_code` – code describing a standard or user-defined set of restrictions. Available codes are: `CRSP_SUB_SCNY(=1)` –CRSP NYSE and NYSEMKT standard restrictions; first digit 1,2,3,4,7 allowed, all second digits allowed except 6 and 7 `CRSP_SUB_SCNQ(=2)` –CRSP NASDAQ standard restrictions; same but also exclude second digit 2 and 5 `CRSP_SUB_SCCAP(=3)` –Cap-Based Portfolios restrictions; same as 1, but also exclude first digit 3 and second digit 2,4,5,8, and 9 `CRSP_SUB_SCSIC(=4)` –CRSP Total Return Indexes; same but also include first digit of 9. `CRSP_SUB_SCFIL(=5)` –Restrictions specified by user. See the following parameters. `char *leftdig` – 10-digit character string made of 0's and 1's specifying which left digits of the Share Code are valid. If the n'th position in the string (starting from 0) is a 1, then a Share Code with a left digit of n is valid. leftdig is ignored unless sc_code is 5. `char *rightdig` – 10-digit character string made of 0's and 1's specifying which right digits of the Share Code are valid. If the `n'th` position in the string (starting from 0) is a 1, then a Share Code with a right digit of n is valid. rightdig is ignored unless `sc_code` is 5. for example, to allow only share codes of 10, 11, and 30, and 31, set `leftdig` to "010100000" and `rightdig` to "1100000000"
RETURN VALUES:	0 = if `shrcd` is valid according to the `scs` structure
	-1 = if `shrcd` is not valid according to the `scs` structure
SIDE EFFECTS:	none
PRECONDITIONS:	none

Translation Functions

These functions translate stock data in one or more time series to another. The different time series can be based on different calendars.

FUNCTION	DESCRIPTION	PAGE
`crsp_trans_comp_returns`	Compounds returns from one time series to another	page 187
`crsp_trans_last`	Translates Time Series Based On Last Value in Range	page 190
`crsp_trans_first`	Translates Time Series Based On First Value in Range	page 187
`crsp_trans_max`	Translates Time Series Based On Maximum Value in Range	page 198
`crsp_trans_min`	Translates Time Series Based On Minimum Value in Range	page 188
`crsp_trans_average`	Translates Time Series Based On Average Value in Range	page 189
`crsp_trans_median`	Translates Time Series Based On Median Value in Range	page 187

FUNCTION	DESCRIPTION	PAGE
crsp_trans_total	Translates Time Series Based On Total Value in Range	page 189
crsp_trans_last_closest	Translates Time Series Based On Closest Nonmissing Value in Range	page 200
crsp_trans_last_previous	Translates Time Series Based On Last Nonmissing Value in Range	page 195
crsp_trans_level	Loads a Target Time Series With Index Level Prices	page 192
crsp_trans_cumret	Loads a Target Time Series With Cumulative Returns	page 191
crsp_trans_port	Maps Portfolio Assignments To a New Time Series	page 191
crsp_trans_stat	Maps Portfolio Statistics To a New Time Series	page 195
crsp_trans_cap	Loads a Target Time Series With Capitalization Data	page 192
crsp_trans_gen_prc	General Translation Price Function	page 192

crsp_trans_comp_returns Compounds Returns From One Time Series to Another

PROTOTYPE:	int crsp_trans_comp_returns(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the compounded returns over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded and space allocated CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	src_ts and trg_ts arrtype is CRSP_FLOAT_NUM and subtype is CRSP_RETURN_NUM

crsp_trans_last Translates Time Series Based on Last Value in Range

PROTOTYPE:	int crsp_trans_last(CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the last price or volume over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	If the src_ts arrtype is CRSP_FLOAT_NUM thesubtype mustbe CRSP_PRICE_NUM or CRSP_PRICE_ADJ_NUM or CRSP_LEVEL_NUM. If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM or CRSP_VOLUME_ADJ_NUM or CRSP_COUNT_NUM. If the src_ts arrtype is CRSP_DOUBLE_NUM the subtype must be CRSP_WEIGHT_NUM or CRSP_CAP_NUM. The src_ts and trg_ts must have the same arrtype and subtype.

crsp_trans_first Translates Time Series Based on First Value in Range

PROTOTYPE:	int crsp_trans_first (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying th.e last price or volume over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	If the src_ts arrtype is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM The src_ts and trg_ts must have the same arrtype and subtype

crsp_trans_max Translates Time Series Based on Maximum Value in Range

PROTOTYPE:	int crsp_trans_max (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the maximum price or volume over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITONS:	If the src_ts arrtype is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM The src_ts and trg_ts must have the same arrtype and subtype

crsp_trans_min Translates Time Series Based on Minimum Value in Range

PROTOTYPE:	int crsp_trans_min (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the minimum price or volume over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITONS:	If the src_ts arrtype is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM The src_ts and trg_ts must have the same arrtype and subtype

`crsp_trans_average` Translates Time Series Based on Average Value in Range

PROTOTYPE:	`int crsp_trans_average (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)`
DESCRIPTION:	loads a target time series from a source time series by copying the average price or volume over each restricted period according to the calendar file.
ARGUMENTS:	`CRSP_TIMESERIES *src_ts` – source time series `CRSP_TIMESERIES *trg_ts` – target time series `int par_flag` – determines how missing values affect `beg` and `end` of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	If the `src_ts arrtype` is `CRSP_FLOAT_NUM`, the `subtype` must be `CRSP_PRICE_NUM` If the `src_ts arrtype` is `CRSP_INTEGER_NUM`, the `subtype` must be `CRSP_VOLUME_NUM` The `src_ts` and `trg_ts` must have the same `arrtype` and `subtype`

`crsp_trans_median` Translates Time Series Based on Median Value in Range

PROTOTYPE:	`int crsp_trans_median (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)`
DESCRIPTION:	loads a target time series from a source time series by copying the median price or volume over each restricted period according to the calendar file.
ARGUMENTS:	`CRSP_TIMESERIES *src_ts` – source time series `CRSP_TIMESERIES *trg_ts` – target time series `int par_flag` – determines how missing values affect `beg` and `end` of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process
PRECONDITIONS:	If the `src_ts arrtype` is `CRSP_FLOAT_NUM`, the `subtype` must be `CRSP_PRICE_NUM` If the `src_ts arrtype` is `CRSP_INTEGER_NUM`, the `subtype` must be `CRSP_VOLUME_NUM` The `src_ts` and `trg_ts` must have the same `arrtype` and `subtype`
SIDE EFFECTS:	Possible performance hit if large time series

`crsp_trans_total` Translates Time Series Based on Total Value in Range

PROTOTYPE:	`int crsp_trans_total (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)`
DESCRIPTION:	loads a target time series from a source time series, totalling data when converting to different calendars. If the source periods are shorter than the target periods, values in all source periods within a target period are summed before loading. If the source periods are longer than target periods, values of the source periods are averaged across all target periods, and the same value is loaded to all target periods in that range.
ARGUMENTS:	`CRSP_TIMESERIES *src_ts` – source time series `CRSP_TIMESERIES *trg_ts` – target time series `int par_flag` – determines how missing values affect `beg` and `end` of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	`CRSP_SUCCESS`: if successfully loaded `CRSP_FAIL`: if error in parameters or loading process

PRECONDITIONS:	The src_ts arrtype must be CRSP_FLOAT_NUM or CRSP_INTEGER_NUM If it is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If it is CRSP_INTEGER_NUM, the subtype must be one of CRSP_VOLUME_NUM, CRSP_VOLUME_ADJ_NUM, or CRSP_COUNT_NUM The target arrtype must be CRSP_INTEGER_NUM, CRSP_FLOAT_NUM, or CRSP_DOUBLE_NUM

crsp_trans_last_closest Translates Time Series Based on Closest Nonmissing Value

PROTOTYPE:	int crsp_trans_last_closest (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, CRSP_ARRAY *dists, int dlylim, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the closest non-missing to last price (price over each restricted period according to the calendar file). This adjusts the price if there are any distributions between it and month- end so that the return will be calculated properly. Passed a limit of days to use before giving up. If ties, preceding data gets precedence. Will not go outside of current or next period. Also sets the begin and end.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series CRSP_ARRAY *dists – CRSP_ARRAY of distributions int dlylim – limit of date periods to use before giving up int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
SIDE EFFECTS:	Target, price flag, and last date time series are loaded and their ranges are set
PRECONDITIONS:	If the src_ts arrtype is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM The src_ts and trg_ts must have the same arrtype and subtype

crsp_trans_last_previous Translates Time Series Based on Last Nonmissing Value in Range

PROTOTYPE:	int crsp_trans_last_previous (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, CRSP_TIMESERIES *prcflag_ts, CRSP_TIMESERIES *lastdt_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the previous non-missing to last price over each restricted period according to the calendar file. Also loads the prcflag and lastdt time series according to the case and the date of the non-missing value found. Will not go outside of current period.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series CRSP_TIMESERIES *prcflag_ts – price flag time series. Each period is set to -1 if no non-zero price if a period- end price is found, and 1 if an earlier price in the period is found. CRSP_TIMESERIES *lastdt_ts – last date time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	If the src_ts arrtype is CRSP_FLOAT_NUM, the subtype must be CRSP_PRICE_NUM If the src_ts arrtype is CRSP_INTEGER_NUM, the subtype must be CRSP_VOLUME_NUM The src_ts and trg_ts must have the same arrtype and subtype

crsp_trans_level Loads a Target Time Series with Index Levels

PROTOTYPE:	int crsp_trans_level (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int basedt, float baseamt)
DESCRIPTION:	loads a target time series with index level prices from a source time series with returns based on a base date and base amount for that date.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series of returns CRSP_TIMESERIES *trg_ts – target time series of prices int basedt – base date, YYYYMMDD date where levels are anchored. Level on this date is set to baseamt and other levels are set by successively compounding returns from the starting point. float baseamt – base amount, if = 0 the baseamt = source on basedt. Target time series will contain baseamt on basedt.
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
SIDE EFFECTS:	target time series is loaded with levels. Subtype of target is set to CRSP_LEVEL_NUM
PRECONDITIONS:	src can be same as trg. Normally target subtype must be CRSP_LEVEL_NUM but if src=trg must be CRSP_RETURN_NUM. src_ts and trg_ts must have the same calendar.

crsp_trans_cumret Loads a Target Time Series with Cumulative Returns

PROTOTYPE:	int crsp_trans_cumret (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int basedt)
DESCRIPTION:	loads a target time series with cumulative returns from a source time series with level prices based on a base date.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series of reruns CRSP_TIMESERIES *trg_ts – target time series of prices int basedt – base date
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	If the src_ts arrtype is CRSP_RETURN_NUM the subtype must be CRSP_RETURN_CUM_NUM If the src_ts arrtype is CRSP_LEVEL_NUM the subtype must be CRSP_RETURN_NUM

crsp_trans_port Maps Portfolio Assignments to a New Time Series

PROTOTYPE:	int crsp_trans_port (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the last portfolio number over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	The src_ts arrtype is CRSP_STK_PORT_NUM The trg_ts arrtype is CRSP_INTEGER_NUM and the subtype must be CRSP_PORT_PORT_NUM

crsp_trans_stat Maps Portfolio Statistics to a New Time Series

PROTOTYPE:	int crsp_trans_stat (CRSP_TIMESERIES *src_ts, CRSP_TIMESERIES *trg_ts, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by copying the last portfolio statistic number over each restricted period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *src_ts – source time series CRSP_TIMESERIES *trg_ts – target time series int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	The src_ts arrtype is CRSP_STK_PORT_NUM The trg_ts arrtype is CRSP_DOUBLE_NUM and the subtype must be CRSP_PORT_STAT_NUM

crsp_trans_cap Loads a Target Time Series with Capitalization Data

PROTOTYPE:	int crsp_trans_cap (CRSP_TIMESERIES *prc_ts, CRSP_TIMESERIES *shr_ts, CRSP_TIMESERIES *cap_ts, int flags)
DESCRIPTION:	loads a target time series with capitalization data from two source time series – one with prices and the other with shares – by multiplying the two values over each period according to the calendar file.
ARGUMENTS:	CRSP_TIMESERIES *prc_ts – input prices time series CRSP_TIMESERIES *shr_ts – input shares time series CRSP_TIMESERIES *cap_ts – output capitalization time series int flags – flags passed to the function: CRSP_ACTUAL means cap from the source is moved to the same period on target cap[i] = prc[i] * shr[i] CRSP_EFFECTIVE means cap from the source is moved to the next period on target cap[i+1] = prc[i] * SHR[I]
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process
PRECONDITIONS:	If the prc_ts subtype is CRSP_PRICE_ADJ_NUM the shr_ts subtype must be CRSP_SHARES_ADJ_NUM If the prc_ts subtype is CRSP_PRICE_NUM the shr_ts subtype must be CRSP_SHARES_IMP_NUM The cap_ts arrtype is CRSP_DOUBLE_NUM and the subtype is CRSP_CAP_NUM The prc_ts shr_ts and cap_ts must have the same calendar

crsp_trans_gen_prc General Translation Price Function

PROTOTYPE:	int crsp_trans_gen_prc (CRSP_TIMESERIES *srcprc_ts, CRSP_TIMESERIES *trgprc_ts, CRSP_STK_STRUCT *stkptr, int case_flag, int adj_flag, int par_flag)
DESCRIPTION:	loads a target time series from a source time series by linking between the two calendars and copying the price values to the target time series. Uses other translation functions to adjust, use last or last nonmissing price in range. General translation price function, used for prc, ask, bid, askhi, bidlo, adjprc, adjask, adjbid, adjaskhi, adjbidlo
ARGUMENTS:	CRSP_TIMESERIES *srcprc_ts – source price time series CRSP_TIMESERIES *trgprc_ts – target price time series CRSP_STK_STRUCT *stkptr – stock structure pointer int case_flag – last value or previous nonmissing price in range (0, 1) int adj_flag – adjust or not (1, 0) int par_flag – determines how missing values affect beg and end of target: 0 = allow missing values at beginning and ending of target range 1 = not allow missing values at beginning of target range 2 = not allow missing values at ending of target range 3 = not allow missing values at beginning and ending of target range
RETURN VALUES:	CRSP_SUCCESS: if successfully loaded CRSP_FAIL: if error in parameters or loading process

CHAPTER 6: LEGACY SET ACCESS IN FORTRAN

FORTRAN-95 DATA STRUCTURES

FORTRAN-95 Programming provides complete support for CRSP databases, including direct access on PERMNO, CUSIP and other header variables, and full support of all data items. INCLUDE files containing TYPE definitions, an object library to support linking, and sample programs illustrating access methods are available.

DATA ORGANIZATION FOR FORTRAN-95 PROGRAMMING

The basic levels of a CRSPAccess database are the database, set type, set id, module, object, and array. They are defined as follows:

- **Database** `(CRSPDB)` is the directory containing the database files. A `CRSPDB` is identified by its database path.
- **Set Type** is a predefined type of financial data. Each set type has its own defined set of data structures, specialized access functions, and keys. CRSPAccess databases support stock (STK) and index (IND) set types. A `CRSPDB` can include more than one set type.
- **Set Identifier** `(SETID)` is a defined subset of a set type. `SETID`s of the same set type use the same access functions, structures, and keys, but have different characteristics within those structures. For example, daily stock sets use the same data structure as monthly stock sets, but time series are associated with different calendars. Multiple `SETID`s of the same set type can be present in one `CRSPDB`.
- **Modules** are the groupings of data found in the data files in a `CRSPDB`. Multiple data items can be present in a module. Data are retrieved from storage on disk at the module level, and access functions retrieve data items for keys based on selected modules. Modules correspond to physical data files.
- **Objects** are the fundamental data types defined for each set type. There are three fundamental object types: time series `(CRSP_TIMESERIES)`, arrays `(CRSP_ARRAY)`, and headers `(CRSP_ROW)`. Objects contain header information such as counts, ranges, or associated calendars `(CRSP_CAL)` plus arrays of data for zero or more observations. Some set types support arrays of objects of a single type. In this case, the number of available objects is determined by the `SETID`, and each of the objects in the list has independent counts, ranges, or associated calendars.
- **Arrays** are attached to each object. Each array contains a set of observations and is the basic level of program- ming access. An observation can be a simple data type such as an integer from an array of volumes, or a complex structure such as one record from name history. When there is an array of objects, there is a corresponding array of arrays within the data.

DATA OBJECTS

There are four basic types of information stored in CRSP databases. Each is associated with a CRSP object structure.

- **Header Information.** These are identifiers with no implied time component. Header data contain the most current CRSPAccess information stored in the databases.
- **Event Arrays.** Arrays can represent status changes, sporadic events, or observations. The time of the event and relevant information is stored for each observation. There is a count of the number of observations for each type of event data.
- **Time Series Arrays.** An observation is available for each period in an associated calendar. Beginning and ending valid data are available for each type of time series data. Data are stored for each period in the range – missing values are stored as placeholders if information is not available for a period.
- **Calendar Arrays.** Each time series corresponds to an array of relevant dates. This calendar array is used in conjunction

with the time series arrays to attach dates to observations.

An observation can be a simple value or contain multiple components such as codes and amounts. Time series, except Portfolios, are based on calendars which share the frequency of the database. In a monthly database, the time series are based on a month-end trading date calendar. In a daily database, the time series are based on a daily trading date calendar that excludes market holidays. Portfolio calendars are dependent on the rebalancing methodology of the specific portfolio type. All calendars are attached automatically to each requested time series object when the database is opened.

There are four base CRSPAccess FORTRAN-95 structures called objects used in CRSPDBs. The following table contains each of the objects in bold upper-case, followed by the components, lower-case and indented, which each object type contains. All data items are defined in terms of the following objects:

OBJECT OR FIELD	USAGE	DATA TYPE
objtype	object type code identifies the structure as a CRSP_ARRAY, always = 3	INTEGER
arrtype	array type code defines the structure in the array. Base FORTRAN-95 types or CRSP-defined structures each have associated codes defined in the constants header file	INTEGER
subtype	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields.	INTEGER
maxarr	maximum number of array elements containing valid data	INTEGER
num	number of array elements containing valid data	INTEGER
dummy	data secondary subtype code	INTEGER
CRSP_ROW	Structure for storing header data	
objtype	object type code identifies the structure as a CRSP_ROW, always = 5	INTEGER
arrtype	array type code defines the structure in the array. Base FORTRAN-95 types or CRSP-defined structures each have associated codes defined in the constants header file	INTEGER
subtype	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields.	INTEGER
objtype	object type code identifies the structure as a CRSP_TIMESERIES, always = 2	INTEGER
arrtype	array type code defines the structure in the array. Base FORTRAN-95 types or CRSP-defined structures each have associated codes defined in the constants header file	INTEGER
subtype	data subtype code defines a subcategory of array data. Subtypes further differentiate arrays with common array type fields.	INTEGER
maxarr	maximum number of array elements	INTEGER
beg	first array index having valid data for the current record. (0 if no valid range.)	INTEGER
end	last array index having valid data for the current record. (0 if no valid range.)	INTEGER
caltype	calendar time period description code describes the type of time periods. Calendar Type (caltype) is always 2, indicating time periods are described in the Calendar Trading Date (caldt) array by the last trading date in the period.	INTEGER
cal	calendar associated with time series is a pointer to the calendar associated with the time series array. The calendar includes the matching period- ending dates for each array index.	CRSP_CAL, POINTER
objtype	object type code identifies the structure as a CRSP_CAL, always = 1	INTEGER
calid	calendar identification number is an identifier assigned to each specific calendar by CRSP	INTEGER
maxarr	maximum number of trading periods allocated for the calendar	INTEGER
loadflag	calendar type availability flag is a code indicating the types of calendar arrays loaded. Currently = 2 for calendar trading date (caldt) only	INTEGER
ndays	number of valid dates in calendar (index of last valid date in caldt)	INTEGER

OBJECT OR FIELD	USAGE	DATA TYPE
name	the calendar name in text	CHAR[80]
callist	calendar period grouping identifiers reserved for array of alternate grouping identifiers for calendar periods	*
caldt	calendar trading date is an array of calendar period ending dates, stored in CCYYMMDD format. Calendars start at element 1 and end at element number of days (ndays)	*
calmap	used to store array of first and last calendar period array elements in a calendar linked to elements in this calendar	CRSP_CAL_MAP *
basecal	used to point to a calendar linked in calmap	CRSP_CAL *

SET STRUCTURES AND USAGE

Stock and index access functions initialize and load data to FORTRAN-95 top-level defined set structures. Top- level structures are built from general object and array structure definitions and contain object and array pointers that have memory allocated to them by access open functions.

Two set types and six set identifiers are currently supported for stock and index data. The identifier must be specified when opening or accessing data from the set.

DATA	SET TYPE	SET IDENTIFIERS	FREQUENCY
CRSP Stock Data	STK	10 STK_DAILY	Daily
		20 STK_MONTHLY	Monthly
CRSP Indexes Data	IND	400 MONTHLY_INDEX_GROUPS	Monthly Groups (in CRSP index product only)
		420 MONTHLY_INDEX_SERIES	Monthly Series
		440 DAILY_INDEX_GROUPS	Daily Groups (in CRSP index product only)
		460 DAILY_INDEX_SERIES	Daily Series

Each set structure has three types of pointer definitions.

- Module pointers to CRSP_OBJECT_ELEMENT linked lists are needed internally to keep track of the objects in a module. These have the suffix _obj and can be ignored by ordinary programming.
- Object pointers define a CRSP_ARRAY, CRSP_ROW, or CRSP_TIMESERIES object type. A suffix, _arr, _ts, or _row is appended to the variable name. Valid range variables num, beg, and end are accessed from these variables.
- Array pointers define a data item array. The array has the same rank as the object but without the suffix. It is a pointer to the array element of the object and is used for general access to the data item.

If a module has multiple types of objects, a group structure is created with definitions for those objects and is included in the main structure.

If a module has a variable number of objects of one type, an integer variable keeps track of the actual number. These variables end with the suffix types and are based on the set type.

Each of the top-level structures contains three standard elements:

- PERMNO – the actual key loaded
- loadflag, a binary flag matching the set wanted parameters indicating which pointers have been allocated. See the open function for the set for more information about wanted parameters.
- setcode, a constant identifying the type of set (1=STK, 3=IND)

For example, the TYPE crsp_stk item has a CRSP_TIMESERIES object named prc_ts containing an array named prc.

FORTRAN-95 LANGUAGE DATA OBJECTS FOR CRSP STOCK DATA

Each `TYPE` (`crsp_stk`) item contains a fixed set of possible objects. These objects contain the header information required to use the CRSP data structures, as well as the data arrays. Data elements are described in the FORTRAN-95 Data Structure Table under the array name.

Time series `beg` and `end` are both 0 if there are no data. Otherwise `beg` > 0, `beg <= end`, and `end <= maxarr`.

The `TYPE` (`crsp_stk`) contains an array of portfolio time series. Each component contains the portfolio statistic and assignment data for one portfolio type. Each component can have an individual range and calendar. The number of Portfolio Types is found in the `port` types variable.

NAME	OBJECT	VALID DATA RANGE
Stock Header Structure	`header_row`	`stk % stkhdr`
Security Name History	`names_arr`	`stk % names_arr(i), i from 1 to stk % num_names`
Distribution History Array	`dists_arr`	`stk % dists_arr(i), i from 1 to stk % num_dists`
Shares Structure Array	`shares_arr`	`stk % shares_arr(i), i from 1 to stk % num_shares`
Delisting Structure Array	`delist_arr`	`stk % delist_arr(i), i from 1 to stk % num_delist`
NASDAQ Structure Array	`nasdin_arr`	`stk % nasdin_arr(i), i from 1 to stk % num_nasdin`
Portfolio Statistics and Assignments	`port_ts( )`	`stk % port_ts(i) % port(j), i from 1 to stk % porttypes, j from stk % port_beg to stk % port_end`
Array of Group Arrays	`group_arr( )`	`stk % group_arr(i) % group(j), i from 1 to grouptypes, j from 1 to stk % num_groups`
Closing Price or Bid/Ask Average	`prc_ts`	`stk % prc(i), from stk % prc_beg to stk % prc_end`
Holding Period Total Return	`ret_ts`	`stk % ret(i), from stk % ret_beg to stk % ret_end`
Bid or Low	`bidlo_ts`	`stk % bidlo(i), from stk % bidlo_beg to stk % bidlo_end`
Ask or High	`askhi_ts`	`stk % askhi(i), from stk % askhi_beg to stk % askhi_end`
NASDAQ Closing Bid	`bid_ts`	`stk % bid(i), from stk % bid_beg to stk % bid_end`
NASDAQ Closing Ask	`ask_ts`	`stk % ask(i), from stk % ask_beg to stk % ask_end`
Return Without Dividends	`retx_ts`	`stk % retx(i), from stk % retx_beg to stk % retx_end`
Alternate Price	`altprc_ts`	`stk % altprcdt(i), from stk % altprcdt_beg to stk % altprcdt_end`
Open Price	`openprc_ts`	`stk % openprc(i), from stk % openprc_beg to stk % openprc_end`
Month End Bid/Ask Spread	`spread_ts`	`stk % spread(i), from stk % spread_beg to stk % spread_end`
Exchange Price	`exchprc_ts`	`stk % exchprc(i), from stk % exchprc_beg to stk % exchprc_end`
Volume Traded	`vol_ts`	`stk % vol(i), from stk % vol_beg to stk % vol_end`
NASDAQ Number of Trades or Alternate Price Date	`numtrd_ts`	`stk % numtrd(i), from stk % numtrd_beg to stk % numtrd_end`
Alternate Price Date	`altprcdt_ts`	`stk % altprc(i), from stk % altprc_beg to stk % altprc_end`

FORTRAN-95 LANGUAGE DATA STRUCTURE FOR CRSP STOCK DATA

All CRSP-defined data type structures have names in all capitals beginning with `CRSP_` and are immediately followed by the definitions in the next level of indentation

Index and Date Ranges for all elements in a structure are the same as for the structure itself. There are three structure levels indicated by the indentation in the mnemonic field. Pointers at any level can be used in a program. The top level contains all other items and is used in all access functions. The second level indicates data grouped in modules. See the CRSPAccess Stock Users Guide for data item definitions.

All character strings, indicated by `character(#)`, are `NULL` terminated. The number of characters – 1 is the maximum string length allowed. Actual maxi- mums may be lower. The top level `stk` structure is an example used by CRSP Stock sample programs. Other names can be used, and multiple `CRSP_STK_STRUCT`s can be declared in a program. See the `CRSP_STK` open access function for initializing a stock structure.

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
stk_data	Master Stock Structure	**stk_data**	**stk**				
Header Data	Stock Header Structure						
hcusip	CUSIP - Header	char[16]		stk % stkhdr % hcusip			
permno	PERMNO	int		stk % stkhdr % permno			
permco	PERMCO	int		stk % stkhdr % permco			
issuno	NASDAQ Issue Number	int		stk % stkhdr % issuno			
compno	NASDAQ Company Number	int		stk % stkhdr % compno			
hexcd	Exchange Code - Header	int		stk % stkhdr % hexcd			
hsiccd	Standard Industrial Classification (SIC) Code - Header	int		stk % stkhdr % hsiccd			
hshrcd	Share Code - Header	int		stk % stkhdr % hshrcd			
hnamecd	Name Code - Header	int		stk % stkhdr % hnamecd			
begdt	Begin of Stock Data	int		stk % stkhdr % begdt			
enddt	End of Stock Data	int		stk % stkhdr % enddt			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
dlstcd	Delisting Code - Header	int		stk % stkhdr % dlstcd			
htick	Ticker Symbol - Header	char[16]		stk % stkhdr % htick			
hnaics	North American Industry Classification System (NAICS) - Header	char[8]		stk % stkhdr % hnaics			
hcomnam	Company Name - Header	char[36]		stk % stkhdr % hcomnam			
htsymbol	Trading Ticker Symbol - Header	char[12]		stk % stkhdr % htsymbol			
hcntrycd	Country Code - Header	char[4]		stk % stkhdr % hcntrycd			
primexch	Primary Exchange - Header	char[1]		stk % stkhdr % hprimexch			
hsubexch	Sub-Exchange - Header	char[1]		stk % stkhdr % hsubexch			
trdstat	Trading Status - Header	char[1]		stk % stkhdr % htrdstat			
hsecstat	Security Status - Header	char[1]		stk % stkhdr % hsecstat			
hshrtype	Share Type - Header	char[1]		stk % stkhdr % shrtype			
hissuercd	Issuer Code - Header	char[1]		stk % stkhdr % hissuercd			
hinccd	Incorporation Code - Header	char[1]		stk % stkhdr % hinccd			
hits	Intermarket Trading System Indicator - Header	char[1]		stk % stkhdr % hits			
hdenom	Trading Denomination - Header	char[1]		stk % stkhdr % hdenom			
heligcd	Eligibility Code - Header	char[1]		stk % stkhdr % heligcd			
hconvcd	Convertible Code - Header	char[1]		stk % stkhdr % hconvcd			
hnameflag	Name Flag - Header	char[1]		stk % stkhdr % hnameflag			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
hrating	Interest Rate or Strike Price - Header	real *		stk % stkhdr % hrating			
Name History Data	Security Name History				**i** between **1** and **stk % num_names**	**i** between **1** and **stk % num_names**	name effective from **stk % names(i) % namedt** to **stk % names(i) % nameenddt**
namedt	Name Effective Date	int	stk % names(i) % namedt	stk % names_arr % names(i) % namedt			
nameenddt	Last Date of Name	int	stk % names(i) % nameenddt	stk % names_arr % names(i) % nameenddt			
ncusip	CUSIP	char[16]	stk % names(i) % ncusip	stk % names_arr % names(i) % ncusip			
ticker	Ticker Symbol	char[8]	stk % names(i) % ticker	stk % names_arr % names(i) % ticker			
comnam	Company Name	char[36]	stk % names(i) % comnam	stk % names_arr % names(i) % comnam			
shrcls	Share Class	char[4]	stk % names(i) % shrcls	stk % names_arr % names(i) % shrcls			
shrcd	Share Code	int	stk % names(i) % shrcd	stk % names_arr % names(i) % shrcd			
exchcd	Exchange Code	int	stk % names(i) % exchcd	stk % names_arr % names(i) % exchcd			
siccd	Standard Industrial Classification (SIC) Code	int	stk % names(i) % siccd	stk % names_arr % names(i) % siccd			
naics	North American Industry Classification System (NAICS) Code	char(8)	stk % names(i) % naics	stk % names_arr % names(i) % naics			
tsymbol	Trading Ticker Symbol	char[12]	stk % names(i) % tsymbol	stk % names_arr % names(i) % tsymbol			
cntrycd	Country Code	char[4]	stk % names(i) % cntrycd	stk % names_arr % names(i) % cntrycd			
primexch	Primary Exchange	char[1]	stk % names(i) % primexch	stk % names_arr % names(i) % primexch			
subexch	Sub-Exchange	char[1]	stk % names(i) % subexch	stk % names_arr % names(i) % subexch			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
trdstat	Trading Status	char[1]	stk % names(i) % trdstat	stk % names_arr % names(i) % trdstat			
secstat	Security Status	char[1]	stk % names(i) % secstat	stk % names_arr % names(i) % secstat			
shrtype	Share Type	char[1]	stk % names(i) % shrtype	stk % names_arr % names(i) % shrtype			
issuercd	Issuer Code	char[1]	stk % names(i) % issuercd	stk % names_arr % names(i) % issuercd			
inccd	Incorporation Code	char[1]	stk % names(i) % inccd	stk % names_arr % names(i) % inccd			
its	Intermarket Trading System Indicator	char[1]	stk % names(i) % its	stk % names_arr % names(i) % its			
denom	Trading Denomination	char[1]	stk % names(i) % denom	stk % names_arr % names(i) % denom			
eligcd	Eligibility Code	char[1]	stk % names(i) % eligcd	stk % names_arr % names(i) % eligcd			
convcd	Convertible Code	char[1]	stk % names(i) % convcd	stk % names_arr % names(i) % convcd			
nameflag	Name Flag	char[1]	stk % names(i) % nameflag	stk % names_arr % names(i) % nameflag			
dists	Distribution History Array				**i** between **1** and **stk % num_dists**	**i** between **1** and **stk % num_dists**	distribution effective on **stk % dists(i) % exdt**
distcd	Distribution Code	int	stk % dists(i) % distcd	stk % dists_arr % dists(i) % distcd			
divamt	Dividend Cash Amount	real	stk % dists(i) % divamt	stk % dists_arr % dists(i) % divamt			
facpr	Factor to Adjust Price	real	stk % dists(i) % facpr	stk % dists_arr % dists(i) % facpr			
facshr	Factor to Adjust Shares Outstanding	real	stk % dists(i) % facshr	stk % dists_arr % dists(i) % facshr			
dclrdt	Distribution Declaration Date	int	stk % dists(i) % dclrdt	stk % dists_arr % dists(i) % dclrdt			
exdt	Ex-Distribution Date	int	stk % dists(i) % exdt	stk % dists_arr % dists(i) % exdt			
rcrddt	Record Date	int	stk % dists(i) % rcrddt	stk % dists_arr % dists(i) % rcrddt			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
paydt	Payment Date	int	stk % dists(i) % paydt	stk % dists_arr % dists(i) % paydt			
acperm	Acquiring PERMNO	int	stk % dists*(i) % acperm	stk % dists_arr % dists*(i) % acperm			
accomp	Acquiring PERMCO	int	stk % dists(i) % accomp	stk % dists_arr % dists(i) % accomp			
shares	Shares Structure Array				**i** between **1** and **stk % num_shares**	**i** between **1** and **stk % num_shares**	shares observation effective from **stk % shares(i) % shrsdt to stk % shares[i] % shrsenddt**
shrout	Shares Outstanding	int	stk % shares(i) % shrout	stk % shares_arr % shares(i) % shrout			
shrsdt	Shares Outstanding Observation Date	int	stk % shares(i) % shrsdt	stk % shares_arr % shares(i) % shrsdt			
shrsenddt	Shares Outstanding Observation End Date	int	stk % shares(i) % shrsenddt	stk % shares_arr % shares(i) % shrsenddt			
shrflg	Shares Outstanding Observation Flag	int	stk % shares(i) % shrflg	stk % shares_arr % shares(i) % shrflg			
delist	Delisting Structure Array				**i** between **1** and **stk % num_delist**	**i** between **1** and **stk % num_delist**	delist observation on **stk % delist(i) % dlstdt**
dlstdt	Delisting Date	int	stk % delist(i) % dlstdt	stk % delist_arr % delist(i) % dlstdt			
dlstcd	Delisting Code	int	stk % delist(i) % dlstcd	stk % delist_arr % delist(i) % dlstcd			
nwperm	New PERMNO	int	stk % delist(i) % nwperm	stk % delist_arr % delist(i) % nwperm			
nwcomp	New PERMCO	int	stk % delist(i) % nwcomp	stk % delist_arr % delist(i) % nwcomp			
nextdt	Delisting Date of Next Available Information	int	stk % delist(i) % nextdt	stk % delist_arr % delist(i) % nextdt			
dlamt	Amount After Delisting	real	stk % delist(i) % dlamt	stk % delist_arr % delist(i) % dlamt			
dlretx	Delisting Return without Dividends	real	stk % delist(i) % dlretx	stk % delist_arr % delist(i) % dlretx			
dlprc	Delisting Price	real	stk % delist(i) % dlprc	stk % delist_arr % delist(i) % dlprc			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
dlpdt	Delisting Payment Date	int	stk % delist(i) % dlpdt	stk % delist_arr % delist(i) % dlpdt			
dlret	Delisting Return	real	stk % delist(i) % dlret	stk % delist_arr % delist(i) % dlret			
nasdin	NASDAQ Structure Array				**i** between **1** and **stk % num_nasdin**	**i** between **1** and **stk % num_nasdin**	Nasdaq status effective from **stk % nasdin(i) % trtsdt** to **stk % nasdin[i] % trtsenddt**
trtscd	NASDAQ Traits Code	int	stk % nasdin(i) % trtscd	stk % nasdin_arr % nasdin(i) % trtscd			
trtsdt	NASDAQ Traits Date	int	stk % nasdin(i) % trtsdt	stk % nasdin_arr % nasdin(i) % trtsdt			
trtsenddt	NASDAQ Traits End Date	int	stk % nasdin(i) % trtsenddt	stk % nasdin_arr % nasdin(i) % trtsenddt			
nmsind	NASDAQ National Market Indicator	int	stk % nasdin(i) % nmsind	stk % nasdin_arr % nasdin(i) % nmsind			
mmcnt	Market Maker Count	int	stk % nasdin(i) % mmcnt	stk % nasdin_arr % nasdin(i) % mmcnt			
nsdinx	NASD Index Code	int	stk % nasdin(i) % nsdinx	stk % nasdin_arr % nasdin(i) % nsdinx			
port	Portfolio Statistics and Assignments				**j between 1 and stk % porttypes, i between stk % port_ts(j) % beg and stk % port_ts(j) % end**	**j between 1 and stk % porttypes, i between stk % port_ts(j) % beg and stk % port_ts(j) % end**	value for period ending **stk % port_ts(j) % cal % caldt(i)**
port	Portfolio Assignment Number	int	stk % port(j,i) % port	stk % port_ts(j) % port(i) % port			
stat	Portfolio Statistic Value	double precision	stk % port(j,i) % stat	stk % port_ts(j) % port(i) % stat			
group	Group Array					**j** between **1** and stk % grouptypes, **i** between **1** and **stk % group_arr(j) % group_parms % num**	value for period ending **stk % group_arr(j) % group(i) % grpenddt**
grpdt	Begin of Group Data	int		stk % group_arr(j) % group(i) % grpdt			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
grpenddt	End of Group Data	int		stk % group_arr(j) % group(i) % grpenddt			
grpflag	Group Flag of Associated Index	int		stk % group_arr(j) % group(i) % grpflag			
grpsubflag	Group Secondary Flag	int		stk % group_arr(j) % group(i) % grpsubflag			
Time Series Data Arrays							
prc	Price or Bid/Ask Average	real *	stk % prc(i)	stk % prc_ts % prc(i)	i between stk % prc_beg and stk % prc_end	i between stk % prc_beg and stk % prc_end	value on date stk % prc_ts % prc_parms % cal % caldt(i)
ret	Holding Period Total Return	real *	stk % ret(i)	stk % ret_ts % ret(i)	i between stk % ret_beg and stk % ret_end	i between stk % ret_beg and stk % ret_end	value on date stk % ret_ts % ret_parms % cal % caldt(i)
bidlo	Bid or Low Price	real *	stk % bidlo(i)	stk % bidlo_ts % bidlo(i)	i between stk % bidlo_beg and stk % bidlo_end	i between stk % bidlo_beg and stk % bidlo_end	value on date stk % bidlo_ts % bidlo_ parms % cal % caldt(i)
askhi	Ask or High Price	real *	stk % askhi(i)	stk % askhi_ts % askhi(i)	i between stk % askhi_beg and stk % askhi_end	i between stk % askhi_beg and stk % askhi_end	value on date stk % askhi_ts % askhi_ parms % cal % caldt(i)
bid	Bid	real *	stk % bid(i)	stk % bid_ts % bid(i)	i between stk % bid_beg and stk % bid_end	i between stk % bid_beg and stk % bid_end	value on date stk % bid_ts % bid_parms % cal % caldt(i)
ask	Ask	real *	stk % ask(i)	stk % ask_ts % ask(i)	i between stk % ask_beg and stk % ask_end	i between stk % ask_beg and stk % ask_end	value on date stk % ask_ts % ask_parms % cal % caldt(i)
retx	Return Without Dividends	real *	stk % retx(i)	stk % retx_ts % retx(i)	i between stk % retx_beg and stk % retx_end	i between stk % retx_beg and stk % retx_end	value on date stk % retx_ts % retx_ parms % cal % caldt(i)

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
openprc	Open Price (daily only)	real *	stk % openprc(i)	stk % openprc_ts % openprc(i)	i between stk % altprc_beg and stk % altprc_end	i between stk % openprc_beg and stk % openprc_end	value on date stk % openprc_ts % openprc_parms % cal % caldt(i)
altprc	Price Alternate (monthly only)	real *	stk % altprc(i)	stk % altprc_ts % altprc(i)	i between stk % altprc_beg and stk % altprc_end	i between stk % altprc_beg and stk % altprc_end	value on date stk % altprc_ts % altprc_parms % cal % caldt(i)
spread	Spread Between Bid and Ask	real *	stk % spread(i)	stk % spread_ts % spread(i)	i between stk % spread_beg and stk % spread_end	i between stk % spread_beg and stk % spread_end	value on date stk % spread_ts % spread_parms % cal % caldt(i)
vol	Volume Traded	int *	stk % vol(i)	stk % vol_ts % vol(i)	i between stk % vol_beg and stk % vol_end	i between stk % vol_beg and stk % vol_end	value on date stk % vol_ts % vol_parms % cal % caldt(i)
numtrd	NASDAQ Number of Trades (daily only)	int *	stk % numtrd(i)	stk % numtrd_ts % numtrd(i)	i between stk % numtrd_beg and stk % numtrd_end	i between stk % numtrd_beg and stk % numtrd_end	value on date stk % numtrd_ts % numtrd_parms % cal % caldt(i)
altprcdt	Alternate Price Date (monthly only)	int *	stk % altprcdt(i)	stk % altprcdt_ts % altprcdt(i)	i between % altprcdt_end	i between stk % altprcdt_beg and stk % altprcdt_end	value on date stk % altprcdt_ts % altprcdt_parms % cal % caldt(i)
caldt	Calendar Trade Date	int *	stk % caldt(i)	stk % caldt(i)	i between 1 and stk % ndays	i between 1 and stk % ndays	n/a

EXAMPLES OF FORTRAN-95 VARIABLE USAGE FOR CRSP STOCK DATA

These assume a FORTRAN-95 variable stk of TYPE (`crsp_stk`)

CRSP Row/Header Data

Object Variable: `(sub-TYPE) stk_header`
Data Structure: `stk % stkhdr`
Sample `WRITE` Statement:

```
WRITE (*, 1) stk % stkhdr % permno, &
& stk % stkhdr % begdt, stk % stkhdr % enddt
1 FORMAT (1X, I5, 1X, I8, 1X, I8)
```

CRSP Array/Distributions

Object Variable: `(sub-TYPE) stk_dist`
Data Structure: `stk % stk_dists_arr % dists`
Sample `WRITE` Statement:

```
DO i = 1, stk % dists_arr % dists_parms % num
WRITE (*,1) stk % dists_arr % dists(i) % distcd, &
& stk % dists_arr % dists(i) % exdt
1 FORMAT (1X, I4, 1X, I8)
END DO
```

CRSP Time Series/Prices

Object Variable: `(sub-TYPE) stk_prc_ts`
Data Structure: `stk % stkprc_ts`
Sample `WRITE` statement:

```
DO i = stk % stkprc_ts % prc_ts % beg, &
& stk % stkprc_ts % prc_ts % end
WRITE (*, 1) stk % stkprc_ts % prc(i), &
& stk % stkprc_ts % prc_ts % cal % caldt(i)
1 FORMAT (1X, F11.5, 1X, I8)
END DO
```

CRSP Array of Time Series/Portfolios

Object Variable: `(sub-TYPE) stk_port`
Data Array: `stk % stk_port_ts(j)`
There are SIZE (`stk % stkport_ts`) portfolios available; `j` above ranges from `1` to `SIZE` (`stk % stkport_ts`)
Sample `WRITE` statement: This statement prints the date and the assignment for each year in the issue's range for `stk % stkport_ts(1)`, the NYSE / NYSEMKT / NASDAQ capitalization deciles.

```
DO i = stk % stkport_ts(1) % port_ts % beg, stk % stkport_ts(1) %
port_ts % end
WRITE (*,1) stk % stkport_ts(1) % port_ts %    &
& cal % caldt(i),    &
& stk % stkport_ts(1) % port(i) % port
1 FORMAT (1X, I8, 1X, I2)
```

CRSP Array of Group Arrays

Object Variable: `(sub-TYPE) stk_group_arr`
Data Array: `stk % stkgroup_arr(j) % group(i)`
There are `SIZE` (`stk % stkgroup_arr`) groups available; `j` above is between `1` and `SIZE` (`stk % stkgroup_arr`).
Sample `WRITE` statement:

This statement is executed only if the security has ever been included in the S&P 500 uni- verse (group type 16).

```
j = 16
IF (stk % stkgroup_arr(16) % croup_arr % num > 0) THEN
DO i = 1, stk % stkgroup_arr(16) % group_arr % num
       WRITE (*, 1) stk % stkgroup_arr(j) % &
              & group(i) % grpdt, stk % stkgroup_arr(j) % &
              & group(i) % grpeenddt, stk % stkgroup_arr(j) % &
              & group(i) % grpflag, stk % stkgroup_arr(j) % &
              & group(i) % grpsubflag
              1 FORMAT (1X, I8, 1X, I8, 1X, I2, 1X, I2)
       END DO
       END IF
```

FORTRAN-95 LANGUAGE DATA OBJECTS FOR CRSP INDEXES DATA

CRSP assigns a Permanent Index Identification Number (INDNO) to access the index data in FORTRAN-95 for individual series or portfolio groups. In the CRSP US Stock Database, a subset of market series is available. Additional series and groups are available when you subscribe to the CRSP US Historical Indexes Database and Security Portfolio Assignment Module. The index structure supports data for one series or group and includes header, rebalancing, and result information for one or more portfolios comprising the index.

Each index structure contains a fixed set of possible objects. Objects contain the header information needed to use the CRSP data structures as well as the data arrays. Data elements are described in the FORTRAN-95 Data Structure Table under the array name.

Time series `beg` and `end` are both equal to 0 if there are no data. Otherwise `beg` > 0, `beg <= end`, and `end < maxarr`. The 0th element of a time series array is reserved for the missing value for that data type.

Multiple series in the index structure refers to portfolio subgroups. Each of these will have the same `beg`, `end`, and calendar. In a `SERIES SETID`, the multiple series has a count of `1`. In a `GROUP SETID`, the count of series is found in the corresponding `xxxtypes` variable.

NAME	OBJECT	OBJECT ARRAY NAME
Indexes Header Object	`indhdr_row`	`ind % indhdr`
Rebalancing Arrays	`rebal_arr( )`	`ind % rebal(j), j` from `1` to `ind % rebaltypes`
List Arrays	`list_arr( )`	`ind % list(j), j` from `1` to `ind % listtypes`
Total Value Time Series	`totval_ts( )`	`ind % totval(j), j` from `1` to `ind % indtypes`
Total Count Time Series	`totcnt_ts( )`	`ind % totcnt(j), j` from `1` to `ind % indtypes`
Used Value Time Series	`usdval_ts( )`	`ind % usdval(j), j` from `1` to `ind % indtypes`
Used Count Time Series	`usdcnt_ts( )`	`ind % usdcnt(j), j` from `1` to `ind % indtypes`
Total Return Time Series	`tret_ts( )`	`ind % tret(j), j` from `1` to `ind % indtypes`
Capital Appreciation Time Series	`aret_ts( )`	`ind % aret(j), j` from `1` to `ind % indtypes`
Income Return Time Series	`iret_ts( )`	`ind % iret(j), j` from `1` to `ind % indtypes`
Total Return Index Level Time Series	`tind_ts( )`	`ind % tind(j), j` from `1` to `ind % indtypes`
Capital Appreciation Index Level Time Series	`aind_ts( )`	`ind % aind(j), j` from `1` to `ind % indtypes`
Income Return Index Level Time Series	`iind_ts( )`	`ind % iind(j), j` from `1` to `ind % indtypes`

FORTRAN-95 LANGUAGE DATA STRUCTURE FOR CRSP INDEXES DATA

All CRSP-defined data types have names in all capitals beginning with `CRSP_` and are immediately followed by the definitions in the next indented level.

Index and date ranges for all elements in a structure are the same as for the structure itself. There are four structure levels indicated by the indentation in the Mnemonic field. Pointers at any level can be used in a program. The top level contains all other items and is used in all access functions. The second level indicates data grouped in modules. See the Data Description Guide for data item definitions.

All character strings, indicated by `char[#]`, are null terminated. The number of characters - 1 is the maximum string length allowed. Actual maximums may be lower. The top level `ind` structure is an example used by CRSP Indexes sample programs. Other names can be used, and multiple `CRSP_IND_STRUCTs` may be declared in a program.

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
ind_data	Master Indexes Structure	**CRSP_IND**	**ind**				
indhdr	Indexes Header Object						
indno	INDNO	int		ind % indhdr % indno			
indco	INDCO	int		ind % indhdr % indco			
primflag	Index Primary Link	int		ind % indhdr % primflag			
portnum	Portfolio Number if Subset Series	int		ind % indhdr % portnum			
indname	Index Name	char[80]		ind % indhdr % indname			
groupname	Index Group Name	char[80]		ind % indhdr % groupname			
method	Index Methodology Description Structure	CRSP_IND_METHOD		ind % indhdr % method			
methcode	Index Method Type Code	int		ind % indhdr % method % methcode			
primtype	Index Primary Methodology Type	int		ind % indhdr % method % primtype			
subtype	Index Secondary Methodology Group	int		ind % indhdr % method % subtype			
wgttype	Index Reweighting Type Flag	int		ind % indhdr % method % wgttype			
wgtflag	Index Reweighting Timing Flag	int		ind % indhdr % method % wgtflag			
flags	Index Exception Handling Flags	CRSP_IND_FLAGS		ind % indhdr % flags			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
flagcode	Index Basic Exception Types Code	int		ind % indhdr % flags % flagcode			
addflag	Index New Issues Flag	int		ind % indhdr % flags % addflag			
delflag	Index Ineligible Issues Flag	int		ind % indhdr % flags % delflag			
delretflag	Return of Delisted Issues Flag	int		ind % indhdr % flags % delretflag			
missflag	Index Missing Data Flag	int		ind % indhdr % flags % missflag			
partuniv	Index Subset Screening Structure	CRSP_UNIV_PARAM		ind % indhdr % partuniv			
partunivcode	Universe Subset Types Code in a Partition Restriction	int		ind % indhdr % partuniv % univcode			
begdt	Partition Restriction Beginning Date	int		ind % indhdr % partuniv % begdt			
enddt	Partition Restriction End Date	int		ind % indhdr % partuniv % enddt			
wantexch	Valid Exchange Codes in the Universe in a Partition Restriction	int		ind % indhdr % partuniv % wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in a Partition Restriction	int		ind % indhdr % partuniv % wantnms			
wantwi	Valid When-Issued Securities in the Universe in a Partition Restriction	int		ind % indhdr % partuniv % wantwi			
wantinc	Valid Incorporation of Securities in the Universe in a Partition Restriction	int		ind % indhdr % partuniv % wantinc			
shrcd	Share Code Screen Structure in a Partition Restriction	CRSP_UNIV_SHRCD		ind % indhdr % partuniv % shrcd			
sccode	Share Code Groupings for Subsets in a Partition Restriction	int		ind % indhdr % partuniv % shrcd % sccode			
fstdig	Valid First Digit of Share Code in a Partition Restriction	int		ind % indhdr % partuniv % shrcd % fstdig			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
secdig	Valid Second Digit of Share Code in a Partition Restriction	int		ind % indhdr % partuniv % shrcd % secdig			
induniv	Partition Subset Screening Structure	CRSP_UNIV_PARAM		ind % indhdr % induniv			
indunivcode	Universe Subset Types Code in an Index Restriction	int		ind % indhdr % induniv % univcode			
begdt	Restriction Begin Date	int		ind % indhdr % induniv % begdt			
enddt	Restriction End Date	int		ind % indhdr % induniv % enddt			
wantexch	Valid Exchange Codes in the Universe in an Index Restriction	int		ind % indhdr % induniv % wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in an Index Restriction	int		ind % indhdr % induniv % wantnms			
wantwi	Valid When-Issued Securities in the Universe in an Index Restriction	int		ind % indhdr % induniv % wantwi			
wantinc	Valid Incorporation of Securities in the Universe in an Index Restriction	int		ind % indhdr % induniv % wantinc			
shrcd	Share Code Screen Structure in an Index Restriction	CRSP_UNIV_SHRCD		ind % indhdr % induniv % shrcd			
sccode	Share Code Groupings for Subsets in an Index Restriction	int		ind % indhdr % induniv % shrcd % sccode			
fstdig	Valid First Digit of Share Code in an Index Restriction	int		ind % indhdr % induniv % shrcd % fstdig			
secdig	Valid Second Digit of Share Code in an Index Restriction	int		ind % indhdr % induniv % shrcd % secdig			
rules	Portfolio Building Rules Structure	CRSP_IND_RULES		ind % indhdr % rules			
rulecode	Index Basic Rule Types Code	int		ind % indhdr % rules % rulecode			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
buyfnct	Index Function Code for Buy Rules	int		ind % indhdr % rules % buyfnct			
sellfnct	Index Function Code for Sell Rules	int		ind % indhdr % rules % sellfnct			
statfnct	Index Function Code for Generating Statistics	int		ind % indhdr % rules % statfnct			
groupflag	Index Statistic Grouping Code	int		ind % indhdr % rules % groupflag			
assign	Related Assignment Information	CRSP_IND_ASSIGN		ind % indhdr % assign			
assigncode	Index Basic Assignment Types Code	int		ind % indhdr % assign % assigncode			
asperm	INDNO of Associated Index	int		ind % indhdr % assign % asperm			
asport	Portfolio Number in Associated Index	int		ind % indhdr % assign % asport			
rebalcal	Calendar Identification Number of Rebalancing Calendar	int		ind % indhdr % assign % rebal_cal			
assigncal	Calendar Identification Number of Assignment Calendar	int		ind % indhdr % assign % assigncal			
calccal	Calendar Identification Number of Calculations Calendar	int		ind % indhdr % assign % calccal			
rebal	Array of Rebalancing Arrays					**j** between 1 and **ind % rebaltypes**, **i** between 1 and **ind % ind_rebal_arr (j) % num**	data valid from **ind % rebal (j,i) % rbbegdt** to **ind % rebal (j,i) % rbenddt**
rbbegdt	Index Rebalancing Begin Date	int		ind % rebal % rebal(j,i) % rbbegdt			
rbenddt	Index Rebalancing End Date	int		ind % rebal % rebal(j,i) % rbenddt			
usdcnt	Count Used as of Rebalancing	int		ind % rebal% rebal(j,i) % usdcnt			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
maxcnt	Maximum Count During Period	int		ind % rebal % rebal(j,i) % maxcnt			
totcnt	Count Available as of Rebalancing	int		ind % rebal % rebal(j,i) % totcnt			
endcnt	Count at End of Rebalancing Period	int		ind % rebal % rebal(j,i) % endcnt			
minid	Statistic Minimum Identifier	int		ind % rebal % rebal(j,i) % minid			
maxid	Statistic Maximum Identifier	int		ind % rebal % rebal(j,i) % maxid			
minstat	Statistic Minimum in Period	double precision		ind % rebal % rebal(j,i) % minstat			
maxstat	Statistic Maximum in Period	double precision		ind % rebal % rebal(j,i) % maxstat			
medstat	Statistic Median in Period	double precision		ind % rebal % rebal(j,i) % medstat			
avgstat	Statistic Average in Period	double precision		ind % rebal % rebal(j,i) % avgstat			
list	List Indexes Arrays				**j** between **1** and **ind % listtypes**, **i** between **1** and **ind % ind_list_arr(j) % num**	**j** between **1** and **ind % listtypes**, **i** between **1** and **ind % ind_list_arr(j) % num**	valid from **ind % list(j,i) % beg** to **ind % list(j,i) % enddt**
list	List Arrays	int	ind % list(j,i) % permno	ind % ind_list_arr % list(j,i) % permno			
permno	Permanent Number of Securities in Index List	int	ind % list(j,i) % permno	ind % ind_list_arr % list(j,i) % permno			

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
begdt	First Date Included in List	int	ind % list(j,i) % begdt	ind % ind_list_arr % list(j,i) % begdt			
enddt	Last Date Included in a List	int	ind % list(j,i) % enddt	ind % ind_list_arr % list(j,i) % enddt			
subind	Index Subcategory Code	int	ind % list(j,i) % subind	ind % ind_list_arr % list(j,i) % subind			
weight	Weight of an Issue	double precision	ind % list(j,i) % weight	ind % ind_list_arr % list(j,i) % weight			
Time Series Data Arrays							
aind	Index Capital Appreciation Index Level	real *	ind % aind(j,i)	ind % indaind_ts % aind(j,i)	j between 1 and indtypes, i between ind % aind_ts(j) % beg and ind_data % aind_ts(j) % end	j between 1 and indtypes, i between ind % aind_ts(j) % beg and ind % aind_ts(j) % end	value on date ind % aind_ts(j) % cal % caldt(i)
aret	Index Capital Appreciation Return	real *	ind % aret(j,i)	ind % indaret_ts % aret(j,i)	j between 1 and indtypes, i between ind % aret_ts(j) % beg and ind_data % aret_ts(j) % end	j between 1 and indtypes, i between ind % aret_ts(j) % beg and ind % aret_ts(j) % end	value on date ind % aret_ts(j) % cal % caldt(i)
iind	Index Income Return Index Level	real *	ind % iind(j,i)	ind % indiind_ts % iind(j,i)	j between 1 and indtypes, i between ind % iind_ts(j) % beg and ind_data % iind_ts(j) % end	j between 1 and indtypes, i between ind % iind_ts(j) % beg and ind % iind_ts(j) % end	value on date ind % iind_ts(j) % cal % caldt(i)
iret	Index Income Return	real *	ind % iret(j,i)	ind % indiret_ts % iret(j,i)	j between 1 and indtypes, i between ind % iret_ts(j) % beg and ind_data % iret_ts(j) % end	j between 1 and indtypes, i between ind % iret_ts(j) % beg and ind % iret_ts(j) % end	value on date ind % iret_ts(j) % cal % caldt(i)

MNEMONIC	NAME	DATA TYPE	DATA USAGE - SHORTCUT	DATA USAGE - FULL VERSION	INDEX RANGE - SHORTCUT	INDEX RANGE - FULL VERSION	DATE USAGE
tind	Index Total Return Index Level	real *	ind % tind(j,i)	ind % indtind_ts % tind(j,i)	j between 1 and indtypes, i between ind % tind_ts(j) % beg and ind_data % tind_ts(j) % end	j between 1 and indtypes, i between ind % tind_ts(j) % beg and ind % tind_ts(j) % end	value on date ind % tind_ts(j) % cal % caldt(i)
tret	Index Total Return	real *	ind % tret(j,i)	ind % indtret_ts % tret(j,i)	j between 1 and indtypes, i between ind % tret_ts(j) % beg and ind_data % tret_ts(j) % end	j between 1 and indtypes, i between ind % tret_ts(j) % beg and ind % tret_ts(j) % end	value on date ind % tret_ts(j) % cal % caldt(i)
usdcnt	Index Used Count	real *	ind % usdcnt(j,i)	ind % indusdcnt_ts % usdcnt(j,i)	j between 1 and indtypes, i between ind % usdcnt_ts(j) % beg and ind_data % usdcnt_ts(j) % end	j between 1 and indtypes, i between ind % usdcnt_ts(j) % beg and ind % usdcnt_ts(j) % end	value on date ind % usdcnt_ts(j) % cal % caldt(i)
totcnt	Index Total Count	real *	ind % totcnt(j,i)	ind % indtotcnt_ts % totcnt(j,i)	j between 1 and indtypes, i between ind % totcnt_ts(j) % beg and ind_data % totcnt_ts(j) % end	j between 1 and indtypes, i between ind % totcnt_ts(j) % beg and ind % totcnt_ts(j) % end	value on date ind % totcnt_ts(j) % cal % caldt(i)
usdval	Index Used Value	real *	ind % usdval(j,i)	ind % indusdval_ts % usdval(j,i)	j between 1 and indtypes, i between ind % usdval_ts(j) % beg and ind_data % usdval_ts(j) % end	j between 1 and indtypes, i between ind % usdval_ts(j) % beg and ind % usdval_ts(j) % end	value on date ind % usdval_ts(j) % cal % caldt(i)
totval	Index Total Value	real *	ind % totval(j,i)	ind % indtotval_ts % totval(j,i)	j between 1 and indtypes, i between ind % totval_ts(j) % beg and ind_data % totval_ts(j) % end	j between 1 and indtypes, i between ind % totval_ts(j) % beg and ind % totval_ts(j) % end	value on date ind % totval_ts(j) % cal % caldt(i)

FORTRAN-95 STOCK SAMPLE PROGRAMS AND SUBROUTINES

SAMPLE PROGRAMS — *SAMP*.F90

The FORTRAN-95 sample programs provide examples of how to access the CRSPAccess stock file daily or monthly data with universal stock access routines. The 14 stock & indexes sample programs give basic examples of the CRSP access routines, and illustrate tasks while using the access and utility routines. To use a sample program, copy it to your directory from the CRSP sample directory. Edit the program to meet your needs and compile, link, and run. See the CRSPAccess Release Notes for FORTRAN-95 Supported Systems. All sample programs that call on an input file have one available in the sample directory.

The sample programs are written to use either daily or monthly data. To switch between daily and monthly data, change the setid from `STK_DAILY` to `STK_MONTHLY`.

STKSAMP1.F90	Reads all securities sequentially - outputs a security list to a file	*STKSAMP1.F90* makes a sequential pass through the daily file in PERMNO order, retrieves Header data, and creates a company list containing CUSIP - Header, PERMNO, Company Name - Header, Exchange Code - Header, SIC Code - Header, and beginning and ending dates the CRSP file contains time series data for the security. The output is printed into a file called `dcnames.dat`.
STKSAMP2.F90	Reads an input file of historical cusips - outputs current cusips to a file and writes header data to terminal window	*STKSAMP2.F90* reads an historical CUSIP list, with CUSIPs in columns 1-8, from a user-created file called `hcusips.dat`. It outputs a partial company list to the terminal, including all historical CUSIPs found in the monthly file and their corresponding current CUSIPs, names, and last price for each security. It also creates a file, `cusips.dat`, with the current CUSIPs. *STKSAMP2.F95* is particularly suited for updating CUSIP lists after some of the CUSIPs have changed.
STKSAMP3.F90	Reads an input file of permnos - outputs security identification & basic delist information to a file	*STKSAMP3.F90* reads desired PERMNOs from a user-created input file called `permnos.dat` for daily data containing PERMNOs in columns 2-6. It looks for each of the PERMNOs in the indicated database and data are retrieved for each PERMNO in the input file that exactly matches a security on the file. The output file, `outperm.dat` contains PERMNO, name, and returns data for the last date, date of price after delisting, and delisting return are printed for each stock found.
STKSAMP4.F90	Reads securities within a range of sic codes - writes header and portfolio data to terminal window	*STKSAMP4.F90* makes a partial sequential pass through the monthly file by processing all stocks whose most recent SIC Code falls between 2000 and 2100. This range of current SIC codes can easily be changed to select different industry groups. It prints to the terminal a partial `namelist` including initial capitalization and portfolio assignment for each stock found.
STKSAMP5.F90	Reads an input file of historical cusips - outputs header data, returns and compound returns to a file	*STKSAMP5.F90* reads the daily database and extracts data using an input file of historical CUSIPs with beginning and ending date ranges. The input file, `retinp.dat`, has CUSIPs in positions 2-9, begin dates (YYYYMMDD) in positions 11-18 and end dates (YYYYMMDD) in positions 20-27. The output file, `returns.dat`, will contain CUSIP - Header, PERMNO, Calendar dates, and the Compound Return followed by returns for each security over the date range specified in the `retinp.dat` file. If a CUSIP included in `retinp.dat` is not in the CRSP database, the begin date and CUSIP will print to the screen.
STKSAMP6.F90	Year-end capitalization & portfolio assignments for current companies that have traded 3 consecutive years are written to a file	*STKSAMP6.F90* makes a sequential pass through the daily file in PERMNO order, and outputs CUSIP - Header, PERMNO, year-end Capitalizations and decile Portfolio Assignments for all firms that traded in the past three years to a file, `mktcaps.dat`.
STKSAMP7.F90	Reads stock and indexes data - writes daily market indexes within a date range to a file	*STKSAMP7.F90* reads daily stock and indexes data by PERMNO from an input file, `permno_date.dat` which contains PERMNO in columns 2-6 and a start date in columns 8-15. The default relative date range is 3 years before and after the start date specified in `permno_date.dat`. The program writes PERMNO, Calendar Date, Company Name, Return without Dividends for the Stock and Returns without Dividends for INDNO 100080, the NYSE/NYSEMKT/NASDAQ Value-Weighted Market Index over a relative date range to an output file, `permno_returns.dat`. To use this program with indexes not included in the Stock product, you must also subscribe to the Indexes product.

STKSAMP8.F90	Reads stock file for mergers within a date range - outputs header and distribution data to a file	*STKSAMP8.F90* reads the monthly stock database for mergers (delist code 2**) that delisted between 19820101 and 19871231. For all securities found, PERMNO, the CUSIP - Header, Company Name - Header, SIC Code - Header, Delisting Date, Delisting Return, and New PERMNO are written to an output file, delist.dat.
STKSAMP9.F90	Reads stock file for spin-offs within a date range - outputs header, distribution data, and market capitalization to a file	*STKSAMP9.F90* reads the daily stock database for spin-offs (distribution codes 3753 and 3763) between 19871231 and 19891231. For each spinoff found, the PERMNO, Company Name at the time of the spinoff, Distribution Declaration Date, Distribution Amount, and the capitalization portfolio of the security during the year the spin-off occurred are printed to an output file, spinoff.dat.t
STKSAMP10.F90	Reads stock file for nasdaq bid, ask, & number of trades data - outputs permno, company name and nasdsaq time series data to a file	*STKSAMP10.F90* sequentially reads the daily stock database for NASDAQ time series data; Bid, Ask and NASDAQ Number of Trades. Outputs PERMNO, Company Name corresponding to the calendar date, and the NASDAQ time series data to an output file, nmsdata.dat. Note that NASDAQ Number of Trades is a daily- only data item. To use this sample program with monthly data, remove NASDAQ Number of Trades from the output.
STKINDSAMP1.F90	Compare the returns of a company to a specified index	The daily excess returns for a stock compared to an index are calculated over teh specified date range. For each date, the Stock Return, the Index Return and the Negative or Positive Excess Return are written to an output file, excess_return.dat, for the most recent 50 days.
STKINDSAMP2.F90	Compare the returns of a company to its peer group based on market capitalization decile ranking	The returns of the portfolio to which a specified company belongs at each point in time are combined into one-time series to create a peer group index. A time series of excess returns is calculated for the company against this peer group index. The output file, portfolio_xs_ret.dat, is created and contains for each date: Company Return, Index Return and Negative or Positive Excess Returns.
STKINDSAMP3.F90	Compare company returns based on trade-only data	Returns are calculated using trade-only prices, with and without dividends. The output file, trade_ only_ret.dat, contains PERMNO, Calendar Date, Price, Trade-Only Price, Return without Dividends and Return with Dividends.
INDSAMP1.F90	Reads indexes data for multiple indexes - outputs desired data a file	**INDNO INDEX NAME** 1000040 CRSP NYSE/NYSEMKT Value-Weighted Market Index 1000041 CRSP NYSE/NYSEMKT Equal-Weighted Market Index 1000052 S&P 500 Composite Index 1000060 CRSP NASDAQ Value-Weighted Market Index 1000061 CRSP NASDAQ Equal-Weighted Market Index 1000503 NASDAQ Composite Index 1000080 CRSP NYSE/NYSEMKT/NASDAQ Value-Weighted Market Index 1000081 CRSP NYSE/NYSEMKT/NASDAQ Equal-Weighted Market Index 1000502 S&P 500 Composite Index 1000080 CRSP NYSE/NYSEMKT/NASDAQ Value-Weighted Market Index 1000081 CRSP NYSE/NYSEMKT/NASDAQ Equal-Weighted Market Index 1000092 CRSP NYSE/NYSEMKT/NASDAQ Market Capitalization Deciles 1000357 CRSP NYSE/NYSEMKT/NASDAQ Nationa Market Cap-Based Portfolios 1000700 CTI Treasury - CRSP 30 Year Bond Returns 1000709 Consumer Price Index

FORTRAN-95 INCLUDE FILES AND DATA STRUCTURES

crsp.inc defines all structures and constants used by the CRSP FORTRAN-95 access and utility functions, and the function definitions. *crsp.inc* includes several other header files. The primary definitions needed for stock databases are in *f95_ params.inc, f95_cal.inc, f95_datatypes.inc, f95_stock.inc,* and *f95_ind.inc.*

The following list summarizes the individual stock and indexes include files that are included in crsp.inc. All ind- clude files are kept in the `CRSP_INCLUDE` directory.

HEADER FILE	DESCRIPTION
Crsp_params.inc	Contains all parameters used in FORTRAN-95 source programs.

Crsp_data_types.inc	Declares all generic FORTRAN-95 TYPEs that are used to process CRSP stock and index data – exclusive of TYPE crsp_stk and TYPE crsp_ind, together with their immediate SUB-TYPES
Crsp_cal.inc	Contains all FORTRAN-95 data which reflect the CRSP calendar for stock and index data
Crsp_stk.inc	Contains all data and pointers used to support manipulation of CRSP stock data.
Crsp_ind.inc	Contains all data and pointers used to support manipulation of CRSP index data.
Crsp_for_unit.inc	Provides the data structure for managing Fortran unit numbers during run-time execution of FORTRAN-95 programs
All_ind.inc	Includes all FORTRAN-95 data TYPEs required to support manipulation of CRSP index data: crsp_params.inc, crsp_data_types.inc and crsp_ind.inc
All_stk.inc	Includes all FORTRAN-95 data TYPEs required to support manipulation of CRSP stock data: crsp_params.inc, crsp_data_types.inc and crsp_stk.inc
All_stk_ind.inc	Includes all FORTRAN-95 data TYPEs required to support (simultaneous) manipulation of CRSP stock and index data: crsp_params.inc, crsp_ data_types.inc, crsp_stk.inc and crsp_ind.inc

CRSPACCESS FORTRAN-95 LIBRARY

The CRSPAccess FORTRAN-95 Library contains the Application Programming Interface (API) used to access and to process CRSP stock and index data. The library is broken into sections based on the type of operations. The following major groups are available. Each can be further subdivided into subgroups. Functions within subgroups are alphabetical. Each function includes a function prototype, description, list of arguments, return values, side effects, and preconditions for use.

FORTRAN-95 LIBRARY CATEGORY	DESCRIPTION	PAGE
Stock Access Functions	Functions used to load stock data from the database into structures	page 215
Index Access Functions	Functions used to load index data from the database into structures	page 220
General Access Functions	General calendar and access functions	page 221
General Utility Functions	Functions utility to process base CRSPAccess structures	page 222

STOCK ACCESS FUNCTIONS

The following tables list the available functions to access CRSPAccess Stock Data. Standard usage is to employ an open function, followed by successive reads and a close. Different databases and sets can be processed simulta- neously if there is a matching structure defined for each one.

FUNCTION	DESCRIPTION	PAGE
`stock_open`	Opens an Existing Stock Set in a CRSPAccess Database	page 215
`stk_read_permno`	Loads Wanted Stock Data Using CRSP PERMNO as the Key	page 230
`stk_read_cusip`	Loads Wanted Stock Data Using Current CUSIP as the Key	page 217
`stk_read_permco`	Loads Wanted Stock Data Using CRSP PERMCO as the Key	page 217
`stk_read_hcusip`	Loads Wanted Stock Data Using Historical CUSIP as the Key	page 218
`stk_read_siccd`	Loads Wanted Stock Data Using Historical SIC Code as the Key	page 233
`stk_read_ticker`	Loads Wanted Stock Data Using Current Ticker Symbol as the Key	page 219
`stock_close`	Closes a Stock Set	page 219

`stock_open` Opens an Existing Stock Set in a CRSPAccess Database

PROTOTYPE:	`stock_open(TYPE(crsp_stk)stk, TYPE(name_string), POINTER:: user_path, crspnum, setid, wanted, status)`
DESCRIPTION:	opens an existing stock set in a CRSPAccess Database

ARGUMENTS:	stk TYPE(crsp_stk) - data object to be allocated and loaded. user_path TYPE(name_string) - directory path to user's CRSPAccess data; if NULL, default CRSPAccess data are used crspnum – returned value associated with stock set which is opened; used in future data retrievals setid – the set identifier 10 – Daily CRSP Stock Database - STK_DAILY 20 – Monthly CRSP Stock Database - STK_MONTHLY wanted – composite mask indicating which modules will be used. The list below shows the wanted values for the stock modules. The wanted values may be summed, or summary wanted values may be used to open multiple modules. Only modules that are specified by the wanted parameter have memory allocated in stk, and only those modules can be accessed in further data retrieval functions from the database. Note that header data is the default wanted, and it is included with all other options. Individual modules: STK_HEAD header structure STK_EVENTS names, dists, shares, delists, nasdin STK_LOWS lows STK_HIGHS highs STK_PRICES close or bid/ask average STK_RETURNS total returns STK_VOLUMES volumes STK_PORTS portfolios STK_BIDS bids STK_ASKS asks STK_RETXS returns without dividends STK_SPREADS spreads STK_TRADES number of trades or STK_ALTPRCDTS alternate price date STK_OPENPRCS open prices or STK_ALTPRCS alternate prices STK_GROUPS groups Group of modules: STK_INFOS header and event data STK_DDATA price, high, low, volume and returns time series STK_SDATA bids, asks, and number of trades time series STK_STD header, events, prices, high, low, volume, returns, and ports STK_ALL all modules
RETURN VALUES:	status – returned value indicating success/failure (CRSP_SUCCESS/CRSP_FAIL) of stock_open() crspnum – (integer) if opened successfully. This crspnum is used in further data retrieval functions from the database. CRSP_SUCCESS - successful invocation of stock_open() CRSP_FAIL – (integer) if error opening or loading files, if bad parameters, root already opened exclusively, stock set already opened rw, wanted not a subset of set's modules, set does not exist in root, set already opened and structure allocated, error allocating memory for internal or stock structures.
SIDE EFFECTS:	This will load root and stock initialization files if needed, open the root including loading the configuration structure and index structures to memory, opening the address file, and if necessary allocating memory to file buffers, loading the free list, and logging information to the log file. Files will be opened for all wanted modules, associated calendars will be loaded, and wanted stock structures will be allocated.
PRECONDITIONS:	None. The root may already be open under a different set in r mode.

stk_read_permno Loads Wanted Stock Data Using CRSP PERMNO as the Key

PROTOTYPE:	**stk_read_permno (crspnum, stk, setid, permno, permno_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a PERMNO

ARGUMENTS:	crspnum – crspdb root identifier previously established by stock_open() stk – TYPE(crsp_stk) data object to be loaded setid – the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY permno – explicit PERMNO of data to load, or integer that will be loaded with the PERMNO key value found if positional permno_select is used. permno_select – constant to search for the PERMNO in *key, or positional constant: CRSP_EXACT - match the specified key value exactly CRSP_FIRST – the first key in the database CRSP_PREV – the previous key CRSP_LAST – the last key in the database CRSP_SAME – the same key CRSP_NEXT – the next key wanted – mask of flags indicating which data modules to load. See stock_open() for module codes. status - returned value indicating success/failure of stk_read_permno()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key value not found in root CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in stk. The position to be used for the next positional read is reset based on the key value found. If permno_select is a positional qualifier, the actual PERMNO found is loaded to permno. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must have been opened previously. crspnum must have been returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stk_read_cusip Loads Wanted Stock Data Using Current CUSIP as the Key

PROTOTYPE:	**stk_read_cusip (crspnum, stkstk, setid, cusip, cusip_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a security using the CUSIP Identifier - Header (hcusip) as the key
ARGUMENTS:	crspnum - crspdb root identifier returned by stock_open() stk - TYPE(crsp_stk) data object to be loaded setid - the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY cusip - CUSIP - Header to load, or TYPE(cusip_string) data that will be loaded with the CUSIP found if a positional cusip_select is used. cusip_select - qualify matching conditions of key value searches: CRSP_EXACT - accept only an exact match CRSP_BACK - find greatest prior key value if no exact match CRSP_FORWARD – find least following key value if no exact match or positional constant: CRSP_FIRST - the first key in the database CRSP_PREV – the previous key CRSP_LAST – the last key in the database CRSP_SAME – the same key CRSP_NEXT - the next key wanted - mask of flags indicating which data modules to load. See stock_open() for module codes. status - returned value indicating success/failure of stk_read_cusip()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key value not found CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid CUSIP index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location(s) in the stock structure. The position used for the next positional read is reset based on the key value found. If cusip_flag is a positional qualifier, the actual CUSIP Identifier - Header found is loaded to cusip. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stk_read_permco Loads Wanted Stock Data Using CRSP PERMCO as the Key

PROTOTYPE:	**stk_read_permco(crspnum, stk, setid, permco, permco_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a security using PERMCO as the key
ARGUMENTS:	crspnum – crspdb root identifier established by stock_open() stk – TYPE(crsp_stk) data object to be loaded setid – the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY permco – PERMCO to load, or an integer that will be loaded with the key value found if a positional permco_select is used. permco_select – positional qualifier or match qualifier – see stk_read_cus wanted – mask of flags indicating which data modules to load. See stock_open for module codes. status - returned value indicating success/failure of stk_read_permco()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key value not found CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid PERMCO index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key value found. If permco_select is a positional qualifier, the actual PERMCO found is loaded to permno. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stk_read_hcusip Loads Wanted Stock Data Using Historical CUSIP as the Key

PROTOTYPE:	**stk_read_hcusip(crspnum, stk, setid, hcusip, hcusip_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a security using name history CUSIP as the key
ARGUMENTS:	crspnum – crspdb root identifier established by stock_open() stk – TYPE(crsp_stk) data object to be loaded setid – the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY hcusip – historical CUSIP identifier to load, or TYPE(cusip_string) data hcusip_select – positional qualifier or match qualifier– see stk_read_cusip() wanted – mask of flags indicating which data modules to load. See stock_open() for module codes. status - returned value indicating success/failure of stk_read_hcusip()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key value not found CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid CUSIP index value
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key value found. If cusip_flag is a positional qualifier, the actual historical CUSIP found is loaded to cusip. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stk_read_siccd Loads Wanted Stock Data Using Historical SIC Code as the Key

PROTOTYPE:	**stk_read_siccd(crspnum, stk, setid, siccd, siccd_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a security using name history Standard Industrial Classification (SIC) Code (siccd) as the key
ARGUMENTS:	crspnum – crspdb root identifier returned by stock_open() stk – TYPE(crsp_stk) data object to be loaded setid – the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY siccd – siccd to load, or an integer that will be loaded with the key value found if a positional siccd_select is used. siccd_select – positional qualifier or match qualifier– see stk_read_siccd() wanted – mask of flags indicating which data modules to load. See stock_open() for module codes. status - returned value indicating success/failure of stk_read_siccd()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key not found CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid siccd index value
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key value found. If siccd_flag is a positional qualifier, the actual SIC Code found is loaded to siccd. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must be previously opened. crspnum must be returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stk_read_ticker Loads the Wanted Stock Data Using Current Ticker Symbol - Header as the Key

PROTOTYPE:	**stk_read_ticker(crspnum, stk, setid, ticker, ticker_select, wanted, status)**
DESCRIPTION:	loads wanted stock data for a security using Ticker - Header as the key
ARGUMENTS:	crspnum – crspdb root identifier established by stock_open() stk – TYPE(crsp_stk) data object to be loaded setid – the set identifier used (10 - monthly stock data, 20 - daily stock data) STK_DAILY / STK_MONTHLY ticker – pointer to Ticker Symbol - Header to load, or TYPE(ticker-string) data that will be loaded with the key found if a positional ticker_select is used. ticker_select – positional qualifier or match qualifier– see stk_read_ticker() wanted – mask of flags indicating which module data to load. See stock_open() for module codes. status - returned value indicating success/failure of stk_read_ticker()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if ticker not found CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum, error in read, impossible wanted, invalid ticker index
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the stock structure. The position used for the next positional read is reset based on the key found. If ticker_flag is a positional qualifier, the actual header ticker found is loaded to ticker. Data are loaded only to wanted data structures within the range of valid data for the security.
PRECONDITIONS:	The stock set must be previously opened. The crspnum must be returned from a previous stock_open() call. stk must have been passed to a previous stock_open() call. wanted must be a subset of the wanted parameter passed to the stock_open() function.

stock_close Closes a Stock Set

PROTOTYPE:	**stock_close(crspnum, setid)**
DESCRIPTION:	closes a stock set
ARGUMENTS:	crspnum – identifier of crsp database, as returned by stock_open() setid – stock set orignially associated with crspnum at invocation of stock_open()
SIDE EFFECTS:	All stock module files are closed, memory allocated by them is freed. If these are the last modules open in the database, the root is also closed.

PRECONDITIONS:	The `crspnum` and `setid` must be taken from a previous invocation of `stock_open`.
CALL SEQUENCE:	Called by external programs, must be preceded by invocation of `stock_open()`.

INDEX ACCESS FUNCTIONS

The following tables list the available functions to access CRSPAccess indexes data. Standard usage is to use an open function, followed by successive reads and a close. Different databases and sets can be processed simultaneously if there is a matching structure defined for each one.

ACCESS FUNCTION	DESCRIPTION	PAGE
`index_open`	Opens an Existing Index Set in an Existing CRSPAccess Database	page 220
`ind_read_indno`	Loads `Wanted` Data For a CRSP INDNO	page 236
`index_close`	Closes an Indexes Set	page 221

`index_open` Opens an Index Set in an Existing CRSPAccess Database

PROTOTYPE:	**`index_open(ind_data, user_path, crspnum, setid, wanted, status)`**
DESCRIPTION:	opens an index set in an existing `crspdb`. This opens database files, allocates needed memory to a structure, and initializes internal structures so index data can be used.

ARGUMENTS:	`ind_data – TYPE(crsp_ind)` data object to be allocated and loaded
	`crspnum` – returned value assocaited with index set which is opened; used in future data retrievals
	`setid` – the set identifier
	`400` = monthly index groups - `MONTHLY_INDEX_GROUPS`
	`420` = monthly index series - `MONTHLY_INDEX_SERIES`
	`440` = daily index groups - `DAILY_INDEX_GROUPS`
	`460` = daily index series - `DAILY_INDEX_SERIES`
	`wanted` – mask indicating which modules will be used. The list below shows the `wanted` values for the index modules. The `wanted` values may be summed, or summary `wanted` values may be used to open multiple modules. Only modules that are selected in the `wanted` parameter have memory allocated in the index structure and only those modules can be accessed in further access functions to the database.
	`IND_HEAD` header structure and index description
	`IND_REBALS` 2rebalancing information for index groups
	`IND_LISTS` issue lists
	`IND_USDCNTS` portfolio used counts
	`IND_TOTCNTS` portfolio total eligible counts
	`IND_USDVALS` portfolio used weights
	`IND_TOTVALS` portfolio eligible weights
	`IND_TRETURNS` total returns
	`IND_ARETURNS` capital appreciation returns
	`IND_IRETURNS` income returns
	`IND_TLEVELS` total return index levels
	`IND_ALEVELS` capital appreciation index levels
	`IND_ILEVELS` income return index levels
	Symbols are available for common groups of modules. `IND_ALL` selects all the index data.
	`IND_INFO=IND_HEAD+ IND_REBALS+IND_LISTS`
	`IND_RETURNS=IND_TRETURNS+ IND_ARETURNS+IND_IRETURNS`
	`IND_LEVELS=IND_TLEVELS+ IND_ALEVELS+IND_ILEVELS`
	`IND_COUNTS=IND_USDCNTS+ IND_TOTCNTS+IND_USDVALS+IND_TOTVALS`
	`IND_RESULTS=IND_HEAD+ IND_USDCNTS+IND_USDVALS+IND_TRETURNS`
	`IND_ARESULTS=IND_HEAD+ IND_USDCNTS+IND_USDVALS+IND_ARETURNS`
	`IND_IRESULTS=IND_HEAD+ IND_USDCNTS+IND_USDVALS+IND_IRETURNS`
	`IND_STD=IND_HEAD+ IND_COUNTS+IND_TRETURNS+IND_ARETURNS`
	`IND_ALL=IND_INFO+ IND_RETURNS+IND_LEVELS+IND_COUNTS`
	`status` – returned value indicating success/failure of `index_open()`

RETURN VALUES:	CRSP_SUCCESS: successful invoation of index_open() CRSP_FAIL: if error opening or loading files, if bad parameters, root already opened exclusively, index set already opened rw, wanted not a subset of set's modules, set does not exist in root, set already opened and structure allocated, error allocating memory for internal or index structures.

index_open Opens an Index Set in an Existing CRSPAccess Database

PROTOTYPE:	**index_open(ind_data, user_path, crspnum, setid, wanted, status)**
SIDE EFFECTS:	This will load root and index initialization files if needed, open the root including loading the configuration structure and index structures to memory, opening the address file, and if necessary allocating memory to file buffers, loading the free list, and logging information to the log file. Files will be opened for all wanted modules. Associated calendars will be loaded. wanted index structures will be allocated.
PRECONDITIONS:	None; the root may already be open. If a new index structure is passed additional fields may be allocated.

ind_read_indno Loads Wanted Index Data For a CRSP INDNO

PROTOTYPE:	**ind_read_indno(crspnum, ind_data, setid, indno, indno_select, wanted, status)**
DESCRIPTION:	loads wanted index data for an INDNO
ARGUMENTS:	crspnum – crspdb root identifier returned by index_open() ind_data – TYPE(crsp_ind) data object to be allocated and loaded setid – the set identifier used in index_open() indno – explicit INDNO of data to load, or integer that will be loaded with the key value found if a positional indno_flag is used. indno_select – constant to search for the INDNO in key, or positional constant: CRSP_EXACT - match specified key value exactly CRSP_FIRST – the first key in the database CRSP_PREV – the previous key CRSP_LAST – the last key in the database CRSP_SAME – the same key CRSP_NEXT – the next key wanted – mask of flags indicating which module data to load. See index_open() for module codes. status – returned value indicating success/failure of ind_read_indno()
RETURN VALUES:	CRSP_SUCCESS: if data loaded successfully CRSP_NOT_FOUND: if explicit key value not found in database CRSP_EOF: if end-of-file / end-of-data is encountered CRSP_FAIL: if error with bad parameters, invalid or unopened crspnum and setid, error in read, impossible wanted
SIDE EFFECTS:	Data from the wanted modules will be loaded to the proper location in the index structure. The position used for the next positional read is reset based on the key found. If indno_select is a positional qualifier, the actual INDNO found is loaded to indno. Data are loaded only to wanted data structures within the range of valid data for the index.
PRECONDITIONS:	The index set must be previously opened. The crspnum must be returned from a previous index_open() call. ind_data must have been passed to a previous index_open() call. wanted must be a subset of the wanted parameter passed to the index_open() function.

index_close Closes an Index Set

PROTOTYPE:	**index_close (crspnum, setid)**
DESCRIPTION:	close an index set
ARGUMENTS:	crspnum – identifier of the CRSP database, as returned by index_open() setid – identifier of the index set code to close, as used in the open
SIDE EFFECTS:	All index module files are closed, and memory allocated by them in the index structure is freed. If these are the last modules open in the database, the root is also closed.
PRECONDITIONS:	The crspnum and setid must be taken from a previous ind_open() call.

GENERAL ACCESS FUNCTIONS

The CRSPAccess general access functions include error functions and portable file operation functions.

`crsp_allocate_unit` Allocates Unused Unit for FORTRAN-95-95 I/O

PROTOTYPE:	**`crsp_allocate_unit()`**
DESCRIPTION:	Allocates a unique integer value in the range 10-79 for use in FORTRAN-95-95 I/O
ARGUMENTS:	none
RETURN VALUES:	integer unit number not previously allcoated; `-1` – if no unallocated units available
SIDE EFFECTS:	none
PRECONDITIONS:	none

`crsp_deallocate_unit` Deallocates Unit Allocated by `crsp_allocate_unit()`

PROTOTYPE:	**`crsp_deallocate_unit(unit)`**
DESCRIPTION:	deallocates integer unit number allocated by `crsp_allocate_unit()`
ARGUMENTS:	unit: integer unit number allocated by `crsp_allocate_unit()`
RETURN VALUES:	none
SIDE EFFECTS:	none
PRECONDITIONS:	none

`crsp_free_all_units` Deallocates All Units Currently Allocated by `crsp_allcoate_unit()`

PROTOTYPE:	**`crsp_free_all_units`**
DESCRIPTION:	deallocates all units currently allocated by `crsp_allocate_unit()`
ARGUMENTS:	none
RETURN VALUES:	none
SIDE EFFECTS:	none
PRECONDITIONS:	none

GENERAL UTILITY FUNCTIONS

The utility functions operate on the base CRSPAccess data structures and are not specific to a type of data. They include operations on calendars CRSP object structures and general utilities.

Calendar Utility Functions

These functions are used to manipulate calendar data in CRSPAccess databases.

`cal_index` Finds CRSP Calendar Index of Date

PROTOTYPE:	**`cal_index (cal, date)`**
DESCRIPTION:	Finds CRSP Calendar Index of Date
ARGUMENTS:	`cal - TYPE(crsp_cal)` calendar object `date` - YYYYMMDD format date whose index in `cal % caldt` is desired
RETURN VALUES:	index of YYYYMMDD argument in `cal % caldt`, or zero if out of range
SIDE EFFECTS:	matches forward to next valid date in `cal % caldt` if YYYYMMDD argument not found

`date_index` Finds CRSP Calendar Index of Date

PROTOTYPE:	**`date_index (cal, date, option)`**
DESCRIPTION:	Finds CRSP Calendar Index of Date
ARGUMENTS:	`cal` - `TYPE(crsp_cal)` calendar object `date` - YYYYMMDD format date whose index in `cal % caldt` is desired `option` - -1, 0, 1 : match backward, exact, forward
RETURN VALUES:	index of YYYYMMDD argument in `cal % caldt`, or zero if not found
SIDE EFFECTS:	none

`stk_usdate` Index of Calendar Trading Date

PROTOTYPE:	**`stk_usdate(stk, datein, dateout, position)`**
DESCRIPTION:	Index of Calendar Trading Date

ARGUMENTS:	`stk` - TYPE(crsp_stk)must have been used in prior invocation of `stk_open()`
	`datein` - YYYYMMDD format date
	`dateout` - YYYYMMDD formate date, to be loaded
	`position` - integer index value in active `stk % caldt()`

RETURN VALUES:	`dateout` is loaded with the next trading date greater than or equal to `datein` position is the index of dateout in `stk % caldt()`
SIDE EFFECTS:	none

CRSPAccess Stock Utility Functions

These functions can be used to access stock data.

FUNCTION	DESCRIPTION	PAGE
`stk_comp_ret`	Compound Returns	page 238
`stk_curdis`	Finds Distributions Between Specified Dates	page 224
`stk_curnam`	Finds Name Data on Specified Date	page 224
`stk_curndi`	Finds Effective NASDAQ Information Structure on Specified Date	page 224
`stk_curshr`	Finds Shares Outstanding on Specified Date and Calendar Index	page 224
`stk_exrdat`	Restricts Real Array Data Between Selected Dates and by Exchange	page 225
`stk_exrinf`	Restricts Event Data Between Selected Dates and by Exchange	page 225
`stk_exrint`	Restricts Integer Array Data Between Selected Dates and by Exchange	page 225
`stk_loadba`	Loads Bid and Ask Data to Price Arrays	page 225
`stk_loadhl`	Loads Trade Only Data to Price Arrays	page 242
`stk_namrng`	Finds Calendar Index Ranges Corresponding to a Name Structure	page 226
`stk_valexc`	Determines if Exchange Code is Valid	page 226
`xs_ret_calc`	Calculates a Stock Excess Return Over an Index	page 227
`comp_ind_calc`	Calculates a Composition Return	page 227
`stk_ret_calc`	Calculates a Return Based on Trade-Only Prices	page 227

`stk_comp_ret` Compound Returns

PROTOTYPE:	**`compret=stk_compret (retv, begind, endind)`**
DESCRIPTION:	Compound returns
ARGUMENTS:	`retv` - array of `REAL` returns to be compounded `begind` - initial index of range to be compounded `endind` - terminal index of range to be compounded

RETURN VALUES:	REAL compound return over internal begind - endind
SIDE EFFECTS:	none
PRECONDITIONS:	retv must have DIMENSION (0:*); [stk % ret is allocated as (0:maxarr)]

stk_curdis Finds Distribtutions Between Specified Dates

PROTOTYPE:	**stk_curdis (stk, dist_type, begdt, enddt)**
DESCRIPTION:	Finds distributions between two dates
ARGUMENTS:	stk - TYPE(crsp_stk) data object loaded dist_type - integer distribution type required: 1 = declaration date 2 = ex-distribution date 3 = record date 4 = payment date begdt - YYYYMMDD initial date enddt - YYYYMMDD terminal date
RETURN VALUES:	The sequential index values of the distributions in stk % dists which fall in the specifed range begdt - enddt. -1 - if dist_type not in range 1-4 or if begdt > enddt 0 - if no distributions exist (or if distributions in range begdt - enddt have been exhausted.)
SIDE EFFECTS:	none
PRECONDITIONS:	stk must have been loaded with distribution data via invocation of stock_read_xxx()

stk_curnam Finds Name Data on Specified Date

PROTOTYPE:	**curnam=stk_curnam (stk, date)**
DESCRIPTION:	Finds name data on specified date
ARGUMENTS:	stk - TYPE(crsp_stk) data object loaded date - YYYYMMDD date in stk % caldt
RETURN VALUES:	index in stk % names if record valid on date or current record if date follows date of latest name change
SIDE EFFECTS:	none
PRECONDITIONS:	stk must have been loaded with names data via the invocation of stk_read_xxx()

stk_curndi Finds Effective NASDAQ Information Structure on Specified Date

PROTOTYPE:	**stk_curndi (stk, date)**
DESCRIPTION:	Finds effective NASDAQ information structure on specified date
ARGUMENTS:	stk - TYPE(crsp_stk) data object to be allocated and loaded date - YYYYMMDD date in active stk % caldt
RETURN VALUES:	index in stk % nasdin of record valid on date 0 - if no nasdin data is available or date is outside the range of valid nasdin data
SIDE EFFECTS:	none
PRECONDITIONS:	stk must have been laoded with nasdin data via the invocation of stk_read_xxx()

stk_curshr Finds Shares Outstanding on Specified Date and Calendar Index

PROTOTYPE:	**shares=stk_curshr (stk, date)**
DESCRIPTION:	Finds shares outstanding on specified date and calendar index
ARGUMENTS:	stk - TYPE(crsp_stk) data object to be allocated and loaded date - YYYYMMDD date in active stk % caldt

RETURN VALUES:	number of shares outstanding in `stk % shares` of record valid on date 0 - if no shares data are available or if the date is outside the range of valid shares data
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loded with shares data via the invocation of `stk_read_xxx()`

`stk_exrdat` Restricts Real Array Data Between Select Dates and by Exchange

PROTOTYPE:	**`stk_exrdat (stk, excode, begind, endind, array, missval)`**
DESCRIPTION:	Restricts `REAL` array data between select dates and by exchange
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `excode` - CRSP exchange code 1-7 `begind` - initial index for data validation `endind` - terminal index for data validation `array` - array of `REAL` data to be scanned and validated `misval` - value to be substituted in `array()` for days in `begind-endind` range when the security is not trading on the specified exchange
RETURN VALUES:	adjusted `array()`
SIDE EFFECTS:	original values in `array()` may be superseded by `misval`
PRECONDITIONS:	`stk` must have been loded with names data via the invocation of `stk_read_xxx()`

`stk_exrinf` Restricts Event Data Between Selected Dates And by Exchange

PROTOTYPE:	**`stk_exrinf (stk, excode)`**
DESCRIPTION:	Restricts event data between selected dates and by exchange
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `excode` - CRSP exchange code: 1-7
RETURN VALUES:	event arrays are “compressed” to exclude periods when the security did not trade on the specified exchange
SIDE EFFECTS:	parts of events arrays may be overwritten and lost
PRECONDITIONS:	stk must have been loaded with events data via the invocation of `stk_read_xxx()`

`stk_exrint` Restricts Integer Array Data Between Selected Dates by Exchange

PROTOTYPE:	**`stk_exrint (stk, excode, begind, endind, array, missval)`**
DESCRIPTION:	Restricts integer array data between selected dates by exchange
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `excode` - CRSP exchange code: 1-7 `begind` - initial index of data to be screened / reset endind - terminal index of data to be screened / reset array - integer array to be screened / reset `misval` - code to be used as replacement value in array when the security is not trading on the specified exchange
RETURN VALUES:	adjusted `array()`
SIDE EFFECTS:	parts of `array()` data may be overwritten and lost
PRECONDITIONS:	`stk` must have been loaded with names data via the invocation of `stk_read_xxx()`

`stk_loadba` Loads Bid and Ask Data to Price Arrays

PROTOTYPE:	**`stk_loadba (stk)`**
DESCRIPTION:	Loads bid and ask data to price arrays
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded
RETURN VALUES:	First if `stk % prc(i)` is non-negative, `stk % prc(i)`, `stk % bidlo(i)`, and `stk % askhi(i)` are set to 0. Then NMS bid and ask data are loaded into bidlo and askhi. Finally resulting bid-ask averages are copied to the price field.
SIDE EFFECTS:	`prc()`, `bidlo()`, and `askhi()` data may be overwritten and lost

PRECONDITIONS:	stk must have been loaded with price, bid, and ask data via the invocation of stk_read_xxx()

stk_loadhl Loads Trade Only Data to Price Arrays

PROTOTYPE:	**stk_loadhl (stk)**
DESCRIPTION:	Loads trade only data to price arrays
ARGUMENTS:	stk - TYPE(crsp_stk) data object loaded
RETURN VALUES:	stk % prc, stk % bidlo, and stk askhi are reset when price (prc) represents an average of bid and ask. Data remain unchanged only when high, low, and price represent valid trading data.
SIDE EFFECTS:	prc(),bidlo(), and askhi() may be overwritten and lost
PRECONDITIONS:	stk must have been loaded with price, bid and ask data via the invocation of stk_read_xxx()

stk_namrng Finds Calendar Index Ranges Corresponding to a Name Structure

PROTOTYPE:	stk_namrng (stk, ind, bind, eind)
DESCRIPTION:	Finds calendar index ranges corresponding to a name structure
ARGUMENTS:	stk - TYPE(crsp_stk) data object to be allocated and loaded ind - index of stk % names() bind - index in stk % caldt() corresponding to initial valid date of stk % names (ind) eind - index of stk_caldt() corresponding to terminal valid date of stk % names (ind)
RETURN VALUES:	bind, eind 0 - if ind is not a valid index in stk % names()
SIDE EFFECTS:	none
PRECONDITIONS:	skt_data must have been loaded with names data via the invocation of stk_read_xxx()

stk_valexc Determines if Exchange Code is Valid

PROTOTYPE:	**valid=stk_valexc (exhave, exwant)**
DESCRIPTION:	Determines if a given exchange code is valid based on a set of wanted exchanges. When-issued trading is not differentiated from regular-way trading.
ARGUMENTS:	exhave – Exchange Code to validate. Codes are standard CRSP stock Exchange Codes: 1=NYSE 2=NYSEMKT 3=NASDAQ 4=ARCA 31=NYSE when-issued 32=NYSEMKT when-issued 33=NASDAQ when-issued 34=ARCA when-issued exwant – acceptable Exchange Code or codes. If multiple exchanges are valid, exwant is the sum of the individual codes below: 1=NYSE 2=NYSEMKT 4=NASDAQ 8=ARCA
RETURN VALUES:	.TRUE. – if exhave is valid according to exwant .FALSE. – if is not valid according to exwant
SIDE EFFECTS:	none
PRECONDITIONS:	none

xs_ret_calc Calculates Stock Excess Return Over an Index

PROTOTYPE:	**xs_ret_calc (ret_ts, indtret_ts, xs_ret, MISSFLAG, STATUS)**
DESCRIPTION:	loads into TYPE (stk_ret_ts) xs_ret the excess returns for each date in the supplied TIMESERIES: based on the intrinisic "returns" data from the TYPE (stk_ret_ts) subtype of TYPE (crsp_stk), versus the CRSP INDEX returns of the subtype TYPE (ind_tret_ts) of TYPE (crsp_ind)
ARGUMENTS:	TYPE (stk_ret_ts) ret_ts [a component of TYPE (crsp_stk)] TYPE (ind_tret_ts) indtret_ts [a component of TYPE (crsp_ind)] TYPE (stk_ret_ts) xs_ret [a component of TYPE (crsp_stk)] INTEGER MISSFLAG: one of CRSP_KEEP: missing returns in ret_ts are copied to xs_ret, and indtret_ts returns are compounded across the gap, CRSP_SMOOTH: first return following any gap is averaged geometrically so that the entire gap has a constant value, or CRSP_IGNORE: missing returns in ret_ts are treated as zero, missing returns in indtret_ts generate a missing xs_ret data point INTEGER STATUS: CRSP_SUCCESS or CRSP_FAIL
RETURN VALUES:	none, though STATUS indicates success/failure of calculations
SIDE EFFECTS:	data, including missing values – where appropriate – are loaded into xs_ret
PRECONDITIONS:	ret_ts, indtret_ts, and xs_ret should have the same CRSP calendar

comp_ind_calc Calculates a Composite Index Return

PROTOTYPE:	**comp_ind_calc (ind, stk, port_type, composite_index_return, status)**
DESCRIPTION:	loads into TYPE (stk_ret_ts) comp_ind_ret the "composite index" for the security, based on the specified portfolio type and using, for each date, the index for the security's portfolio decile on that date, thereby creating a TIMESERIES of index values which is not a "standard" CRSP data item
ARGUMENTS:	TYPE (crsp_ind) ind TYPE (crsp_stk) INTEGER port_type: the portfolio index to be used for generation of the composite index return TYPE (stk_ret_ts) composite_index_return: loaded with the values of the composite index INTEGER STATUS: CRSP_SUCCESS or CRSP_FAIL
RETURN VALUES:	none, though STATUS indicates success/failure of calculations
SIDE EFFECTS:	composite index returns values are loaded into composite_index_return
PRECONDITIONS:	ind and stk must have the same CRSP calendar

stk_ret_calc Calculates a Stock Return based on Trade Only Prices

PROTOTYPE:	**stk_ret_calc (stk_setid, stk_wanted, stk, trade_only_prc_ts, alt_prc_ts, trade_only_ret_ts, trade_only_retx_ts, trade_only_start, trade_only_end, gap_window, valid_exch)**
DESCRIPTION:	loads into TYPE (stk_ret_ts) trade_only_ret_ts and TYPE (stk_ret_ts) trade_only_retx_ts the "returns data" [with and without dividends, respectively] based on the user's supplied values in TYPE (stk_prc_ts) trade_only_prc_ts over the range of indexes bounded inclusively by the user's values for "trade_only_start" and "trade_only_end"; values are computed in accordance with CRSP conventions for missing values over a maximum allowable interval indicated by "gap_window" and "on–" or "off–" exchange trading

ARGUMENTS:	INTEGER stk_setid: the dataset identifier of TYPE (crsp_stk) used (below) INTEGER stk_wanted: the CRSP identifier of the data loaded into TYPE (crsp_stk) stk (below) TYPE (crsp_stk) stk TYPE (stk_prc_ts) trade_only_prc_ts: TIMESERIES created from TYPE (crsp_stk) by setting to zero all values for "stk % prc_ts" which are negative (i.e., those NOT arising from a valid value for "closing price") TYPE (stk_prc_ts) alt_prc: TIMESERIES to be used to supply "valid" values to supersede zero values present in "trade_only_prc_ts" TYPE (stk_ret_ts) trade_only_ret_ts: TIMESERIES to be loaded with returns calculated from the trade- only- price TIMESERIES TYPE (stk_ret_ts) trade_only_retx_ts: TIMESERIES to be loaded with "returns without dividends" calculated from the trade-only-price TIMESERIES INTEGER trade_only_start: index value indentifying the first data point for which trade-only returns are to be calculated INTEGER trade_only_end: index value identifying the last data point for which trade-only returns are to be calculated INTEGER gap_window: permitted successive missing values in trade_only_prc without computed returns being set to missing (zero is default) INTEGER valid_exch: binary code specifying valid exchange(s): 1 = NYSE, 2 = NYSEMKT, 4 = NASD, 8 = ARCA, 0 = all
RETURN VALUES:	none
SIDE EFFECTS:	computed returns are loaded into trade_only_ret_ts and trade_only_retx_ts
PRECONDITIONS:	"stk" must contain data corresponding to "stk_setid" and "stk_wanted"; trade_only_prc_ts" must contain zero values wherever stk % prc(k) is negative (i.e., represents "bid-asked" average)

CRSPAccess Print Utility Functions

The following functions are FORTRAN-95 print utiltiy functions.

FUNCTION	DESCRIPTION	PAGE
stk_outdat	Outputs Price, Volume, and Return Data	
stk_outdel	Outputs Delisting Data	
stk_outdis	Outputs Distribution Data	
stk_outhdr	Outputs Header Data	
stk_outint	Outputs Data for One Integer Array	
stk_outnam	Outputs Name Data	
stk_outndi	Outputs NASDAQ Information Data	
stk_outnms	Outputs NASDAQ Time Series Data	
stk_outone	Outputs Data for One Real Array	
stk_outshr	Outputs Shares Data	
stk_outyr	Outputs Year and Portfolio Data	

stk_outdat Outputs Price, Volume, and Return Data

PROTOTYPE:	STK_OUTDAT (STK, UNIT, BIND, EIND, STEP)
DESCRIPTION:	Outputs price, volume, and return data
ARGUMENTS:	stk - TYPE(crsp_stk) data object to be allocated and loaded unit - FORTRAN-95 I/O unit open for writing bind - initial index of stk % prc() to be used eind - terminal index of stk % prc() to be used step - "stride" increment for traversal of stk % prc()
RETURN VALUES:	records from stk % bidlo(), stk % askhi(), stk % prc(), stk % vol, and stk % ret() are written to the file open on unit
SIDE EFFECTS:	none
PRECONDITIONS:	stk must have been loaded with prc(), bidlo(), askhi(), vol(), and ret() via invocation of stk_read_xxx()

`stk_outdel` Outputs Delisting Data

PROTOTYPE:	`STK_OUTDEL (STK, UNIT, FIRST, LAST)`
DESCRIPTION:	Outputs delisting data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `first` - initial index of `stk % delist()` to be used `last` - terminal index of `stk % delist()` to be used
RETURN VALUES:	records from `stk % delist()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with delisting records via invocation of `stk_read_xxx()`

`stk_outdis` Outputs Distribution Data

PROTOTYPE:	`STK_OUTDIS (STK, UNIT, FIRST, LAST)`
DESCRIPTION:	Outputs distribution data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `first` - initial index of `stk % dists()` to be used `last` - terminal index of `stk % dists()` to be used
RETURN VALUES:	records from `stk % dists()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with distribution data via invocation of `stk_read_xxx()`

`stk_outhdr` Outputs Header Data

PROTOTYPE:	`STK_OUTHDR (STK, UNIT)`
DESCRIPTION:	Outputs Header data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing
RETURN VALUES:	`HEADER` data values are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with header data via invocation of `stk_read_xxx()`

`stk_outint` Outputs Data for One Integer Array

PROTOTYPE:	`STK_OUTINT (STK, UNIT, TITLE, ARRAY, BIND, EIND, STEP)`
DESCRIPTION:	Outputs data for one integer array
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit - unit` - FORTRAN-95 I/O unit open for writing `title - TYPE(name string)` character string `array` - INTEGER array whose values are to be displayed `bind` - initial index of `array()` to be used `eind` - terminal index of `array()` to be used `step` - "stride" increment for traversal of `stk % array()`
RETURN VALUES:	data from `array()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk % caldt()` must have valid data, generally read via invocation of `stk_read_xxx()`

`stk_outnam` Outputs Name Data

PROTOTYPE:	`STK_OUTNAM (STK, UNIT, FIRST, LAST)`
DESCRIPTION:	Outputs Name data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `first` - initial index of `stk % names()` to be used `last` - terminal index of `stk % names()` to be used
RETURN VALUES:	records from `stk % names()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with names data via invocation of `stk_read_xxx()`

`stk_outndi` Outputs NASDAQ Information Data

PROTOTYPE:	`STK_OUTNDI (STK, UNIT, FIRST, LAST)`
DESCRIPTION:	Outputs NASDAQ information data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `first` - initial index of `stk % nasdin()` to be used `last` - terminal index of `stk % nasdin()` to be used
RETURN VALUES:	records from `stk % nasdin()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with nasdin data via invocation of `stk_read_xxx()`

`stk_outnms` Outputs NASDAQ Time Series Data

PROTOTYPE:	`STK_OUTNMS (STK, UNIT, BIND, EIND, STEP)`
DESCRIPTION:	Outputs NASDAQ data
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `bind` - initial index of `stk % bid(), stk % ask(), stk % numtrd()` to be used `eind` - terminal index of `stk % bid(), stk % ask(), stk % numtrd()` to be used `step` - "stride" increment for traversal of `stk % bid(), stk % ask(), stk % numtrd()`
RETURN VALUES:	records from `stk % bid(), stk % ask(), stk % numtrd()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with `bid(), ask(), numtrd()` data via invocation of `stk_read_xxx()`

`stk_outone` Outputs Data for One Real Array

PROTOTYPE:	`STK_OUTONE (STK, UNIT, TITLE, ARRAY, BIND, EIND, STEP)`
DESCRIPTION:	Outputs data for one real array
ARGUMENTS:	`stk - TYPE(crsp_stk)` data object to be allocated and loaded `unit` - unit - FORTRAN-95 I/O unit open for writing `title - TYPE(name string)` character string `array` - array whose valies are to be displayed `bind` - initial index of `array()` to be used `eind` - terminal index of `array()` to be used `step` - "stride" increment for traversal of `array()`
RETURN VALUES:	data from `array()` are written to the file open on `unit`
SIDE EFFECTS:	none

PRECONDITIONS:	`stk % caldt()` must have valid data, generally read via invocation of `stk_read_xxx()`

`stk_outshr` Outputs Shares Data

PROTOTYPE:	**`STK_OUTSHR (STK, UNIT, FIRST, LAST)`**
DESCRIPTION:	Outputs shares data
ARGUMENTS:	`stk` - `TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `first` - initial index of `stk % shares()` to be used `last` - terminal index of `stk % shares()` to be used
RETURN VALUES:	records from `stk % shares()` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been loaded with shares data via invocation of `stk_read_xxx()`

`stk_outyr` Outputs Year and Portfolio Data

PROTOTYPE:	**`STK_OUT_PORT (STK, UNIT, BIND, EIND, PORT_NUM)`**
DESCRIPTION:	Outputs year and portfolio data
ARGUMENTS:	`stk` - `TYPE(crsp_stk)` data object to be allocated and loaded `unit` - FORTRAN-95 I/O unit open for writing `bind` - initial index of portfolio data to be used `eind` - terminal index of portfolio data to be used `port_num` - index of `stk % port_ts()` to be used
RETURN VALUES:	`portfolio statistics - port` and `stat` are written to the file open on `unit`
SIDE EFFECTS:	none
PRECONDITIONS:	`stk` must have been laoded with portfolio data via invocation of `stk_read_xxx()`