CHICAGO BOOTH

Center for Research in Security Prices

CRSP

PROGRAMMER'S GUIDE

CRSP US Stock & US Indices Databases

CHICAGO BOOTH Center for Research in Security Prices

105 West Adams, Suite 1700
Chicago, IL 60603
Tel: 312.263.6400
Fax: 312.263.6430
Email: Support@crsp.ChicagoBooth.edu

Version CA297.200707.1

CHAPTER 1: INTRODUCTION

1.1 About CRSP

The Center for Research in Security Prices (Prof. Eugene F. Fama, Chairman) has been an integral part of the academic and commercial world of financial and economic research. Since its inception in 1960, CRSP has provided an unparalleled foundation as the leading source for the most comprehensive and accurate historical US databases available. CRSP is a research institute of the Graduate School of Business of the University of Chicago, which has a history of being a catalyst for innovation and progress, and has been a resource for other academic institutions and corporations alike.

In 1959, Louis Engel, vice president of Merrill Lynch, Pierce, Fenner & Smith, called Professor James H. Lorie (PhD 1947; Professor of Business Administration) with an inquiry which resulted in a grant from Merrill Lynch and the establishment of CRSP.

The inquiry developed into a project which involved compiling, cleaning and codifying the prices, dividends and rates of return of all stocks listed and trading on the NYSE since 1926. It resulted in an academic research-grade database that remains invaluable to empirical research due to its breadth, depth, and completeness, and includes CRSP's unique permanent identifiers, allowing for clean and accurate time-series research and event studies.

CRSP files continue to provide a strong foundation for economic forecasting, stock market research, and financial analyses by academic institutions, investment banks, brokerage firms, corporations, banks and government agencies. CRSP provides the following data files: common stocks on the NYSE, AMEX and NASDAQ; CRSP Indices; NASDAQ, and S&P 500 composite indices; NASDAQ and AMEX Industry Indices; US Treasury bonds; Survivor-Bias-Free Mutual Funds; market capitalization reports; proxy graphs for 10K SEC filings and custom datasets. Additionally, CRSP continues to develop new research resources such as the new CRSP/Ziman Real Estate Data Series.

1.2 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 indices. 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 calendars. Multiple `SETID`s 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 contain 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 programming 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 indices, 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 missing.

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.

FORTRAN Programming with CRSPAccess

CRSPAccess supports FORTRAN programming using FORTRAN-95 language standards. The Data Descriptions Guide and the Programmer's Guide describe the data organization, variables, access functions, and sample programs provided for FORTRAN-95 access. Data organization for FORTRAN-95 is now analogous to that used in C, allowing a common structure for all programming access.

With FORTRAN-95, the following enhancements have been added:

- Access to all CRSPAccess stock and indices variables
- A single consistent interface to all available CRSP index data using permanent identifiers
- Clean presentation of multiple component event structure. Use of TYPEs allow clearer organization of events
- Ability to simultaneously access multiple data sets: Stock, Index, Daily and Monthly
- Concurrent access to all of the CRSPAccess portfolio definitions
- Increased precision. FORTRAN-95 supports double precision where supported in CRSPAccess fields.
- Full support for all group data, such as universe inclusion flags for CRSP's historical S&P 500 Constituent list

1.3 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 helvetica* 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 FORTRAN-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_`.
- The text of this document is in Times New Roman. *Italics* and **bold** styles are used to emphasize headings, names, definitions and related functions.

CHAPTER 2: ACCESSING DATA IN C

2.1 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 structureH**6**s, specialized 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 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 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 associated calendars.
- **Arrays** are attached to each object. The array contains the set of observations and is the basic level of programming 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 CRSPDBs. 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 *

OBJECT or Field	Usage	Data Type
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 *
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 indices 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 indices 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	Daily
		20	Monthly
CRSP Indices 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 0^{th} 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 Sixe	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
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

Module	Object	Name	Object Type	Array Type	Data Subtype	Array Structure Sixe	Range Elements on a Security Basis	Elements of a Set Basis	Array Name
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
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
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			

Mnemonic	Name	Data Type	Data Usage	Index Range	Date Usage	Object Usage
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			
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			
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			

Mnemonic	Name	Data Type	Data Usage	Index Range	Date Usage	Object Usage
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
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/AMEX/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 Indices Data

CRSP assigns a Permanent Index Identification Number (`indno`) to access the indices 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 Indices 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 0^{th} 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`	Indices 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` from 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 to `ind.indtypes -1`
`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 to `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 to `ind.indtypes -1`
`IND_TLEVELS` Total 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 to `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_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 to 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 from 0 to ind.indtypes -1

C Language Data Structure for CRSP Indices 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 Indices 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 Indices Structure**	**CRSP_IND_STRUCT**	**ind**			
indhdr	**Indices 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			
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			

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 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			
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			

Mnemonic	Name	C Data Type	C Data Usage	C Index Range	C Date Usage	C Object Type
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			
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
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

Mnemonic	Name	C Data Type	C Data Usage	C Index Range	C Date Usage	C Object Type
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

2.2 C Sample Programs

There are two sample programs provided that can process the CRSP Stock Database using C. These programs can load stock and indices data structures for processing. The sample program code contains additional comment information. See system-dependent C programming instructions at the end of this section for instructions to run sample programs on supported systems. See C usage tables in 3.1 and variables descriptions in the Data Description Guide for possible data usage. Sample programs are included on the Tools and Installation CD.

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, indices 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 indices 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, indices 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 sufficient 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 indices 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 indices 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 indices header file includes all needed header files for CRSP Indices access
`crsp_ind_objects.h`	defines top level `CRSP_IND_STRUCT` structure for Indices Data
`crsp_ind_const.h`	define indices 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

2.3 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 26
Index Access Functions	Functions used to load index data from the database into structures	Page 40
General Access Functions	General calendar and access functions	Page 48
General Utility Functions	Functions utility to process base CRSPAccess structures	Page 54
Data Utility Functions	Functions used to manipulate stock or indices data	Page 83

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 27
`crsp_stk_close`	Closes a Stock Set	Page 27
`crsp_stk_free`	Deallocates Memory and Reinitializes a Stock Set Structure	Page 27
`crsp_stk_init`	Initializes a CRSPAccess Database for Stock Access	Page 28
`crsp_stk_open`	Opens a Stock Set in a CRSPAccess Database	Page 29
`crsp_stk_read`	Loads Wanted Stock Data For a PERMNO	Page 30
`crsp_stk_read_cus`	Loads Wanted Stock Data Using Header CUSIP Identifier, Header as the Key	Page 31
`crsp_stk_read_permco`	Loads Wanted Stock Data Using `PERMCO` as the Key	Page 32
`crsp_stk_read_hcus`	Loads Wanted Stock Data Using Historical CUSIP as the Key	Page 32
`crsp_stk_read_siccd`	Loads Wanted Stock Data Using Historical SIC Code as the Key	Page 33
`crsp_stk_read_ticker`	Loads Wanted Stock Data Using Ticker Symbol, Header as the Key	Page 33
`crsp_stk_read_subset`	Loads Wanted Stock Data for a PERMNO Applying All Subsetting Filters	Page 34
`crsp_stk_read_key`	Loads Wanted Stock Data Using Any Supported Key	Page 34
`crsp_stk_read_key_subset`	Loads Wanted Stock Data Using Supported Key Applying Subsetting Filters	Page 35
`crsp_stk_alloc`	Allocates and Initializes Stock Structures	Page 36
`crsp_stk_copy`	Copies Data from One Stock Structure to Another	Page 36
`crsp_stk_delete`	Deletes Stock Data for an Existing PERMNO	Page 36
`crsp_stk_insert`	Inserts New Stock Data for a PERMNO	Page 37
`crsp_stk_modload`	Allocates and Loads a Module Structure	Page 37
`crsp_stk_newset`	Inserts a Set of Stock Modules to a CRSP Root Directory	Page 37
`crsp_stk_null`	Function to Zero out the Stock Structure Before Used	Page 38
`crsp_stk_update`	Updates Stock Data for an Existing PERMNO	Page 38
`crsp_stk_del_fromset`	Removes Modules from Stock Set from a CRSP Root Directory	Page 38
`crsp_stk_add_toset`	Adds Modules to Stock Set to a CRSP Root Directory	Page 39
`crsp_stk_get_allissues_key`	Get all Issues for a key	Page 39

`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` `16384` 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` – find 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 (page 113).

`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 113).

`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. 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 `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.
Call Sequence:	

`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.
Call Sequence:	

`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
Side Effects:	
Preconditions:	
Call Sequence:	

`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.
Preconditions:	
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.
Call Sequence:	

`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`.
Call Sequence:	

`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
Call Sequence:	

`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.
Call Sequence:	

`crsp_stk_get_issues_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_SUBSET_PARAMLOAD` 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.
Call Sequence:	

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 Indices Set Structure	Page 40
crsp_ind_close	Closes an Indices Set	Page 41
crsp_ind_free	Deallocates Memory And Reinitializes an Indices Set Structure	Page 41
crsp_ind_init	Initializes a CRSPAccess database for Indices Access	Page 41
crsp_ind_open	Opens an Indices Set in a CRSPAccess Database	Page 42
crsp_ind_read	Loads Wanted Data For an Index	Page 43
crsp_ind_alloc	Allocates and Initializes Indices Structures	Page 43
crsp_ind_copy	Copies Data from One Stock Structure to Another	Page 44
crsp_ind_delete	Deletes Indices Data for an Existing INDNO	Page 44
crsp_ind_insert	Inserts New Indices Data for a PERMNO	Page 44
crsp_ind_modload	Allocates and Loads a Module Structure	Page 45
crsp_ind_newset	Inserts a Set of Indices Modules to a Root	Page 45
crsp_ind_null	Function to Zero Out the Index Structure Before it is Used	Page 45
crsp_ind_read_subset	Reads Indices Data for One INDNO Applying Subsets	Page 46
crsp_ind_update	Updates Indices Data for an Existing INDNO	Page 46
crsp_ind_del_fromset	Removes Modules from Indices Set from a Root Directory	Page 47
crsp_ind_add_toset	Adds Modules to Indices Set to a Root Directory	Page 47
crsp_ind_free_ind	Frees Memory for Allocated Indices Structure	Page 47

`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 Indices 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 *indptr, int wanted)`
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` 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 Indices 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 Indices set object lists, allocates memory, and sets pointers
Arguments:	`int crspnum` - identifier of crsp database, as returned by open `int setid` - identifier of the Indices set to allocate `CRSP_IND_STRUCT *ind` - pointer to Indices structure `int wanted` - binary code of modules wanted
Return Values:	`CRSP_SUCCESS` - if successfully initialized and allocated Indices structure `CRSP_FAIL` - if error allocating memory, error in paramters
Side Effects:	Three levels of pointers are allocated in the Indices 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 Indices Structure to Another

Prototype:	`int crsp_ind_copy (CRSP_IND_STRUCT *indtrg, CRSP_IND_STRUCT *indsrc, int wanted)`
Description:	Copy data from one indices structure to another
Arguments:	`CRSP_IND_STRUCT *indtrg` - pointer to indices structure target `CRSP_IND_STRUCT *indsrc` - pointer to indices 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 `maxarrs` that the counterparts in the target.

`crsp_ind_delete` Deletes a PERMNO from an Indices Set

Prototype:	`int crsp_ind_delete (int crspnum, int setid, int key)`
Description:	Deletes a PERMNO from an Indices set
Arguments:	`int crspnum` - identifier of root `int setid` - identifier of the set `int key` - PERMNO to erase from Indices set
Return Values:	`CRSP_SUCCESS` - if successfully closed Indices 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 Indices modules will be set to `NULL`
Preconditions:	The root and Indices 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 Indices 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 Indices 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 Indices structure, data that will be added to the `wanted` modules `int wanted` - a Indices `wanted` parameter indicating which Indices 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 Indices 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
Side Effects:	
Preconditions:	

`crsp_ind_newset` Adds a Set of Indices Modules to a crspdb

Prototype:	**`int crsp_ind_newset (char *root, int setid, int wanted)`**
Description:	Adds a set of Indices modules to a `crspdb`
Arguments:	`char *root` - path of existing crspdb root directory `int setid` - known Indices set number from initialization file `int wanted` - mask determining which Indices modules are supported in the set
Return Values:	`CRSP_SUCCESS` - if the Indices set is added `CRSP_FAIL` - if bad parameters or error manipulating root structures
Side Effects:	`crsp_ind_newset` adds a set of Indices 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`
Preconditions:	

`crsp_ind_read_subset` Loads Wanted Indices 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` indices 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 indices 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 Indices Data for a Key

Prototype:	**`int crsp_ind_update (int crspnum, int setid, int key, CRSP_IND_STRUCT *indptr, int wanted)`**
Description:	Update Indices 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 Indices 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 Indices 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
Side Effects:	
Preconditions:	

`crsp_ind_free_ind` Function to free an Index structure

Prototype:	
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
Side Effects:	
Preconditions:	

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 48
Portable File Operations	These functions call standard I/O functions in the C run time library	page 50

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 OpenVMS systems without changes and incorporate the CRSP error handling function.

Function	Description	Page
crsp_file_append	generic file append for multiple platforms	page 50
crsp_file_close	generic file close for multiple platforms	page 50
crsp_file_fopen	generic file fopen for multiple platforms	page 51
crsp_file_lseek	generic file lseek for multiple platforms	page 51
crsp_file_open	generic file open for multiple platforms	page 51
crsp_file_read	generic file read for multiple platforms	page 52
crsp_file_remove	generic file delete for multiple platforms	page 52
crsp_file_rename	generic file rename for multiple platforms	page 52
crsp_file_search	generic check for existence of a file on multiple platforms	page 52
crsp_file_stamp	generic generation of a unique file name for multiple platforms	page 53
crsp_file_write	generic file write for multiple platforms	page 53
crsp_free	generic memory free for multiple platforms	page 53

`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 `fopen` for options, and OpenVMS documentation for all RMS options.

`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 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 54
Calendar Access Functions	Functions Used to Access CRSP Calendars	Page 58
Compare Function	Functions Used to Compare Data in Two Structures	page 61
Database Information Functions	Find CRSPAccess Database Information	page 82
Object Clear Functions	Functions Used to Clear CRSP Structures	page 73
Object Utility Functions	Functions Used to Manipulate CRSP Object Structures	page 62
String Functions	Functions Used to Manipulate Character Strings	page 67
Object Translation Functions	Functions Used to Map Data to Time Series	page 80
Clear Functions	Functions Used to Clear CRSP Structures	page 73

Calendar Utility Functions

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

Function	Description	Page
`crsp_cal_datecmp`	`CALDT` Date Search	page 54
`crsp_cal_dt2lin`	Transforms YYYYMMDD Format into Linear Date	page 55
`crsp_cal_dt2parts`	Separates the YYYYMMDD Format into Year, Month and Day	page 55
`crsp_cal_lin2dt`	Transfers Linear Date into YYYYMMDD Format	page 55
`crsp_cal_middt`	Finds the Mid-Point Date of a Range	page 55
`crsp_cal_diffdays`	Finds the Number of Calendar Days Between two Dates	page 55
`crsp_cal_link`	Creates a Link to Map Periods of two Calendars	page 56
`crsp_cal_search`	Generic Calendar Date Search	page 56
`crsp_cal_incr`	Increments an Integer Date to the Next Date	page 57
`crsp_cal_decr`	Decrements an Integer Date to the Previous Date	page 57

`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 58
crsp_obj_free_cal	Free memory allocated for a CRSP calendar structure	Page 59
crsp_obj_init_cal	Initialize and allocate a CRSP calendar structure	Page 59
crsp_cal_load	Load a calendar available in a CRSPAccess database	Page 60

`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.

Function	Description	Page
`crsp_cmp_int`	Compares Two Integers	page 61
`crsp_cmp_string`	Compares Two Strings	page 61

`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 62
crsp_obj_verify_arr	Verifies a CRSP Array Object	page 63
crsp_obj_verify_row	Verifies a CRSP Row Object	page 63
crsp_obj_init_ts	Initializes a CRSP Time Series Object	page 63
crsp_obj_init_arr	Initializes a CRSP Array Object	page 64
crsp_obj_init_row	Initializes a CRSP Row Object	page 64
crsp_obj_comp_ts	Compares two CRSP Time Series Objects	page 64
crsp_obj_comp_arr	Compares two CRSP Array Objects	page 65
crsp_obj_comp_row	Compares two CRSP Row Objects	page 65
crsp_obj_free_ts	Frees a CRSP Time Series Object	page 65
crsp_obj_free_arr	Frees a CRSP Array Object	page 65
crsp_obj_free_row	Frees a CRSP Row Object	page 66
crsp_obj_free	Frees a CRSP Object Element Link List	page 66

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 then maxarr in time series 1258: cal pointer cannot be NULL in time series 1259: ndays cannot be > 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
Side Effects:	

String Utilities

These functions can be used to manipulate strings.

Function	Description	Page
crsp_util_convtype	Converts CRSP Constant Names to Integers	page 67
crsp_util_lowercase	Converts Strings to All Lowercase Letters	page 67
crsp_util_strtrim	Removes Trailing Blanks from Strings	page 67
crsp_util_uppercase	Converts Strings to All Uppercase Letters	page 68
crsp_util_squeeze	Removes White Space from Character Strings	page 68
crsp_util_strtoken	Locates the First Delimiter in a String	page 68
crsp_util_cvt_date_mmddyy_i	Converts Character Date String YYMMDD into a Y-2K Compliant Date	page 68
crsp_util_cvt_t_i	Converts a Text String to an Integer	page 69
crsp_util_cvt_t_l	Converts a Numeric Text String into a Long Integer	page 69
crsp_util_cvt_t_f	Converts a Text String to a Floating Point Number	page 69
crsp_util_cvt_t_d	Converts a Text String to a Double Floating Point Number	page 69
crsp_util_cvt_cdate_i	Convert a Character Date String into an Integer	page 70
crsp_util_cvt_i_cdate	Convert Integer Date to Character Date String	page 70
crsp_util_cvt_i_ingdate	Convert an Integer Date (YYYYMMDD) into an Date-Field-Compatible Character String Date	page 70

`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 removeing 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:	strings 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
Preconditions:	

`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
Preconditions:	

`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:	See description. 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 71
crsp_util_copy_arr	Copy CRSP Array Data to Another CRSP Array	page 71
crsp_util_copy_cal2ts	Copy a Calendar to a Time Series	page 72

`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.
Arguments:	

`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.
Arguments:	

`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` 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`.
Arguments:	

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 73
crsp_util_clear_elem	Sets one structure to missing values	page 73
crsp_util_clear_row	Sets a CRSP_ROW to missing values	page 74
crsp_util_clear_ts	Sets a CRSP_TIMESERIES to missing values	page 74
crsp_util_delete_ts	Deletes ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 76
crsp_util_insert_ts	Inserts ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 77
crsp_util_update_ts	Updates ranges from a CRSP_TIMESERIES given a second CRSP_TIMESERIES	page 77
crsp_util_is_missing	Handle missing value problem in CRSP_TIMESERIES structure parameters	page 78
crsp_util_reset_enddts	Resets end date for an array structure	page 78
crsp_util_merge_arr	Merges two array structures to a third, single array	page 78
crsp_util_merge_ts	Merges two time series to a third, single time series	page 79

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 arrtype.
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 `array type` 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_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. See `crsp_util_cmp_string` for values. int epsflag - flag used to compare float fields within structure. See `crsp_util_cmp_float` 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` for values. `int epsflag` - flag used to compare float fields within structure. See `crsp_util_cmp_float` 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
Preconditions:	

`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
Side Effects:	
Preconditions:	

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 80
crsp_util_map_row2ts	Maps a subset of fields from a CRSP_ROW to a CRSP_TIMESERIES	Page 81
crsp_util_map_ts2ts	Maps a subset of fields from one CRSP_TIMESERIES to another	Page 81

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 by the 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 section 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 83
Excess Returns Functions	Functions to Make Excess Returns Calculations	Page 86
Name Mapping Functions	Functions to Map Name History Fields to Time Series	Page 87
NASDAQ Information Mapping Functions	Functions to Map NASDAQ Information Elements to Time Series	Page 90
Returns Functions	Functions to Calculate Returns	Page 92
Shares Functions	Functions to Manipulate Shares Data	Page 97
Stock Print Functions	Functions to Print Specialized Stock Data	Page 100
Translation Functions	Functions to Translate Data to New Time Series	Page 120

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 83
`crsp_adj_map_ts`	Adjusts a Source `CRSP_TIMESERIES` According to an Adjustment Array	Page 84
`crsp_adj_map_arr`	Adjusts a Source `CRSP_ARRAY` According to an Adjustment Array	Page 84
`crsp_adj_stk`	Adjusts all relevant fields in a Source Stock Structure	Page 85

`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 Return 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.

Function	Description	Page
crsp_xs_calc	Builds Excess Returns Time Series	Page 86
crsp_xs_port	Builds Associated Portfolio Returns into a Single Time Series for Excess Returns	Page 86

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 indices returns time series int indtypes – total number of indices 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 indices 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 indices time series must exist prior calling the function and must all verify (see crsp_obj_verify_ts) 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 87
`crsp_map_exchcd`	Map name history exchange codes to a time series	Page 87
`crsp_map_siccd`	Map name history siccd codes to a time series	Page 88
`crsp_map_ncusip`	Map name history name CUSIPs to a time series	Page 88
`crsp_map_ticker`	Map name history tickers to a time series	Page 88
`crsp_map_comnam`	Map name history company names to a time series	Page 89
`crsp_map_shrcls`	Map name history share classes to a time series	Page 89
`crsp_cur_name`	Finds index of name structure on a select date	Page 89

`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 90
crsp_map_nmsind	Map NASDAQ National Market indicator to a time series	Page 90
crsp_map_mmcnt	Map NASDAQ Market Maker count to a time series	Page 91
crsp_map_nsdinx	Map NASDAQ index code to a time series	Page 91

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 92
crsp_ret_calc_del	CRSP Delisting Returns Calculations	Page 93
crsp_ret_calc_one	Returns Calculation for One Return	Page 93
crsp_ret_off_exch	Marks Returns when it is Not Traded on the Exchange	Page 94
crsp_ret_ordinary	Determines if a Distribution Is Considered Ordinary	Page 94
crsp_ret_payments	Calculates Price Factor and Cash Dividend Amounts	Page 94
crsp_stk_ret_append_ts	Appends Return to the End of the Returns Time Series	Page 95
crsp_stk_ret_append_dlret	Appends Delisting Return to the Returns Time Series	Page 95
crsp_stk_delret_params	Parses a Delisting Parameter File	Page 95

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=amex, 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: 1.find if sufficient delisting information exists to calculate a return; if not, use the correct missing value. 2.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. 3.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, AMEX, 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 = AMEX, 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 97
crsp_shr_reimp	Converts Raw Shares to Imputed Shares in Place	Page 97
crsp_shr_num	Returns Shares Outstanding on a Given Date	Page 98
crsp_shr_map	Maps The Imputed Shares Array to a Time Series	Page 98
crsp_shr_raw	Converts Imputed Shares to Raw Shares	Page 99

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`

stk_print Output Functions

These output functions are used to output specific stock data. Some functions expect values set in the STKFLAGS structure, defined in the *crsp_stk_flags.h* header file. C `printf` format statements are defined in the *crsp_stk_format.h* header file. The output functions are used by the *stk_print* utility and are currently not written for general output functionality.

`crsp_stkwrt_dx` Writes Daily Data Information With Shares

Prototype:	`int crsp_stkwrt_dx (FILE *file, CRSP_STK_STRUCT *dd, CRSP_TIMESERIES *sh_ts, int format, int beg, int end)`
Description:	writes daily data information with shares into the file pointer passed as an argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to stock structure `CRSP_TIMESERIES *sh_ts` – pointer to time series structure `int format` – `wanted` format `int beg` – `wanted` beginning date of data `int end` – `wanted` ending date of data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar
Side Effects:	none

`crsp_stkwrt_dr` Writes Return Information

Prototype:	`int crsp_stkwrt_dr (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg, int end)`
Description:	writes return information from daily data structure into the file. File pointer passed as an argument, using given format.
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to stock structure `int format` – `wanted` format for output `int beg` – `wanted` beginning date of data `int end` – `wanted` ending date of data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar
Side Effects:	none

`crsp_stkwrt_r` Writes Return Arrays

Prototype:	`int crsp_stkwrt_r (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg, int end)`
Description:	writes returns array into file pointer passed as an argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to stock data structure `int format` – `wanted` format for output data `int beg` – `wanted` beginning date for output data `int end` – `wanted` ending date for output data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar
Side Effects:	none

`crsp_stkwrt_dd` Writes Daily Data In Formatted Output

Prototype:	`int crsp_stkwrt_dd (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg,int end)`
Description:	writes daily data in formatted output
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to the whole stock structure `int format` – `wanted` format for output `int beg` – `wanted` beginning date of output data `int end` – `wanted` ending date of output data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar or array
Side Effects:	none

`crsp_stkwrt_ds` Writes Daily Data with Index Levels

Prototype:	`int crsp_stkwrt_ds (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg, int end, float baseamt, int basedt)`
Description:	writes daily data info with index levels into the file pointer passed as an argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to stock structure `int format` – `wanted` format `int beg` – `wanted` beginning date of data `int end` – `wanted` ending date of data `float baseamt` – base value `int basedt` – base date
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar
Side Effects:	none

`crsp_stkwrt_nms` Writes NASDAQ National Market Data in Formatted Output

Prototype:	`int crsp_stkwrt_nms (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg,int end)`
Description:	writes `nms` data in formatted output
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to the whole stock structure `int format` – `wanted` format for output `int beg` – `wanted` beginning date of output data `int end` – `wanted` ending date of output data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates or type of calendar or array
Side Effects:	none

`crsp_stkwrt_sh` Writes a Share Structure into a File

Prototype:	`int crsp_stkwrt_sh (FILE *file, CRSP_STK_SHARE *sh, int format)`
Description:	writes share structure into file pointer passed as an argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_SHARE *sh` – pointer to share structure `int format` – `wanted` format for output data
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_hdr1` Writes a Stock Header Structure without a Date Range

Prototype:	`int crsp_stkwrt_hdr1 (FILE *file, CRSP_CAL_DATERANGE *r, CRSP_STK_STRUCT *h, int format)`
Description:	writes stock header structure without date range to given stream
Arguments:	`FILE *file` – pointer to output file `CRSP_CAL_DATERANGE *r` – pointer to calendar dates range structure `CRSP_STK_STRUCT *h` – pointer to stock structure `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_hdr` Writes a Stock Header Structure With Date Ranges

Prototype:	`int crsp_stkwrt_hdr (FILE *file, CRSP_CAL_DATERANGE *r, CRSP_STK_STRUCT *h, int format)`
Description:	writes stock header structure with date ranges to given format
Arguments:	`FILE *file` – pointer to output file `CRSP_CAL_DATERANGE *r` – pointer to calendar dates range structure `CRSP_STK_STRUCT *h` – pointer to stock structure `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_name` Writes a Stock Name Structure

Prototype:	`int crsp_stkwrt_name (FILE *file, CRSP_STK_NAME *n, int format)`
Description:	writes stock name structure into file pointer passed as an argument, in given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_NAME *n` – pointer to stock name structure `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_di` Writes a Distribution Structure

Prototype:	`int crsp_stkwrt_di (FILE *file, CRSP_STK_DIST *di, int format)`
Description:	writes distribution structure into the file pointer passed as an argument, using the given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_DIST *di` – pointer to distribution structure `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_de` Writes Delisting Information

Prototype:	`int crsp_stkwrt_de (FILE *file, CRSP_STK_DELIST *de, int format)`
Description:	writes delisting information into the file pointer passed as an argument, using the given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_DELIST *de` – pointer to stock delisting structure `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_ni` Writes NASDAQ Information

Prototype:	`int crsp_stkwrt_ni (FILE *file, CRSP_STK_NASDIN *n, int format)`
Description:	writes NASDAQ info into file pointer passed as argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_NASDIN *n` – pointer to NASDAQ structure to print `int format` – `wanted` format for output
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

`crsp_stkwrt_it` Writes A CRSPDB Record

Prototype:	`int crsp_stkwrt_it (CRSP_STK_STRUCT *r, struct STKFLAGS *flags)`
Description:	writes a `dbstruct` database record according to flags given. The record is modified for writing.
Arguments:	`CRSP_STK_STRUCT *r` – pointer to the whole stock structure `struct STKFLAGS *flags` – pointer to `STKFLAGS` structure
Return Values:	`CRSP_SUCCESS`: if data has been printed according to flags `CRSP_FAIL`: if an error occurred during writing data
Side Effects:	none
Preconditions:	There should be no `fprintfs` in this function. The output should be done in the called functions

`crsp_stkwrt_dtrstr` Sets Up Date Limits For Different Kinds of Stock Data

Prototype:	`void crsp_stkwrt_dtrstr (CRSP_STK_STRUCT *r, struct STKFLAGS *f)`
Description:	sets up date limits for different kinds of stock data according to dates/indices in `STKFLAGS` structure
Arguments:	`CRSP_STK_STRUCT *r` – pointer to stock structure `struct STKFLAGS *f` – pointer to `STKFLAGS` structure
Return Values:	none
Side Effects:	beginning and ending dates in some stock structures are set

`crsp_stkwrt_portf` Writes Formatted Output for Portfolios

Prototype:	`int crsp_stkwrt_portf (FILE *file, CRSP_STK_STRUCT *stk, OPT_PORTFSTRUCT *portlist, struct STKFLAGS *flags)`
Description:	writes formatted output for portfolios
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *stk` – pointer to stock structure `OPT_PORTFSTRUCT *portlist` – pointer to linked list of desired portfolio structures `struct_STKFLAGS *flags` – pointer to `STKFLAGS` structure
Return Values:	`CRSP_SUCCESS`: if everything is ok `CRSP_FAIL`: if an error in format occurs
Side Effects:	none

`crsp_stkwrt_pdate` Finds the Earliest and the Latest Dates for Available Data

Prototype:	`int crsp_stkwrt_pdate (CRSP_STK_STRUCT *stkptr, struct STKFLAGS *flags, int *start, int *finish)`
Description:	finds the earliest and the latest dates for available data, compares them to `flags.begin_date` and `flags.end_date` and sets up the limits accordingly
Arguments:	`CRSP_STK_STRUCT *stkptr` – pointer to stock structure `struct STKFLAGS *flags` – pointer to `STKFLAGS` structure `int *start, *finish` – values have to be set
Return Values:	`CRSP_SUCCESS`: (integer) no errors `CRSP_FAIL`: (integer) wrong parameters sent
Side Effects:	none

`crsp_stkwrt_popts` Writes a Single Time Series Array

Prototype:	**`int crsp_stkwrt_popts (FILE *file, CRSP_STK_STRUCT *dd, int format, int beg, int end, struct STKFLAGS *flags)`**
Description:	writes a single time series array into file pointer passed as an argument, using given format
Arguments:	`FILE *file` – pointer to output file `CRSP_STK_STRUCT *dd` – pointer to stock structure `int format` – `wanted` format for output `int beg` – `wanted` beginning date for output data `int end` – `wanted` ending date for output data `struct_STKFLAGS *flags` – pointer to `STKFLAGS` structure
Return Values:	`CRSP_SUCCESS`: if data has been printed ok `CRSP_FAIL`: if error in format or dates
Side Effects:	none

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 106
crsp_stk_subset_exch	restricts a stock structure by exchange	Page 107
crsp_stk_subset_shrcd	restricts a stock structure by share code	Page 108
crsp_stk_subset_range	restricts a stock structure by date range	Page 109
crsp_stk_subset_nmsind	restricts a stock structure by NASDAQ National Market status	Page 110
crsp_stk_subset_wi	restricts a stock structure by when-issued status	Page 111
crsp_stk_subset_freq	maps data with a new frequency into a new stock structure	Page 112
crsp_stk_subset_parload	loads a structure of subset parameters (a CRSP_UNIV_PARAM_LOAD structure) used by other subset functions	Page 113
crsp_stk_gen_sum_nasdin	summarizes NASDAQ market maker count	Page 113

`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 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=AMEX, 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 = AMEX 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. The keywords and usage are: (see Parameter Options Specifications for `crsp_stk_subset` utility program, page 118 in the Utilities 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.

Function	Description	Page
`crsp_stk_gen_sum_nasdin`	Summarizes NASDAQ Information Events	Page 113
`crsp_stk_gen_hdr_fromnam`	Resets Header Identification Information	Page 113

`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 114
crsp_stk_delrng_names	Delete ranges of data from names array	Page 115
crsp_stk_delrng_dists	Delete ranges of data from distribution array	Page 115
crsp_stk_delrng_nasdin	Delete ranges of data from NASDAQ Information array	Page 116
crsp_stk_delrng_groups	Delete ranges of data from groups array	Page 116
crsp_stk_delrng_delists	Delete ranges of data from delisting array	Page 115
crsp_stk_delrng_shares	Delete ranges of data from shares array	Page 116
crsp_stk_delrng_resetdt	Reset the header beginning and ending dates	Page 117

`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 117
`crsp_stk_valid_nmsind`	Determines if NASDAQ National Market Indicator is valid	Page 118
`crsp_stk_valid_shrcd`	Determines if Share Code is valid	Page 118
`crsp_stk_valid_shrcd_ld`	Sets up a `CRSP_UNIV_SHRCD` structure for checking valid share codes	Page 119

`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=AMEX 3=NASDAQ 4=ARCA 31=NYSE when-issued 32=AMEX 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=AMEX 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 AMEX 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 Indices; 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 120
crsp_trans_last	Translates Time Series Based On Last Value in Range	Page 121
crsp_trans_first	Translates Time Series Based On First Value in Range	Page 121
crsp_trans_max	Translates Time Series Based On Maximum Value in Range	Page 122
crsp_trans_min	Translates Time Series Based On Minimum Value in Range	Page 122
crsp_trans_average	Translates Time Series Based On Average Value in Range	Page 122
crsp_trans_median	Translates Time Series Based On Median Value in Range	Page 123
crsp_trans_total	Translates Time Series Based On Total Value in Range	Page 123
crsp_trans_last_closest	Translates Time Series Based On Closest Nonmissing Value in Range	Page 124
crsp_trans_last_previous	Translates Time Series Based On Last Nonmissing Value in Range	Page 124
crsp_trans_level	Loads a Target Time Series With Index Level Prices	Page 125
crsp_trans_cumret	Loads a Target Time Series With Cumulative Returns	Page 125
crsp_trans_port	Maps Portfolio Assignments To a New Time Series	Page 125
crsp_trans_stat	Maps Portfolio Statistics To a New Time Series	Page 126
crsp_trans_cap	Loads a Target Time Series With Capitalization Data	Page 126
crsp_trans_gen_prc	General Translation Price Function	Page 127

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` the `subtype` must be `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 3: ACCESSING DATA IN FORTRAN-95

3.1 CRSPAccess 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 programming 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 bolded 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
CRSP_ARRAY	**Structure for storing event-type data**	
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
CRSP_TIMESERIES	**Structure for storing time series data**	
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

OBJECT or Field	Usage	Data Type
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
CRSP_CAL	**TYPE for storing calendar period data**	
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
name	the calendar name in text	CHARACTER[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 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 indices 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 indices 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 Indices 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 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 - 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 Strucutre**						
hcusip	CUSIP - Header	CHARACTER[16]		stk % stkhdr % hcusip			
permno	PERMNO	INTEGER		stk % stkhdr % permno			
permco	PERMCO	INTEGER		stk % stkhdr % permco			
issuno	NASDAQ Issue Number	INTEGER		stk % stkhdr % issuno			
compno	NASDAQ Company Number	INTEGER		stk % stkhdr % compno			
hexcd	Exchange Code - Header	INTEGER		stk % stkhdr % hexcd			
hsiccd	Standard Industrial Classification (SIC) Code - Header	INTEGER		stk % stkhdr % hsiccd			
hshrcd	Share Code - Header	INTEGER		stk % stkhdr % hshrcd			
hnamecd	Name Code - Header	INTEGER		stk % stkhdr % hnamecd			
begdt	Begin of Stock Data	INTEGER		stk % stkhdr % begdt			
enddt	End of Stock Data	INTEGER		stk % stkhdr % enddt			
dlstcd	Delisting Code - Header	INTEGER		stk % stkhdr % dlstcd			
htick	Ticker Symbol - Header	CHARACTER[16]		stk % stkhdr % htick			
hnaics	North American Industry Classification System (NAICS) - Header	CHARACTER[8]		stk % stkhdr % hnaics			
hcomnam	Company Name - Header	CHARACTER[36]		stk % stkhdr % hcomnam			
htsymbol	Trading Ticker Symbol - Header	CHARACTER[12]		stk % stkhdr % htsymbol			
hcntrycd	Country Code - Header	CHARACTER[4]		stk % stkhdr % hcntrycd			
primexch	Primary Exchange - Header	CHARACTER[1]		stk % stkhdr % hprimexch			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
hsubexch	Sub-Exchange - Header	CHARACTER[1]		stk % stkhdr % hsubexch			
trdstat	Trading Status - Header	CHARACTER[1]		stk % stkhdr % htrdstat			
hsecstat	Security Status - Header	CHARACTER[1]		stk % stkhdr % hsecstat			
hshrtype	Share Type - Header	CHARACTER[1]		stk % stkhdr % shrtype			
hissuercd	Issuer Code - Header	CHARACTER[1]		stk % stkhdr % hissuercd			
hinccd	Incorporation Code - Header	CHARACTER[1]		stk % stkhdr % hinccd			
hits	Intermarket Trading System Indicator - Header	CHARACTER[1]		stk % stkhdr % hits			
hdenom	Trading Denomination - Header	CHARACTER[1]		stk % stkhdr % hdenom			
heligcd	Eligibility Code - Header	CHARACTER[1]		stk % stkhdr % heligcd			
hconvcd	Convertible Code - Header	CHARACTER[1]		stk % stkhdr % hconvcd			
hnameflag	Name Flag - Header	CHARACTER[1]		stk % stkhdr % hnameflag			
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	INTEGER	stk % names(i) % namedt	stk % names_arr % names(i) % namedt			
nameenddt	Last Date of Name	INTEGER	stk % names(i) % nameenddt	stk % names_arr % names(i) % nameenddt			
ncusip	CUSIP	CHARACTER[16]	stk % names(i) % ncusip	stk % names_arr % names(i) % ncusip			
ticker	Ticker Symbol	CHARACTER[8]	stk % names(i) % ticker	stk % names_arr % names(i) % ticker			
comnam	Company Name	CHARACTER[36]	stk % names(i) % comnam	stk % names_arr % names(i) % comnam			
shrcls	Share Class	CHARACTER[4]	stk % names(i) % shrcls	stk % names_arr % names(i) % shrcls			
shrcd	Share Code	INTEGER	stk % names(i) % shrcd	stk % names_arr % names(i) % shrcd			
exchcd	Exchange Code	INTEGER	stk % names(i) % exchcd	stk % names_arr % names(i) % exchcd			
siccd	Standard Industrial Classification (SIC) Code	INTEGER	stk % names(i) % siccd	stk % names_arr % names(i) % siccd			
naics	North American Industry Classification System (NAICS) Code	CHARACTER(8)	stk % names(i) % naics	stk % names_arr % names(i) % naics			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
tsymbol	Trading Ticker Symbol	CHARACTER[12]	stk % names(i) % tsymbol	stk % names_arr % names(i) % tsymbol			
cntrycd	Country Code	CHARACTER[4]	stk % names(i) % cntrycd	stk % names_arr % names(i) % cntrycd			
primexch	Primary Exchange	CHARACTER[1]	stk % names(i) % primexch	stk % names_arr % names(i) % primexch			
subexch	Sub-Exchange	CHARACTER[1]	stk % names(i) % subexch	stk % names_arr % names(i) % subexch			
trdstat	Trading Status	CHARACTER[1]	stk % names(i) % trdstat	stk % names_arr % names(i) % trdstat			
secstat	Security Status	CHARACTER[1]	stk % names(i) % secstat	stk % names_arr % names(i) % secstat			
shrtype	Share Type	CHARACTER[1]	stk % names(i) % shrtype	stk % names_arr % names(i) % shrtype			
issuercd	Issuer Code	CHARACTER[1]	stk % names(i) % issuercd	stk % names_arr % names(i) % issuercd			
inccd	Incorporation Code	CHARACTER[1]	stk % names(i) % inccd	stk % names_arr % names(i) % inccd			
its	Intermarket Trading System Indicator	CHARACTER[1]	stk % names(i) % its	stk % names_arr % names(i) % its			
denom	Trading Denomination	CHARACTER[1]	stk % names(i) % denom	stk % names_arr % names(i) % denom			
eligcd	Eligibility Code	CHARACTER[1]	stk % names(i) % eligcd	stk % names_arr % names(i) % eligcd			
convcd	Convertible Code	CHARACTER[1]	stk % names(i) % convcd	stk % names_arr % names(i) % convcd			
nameflag	Name Flag	CHARACTER[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	INTEGER	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	INTEGER	stk % dists(i) % dclrdt	stk % dists_arr % dists(i) % dclrdt			
exdt	Ex-Distribution Date	INTEGER	stk % dists(i) % exdt	stk % dists_arr % dists(i) % exdt			
rcrddt	Record Date	INTEGER	stk % dists(i) % rcrddt	stk % dists_arr % dists(i) % rcrddt			
paydt	Payment Date	INTEGER	stk % dists(i) % paydt	stk % dists_arr % dists(i) % paydt			
acperm	Acquiring PERMNO	INTEGER	stk % dists*(i) % acperm	stk % dists_arr % dists*(i) % acperm			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
accomp	Acquiring PERMCO	INTEGER	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	INTEGER	stk % shares(i) % shrout	stk % shares_arr % shares(i) % shrout			
shrsdt	Shares Outstanding Observation Date	INTEGER	stk % shares(i) % shrsdt	stk % shares_arr % shares(i) % shrsdt			
shrsenddt	Shares Outstanding Observation End Date	INTEGER	stk % shares(i) % shrsenddt	stk % shares_arr % shares(i) % shrsenddt			
shrflg	Shares Outstanding Observation Flag	INTEGER	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	INTEGER	stk % delist(i) % dlstdt	stk % delist_arr % delist(i) % dlstdt			
dlstcd	Delisting Code	INTEGER	stk % delist(i) % dlstcd	stk % delist_arr % delist(i) % dlstcd			
nwperm	New PERMNO	INTEGER	stk % delist(i) % nwperm	stk % delist_arr % delist(i) % nwperm			
nwcomp	New PERMCO	INTEGER	stk % delist(i) % nwcomp	stk % delist_arr % delist(i) % nwcomp			
nextdt	Delisting Date of Next Available Information	INTEGER	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			
dlpdt	Delisting Payment Date	INTEGER	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	INTEGER	stk % nasdin(i) % trtscd	stk % nasdin_arr % nasdin(i) % trtscd			
trtsdt	NASDAQ Traits Date	INTEGER	stk % nasdin(i) % trtsdt	stk % nasdin_arr % nasdin(i) % trtsdt			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
trtsenddt	NASDAQ Traits End Date	INTEGER	stk % nasdin(i) % trtsenddt	stk % nasdin_arr % nasdin(i) % trtsenddt			
nmsind	NASDAQ National Market Indicator	INTEGER	stk % nasdin(i) % nmsind	stk % nasdin_arr % nasdin(i) % nmsind			
mmcnt	Market Maker Count	INTEGER	stk % nasdin(i) % mmcnt	stk % nasdin_arr % nasdin(i) % mmcnt			
nsdinx	NASD Index Code	INTEGER	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	INTEGER	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	INTEGER		stk % group_arr(j) % group(i) % grpdt			
grpenddt	End of Group Data	INTEGER		stk % group_arr(j) % group(i) % grpenddt			
grpflag	Group Flag of Associated Index	INTEGER		stk % group_arr(j) % group(i) % grpflag			
grpsubflag	Group Secondary Flag	INTEGER		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)

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
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)
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	INTEGER *	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)	INTEGER *	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)	INTEGER *	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	INTEGER *	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 / AMEX / 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 universe (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 Indices Data

CRSP assigns a Permanent Index Identification Number (INDNO) to access the indices 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 Indices 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 0^{th} 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
Indices 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 Indices 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 Indices 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 Indices Structure**	**CRSP_IND**	**ind**				
indhdr	**Indices Header Object**						
indno	INDNO	INTEGER		ind % indhdr % indno			
indco	INDCO	INTEGER		ind % indhdr % indco			
primflag	Index Primary Link	INTEGER		ind % indhdr % primflag			
portnum	Portfolio Number if Subset Series	INTEGER		ind % indhdr % portnum			
indname	Index Name	CHARACTER[80]		ind % indhdr % indname			
groupname	Index Group Name	CHARACTER[80]		ind % indhdr % groupname			
method	Index Methodology Description Structure	CRSP_IND_METHOD		ind % indhdr % method			
methcode	Index Method Type Code	INTEGER		ind % indhdr % method % methcode			
primtype	Index Primary Methodology Type	INTEGER		ind % indhdr % method % primtype			
subtype	Index Secondary Methodology Group	INTEGER		ind % indhdr % method % subtype			
wgttype	Index Reweighting Type Flag	INTEGER		ind % indhdr % method % wgttype			
wgtflag	Index Reweighting Timing Flag	INTEGER		ind % indhdr % method % wgtflag			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
flags	Index Exception Handling Flags	CRSP_IND_FLAGS		ind % indhdr % flags			
flagcode	Index Basic Exception Types Code	INTEGER		ind % indhdr % flags % flagcode			
addflag	Index New Issues Flag	INTEGER		ind % indhdr % flags % addflag			
delflag	Index Ineligible Issues Flag	INTEGER		ind % indhdr % flags % delflag			
delretflag	Return of Delisted Issues Flag	INTEGER		ind % indhdr % flags % delretflag			
missflag	Index Missing Data Flag	INTEGER		ind % indhdr % flags % missflag			
partuniv	Index Subset Screening Structure	CRSP_UNIV_PARAM		ind % indhdr % partuniv			
partunivcode	Universe Subset Types Code in a Partition Restriction	INTEGER		ind % indhdr % partuniv % univcode			
begdt	Partition Restriction Beginning Date	INTEGER		ind % indhdr % partuniv % begdt			
enddt	Partition Restriction End Date	INTEGER		ind % indhdr % partuniv % enddt			
wantexch	Valid Exchange Codes in the Universe in a Partition Restriction	INTEGER		ind % indhdr % partuniv % wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in a Partition Restriction	INTEGER		ind % indhdr % partuniv % wantnms			
wantwi	Valid When-Issued Securities in the Universe in a Partition Restriction	INTEGER		ind % indhdr % partuniv % wantwi			
wantinc	Valid Incorporation of Securities in the Universe in a Partition Restriction	INTEGER		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	INTEGER		ind % indhdr % partuniv % shrcd % sccode			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
fstdig	Valid First Digit of Share Code in a Partition Restriction	INTEGER		ind % indhdr % partuniv % shrcd % fstdig			
secdig	Valid Second Digit of Share Code in a Partition Restriction	INTEGER		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	INTEGER		ind % indhdr % induniv % univcode			
begdt	Restriction Begin Date	INTEGER		ind % indhdr % induniv % begdt			
enddt	Restriction End Date	INTEGER		ind % indhdr % induniv % enddt			
wantexch	Valid Exchange Codes in the Universe in an Index Restriction	INTEGER		ind % indhdr % induniv % wantexch			
wantnms	Valid NASDAQ Market Groups in the Universe in an Index Restriction	INTEGER		ind % indhdr % induniv % wantnms			
wantwi	Valid When-Issued Securities in the Universe in an Index Restriction	INTEGER		ind % indhdr % induniv % wantwi			
wantinc	Valid Incorporation of Securities in the Universe in an Index Restriction	INTEGER		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	INTEGER		ind % indhdr % induniv % shrcd % sccode			
fstdig	Valid First Digit of Share Code in an Index Restriction	INTEGER		ind % indhdr % induniv % shrcd % fstdig			
secdig	Valid Second Digit of Share Code in an Index Restriction	INTEGER		ind % indhdr % induniv % shrcd % secdig			
rules	Portfolio Building Rules Structure	CRSP_IND_RULES		ind % indhdr % rules			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
rulecode	Index Basic Rule Types Code	INTEGER		ind % indhdr % rules % rulecode			
buyfnct	Index Function Code for Buy Rules	INTEGER		ind % indhdr % rules % buyfnct			
sellfnct	Index Function Code for Sell Rules	INTEGER		ind % indhdr % rules % sellfnct			
statfnct	Index Function Code for Generating Statistics	INTEGER		ind % indhdr % rules % statfnct			
groupflag	Index Statistic Grouping Code	INTEGER		ind % indhdr % rules % groupflag			
assign	Related Assignment Information	CRSP_IND_ASSIGN		ind % indhdr % assign			
assigncode	Index Basic Assignment Types Code	INTEGER		ind % indhdr % assign % assigncode			
asperm	INDNO of Associated Index	INTEGER		ind % indhdr % assign % asperm			
asport	Portfolio Number in Associated Index	INTEGER		ind % indhdr % assign % asport			
rebalcal	Calendar Identification Number of Rebalancing Calendar	INTEGER		ind % indhdr % assign % rebal_cal			
assigncal	Calendar Identification Number of Assignment Calendar	INTEGER		ind % indhdr % assign % assigncal			
calccal	Calendar Identification Number of Calculations Calendar	INTEGER		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	INTEGER		ind % rebal % rebal(j,i) % rbbegdt			
rbenddt	Index Rebalancing End Date	INTEGER		ind % rebal % rebal(j,i) % rbenddt			

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
usdcnt	Count Used as of Rebalancing	INTEGER		ind % rebal% rebal(j,i) % usdcnt			
maxcnt	Maximum Count During Period	INTEGER		ind % rebal % rebal(j,i) % maxcnt			
totcnt	Count Available as of Rebalancing	INTEGER		ind % rebal % rebal(j,i) % totcnt			
endcnt	Count at End of Rebalancing Period	INTEGER		ind % rebal % rebal(j,i) % endcnt			
minid	Statistic Minimum Identifier	INTEGER		ind % rebal % rebal(j,i) % minid			
maxid	Statistic Maximum Identifier	INTEGER		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 Indices 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	INTEGER	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
permno	Permanent Number of Securities in Index List	INTEGER	ind % list(j,i) % permno	ind % ind_list_arr % list(j,i) % permno			
begdt	First Date Included in List	INTEGER	ind % list(j,i) % begdt	ind % ind_list_arr % list(j,i) % begdt			
enddt	Last Date Included in a List	INTEGER	ind % list(j,i) % enddt	ind % ind_list_arr % list(j,i) % enddt			
subind	Index Subcategory Code	INTEGER	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)

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
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)
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)

Mnemonic	Name	Data Type	Data Usage - Shortcut	Data Usage - Full Version	Index Range - Shortcut	Index Range - Full Version	Date Usage
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)

3.2 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 & indices 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 Indices data - Writes Daily Market Indices within a Date Range to a File**

STKSAMP7.F90 reads daily stock and indices 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/AMEX/NASDAQ Value-Weighted Market Index over a relative date range to an output file, `permno_returns.dat`.

To use this program with indices not included in the Stock product, you must also subscribe to the Indices 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`.

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 Indices Data for multiple indices - Outputs Desired Data a File**

INDNO	Index Name
1000040	CRSP NYSE/AMEX Value-Weighted Market Index
1000041	CRSP NYSE/AMEX 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/AMEX/NASDAQ Value-Weighted Market Index
1000081	CRSP NYSE/AMEX/NASDAQ Equal-Weighted Market Index
1000502	S&P 500 Composite Index
1000080	CRSP NYSE/AMEX/NASDAQ Value-Weighted Market Index
1000081	CRSP NYSE/AMEX/NASDAQ Equal-Weighted Market Index
1000092	CRSP NYSE/AMEX/NASDAQ Market Capitalization Deciles
1000357	CRSP NYSE/AMEX/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 indices 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.

3.3 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 154
Index Access Functions	Functions used to load index data from the database into structures	Page 160
General Access Functions	General calendar and access functions	Page 162
General Utility Functions	Functions utility to process base CRSPAccess structures	Page 163

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 simultaneously if there is a matching structure defined for each one.

Function	Description	Prototype
stock_open	Opens an Existing Stock Set in a CRSPAccess Database	Page 155: **stock_open(TYPE(crsp_stk)stk, TYPE(name_string), POINTER:: user_path, crspnum, setid, wanted, status)**
stk_read_permno	Loads Wanted Stock Data Using CRSP PERMNO as the Key	Page 156: **stk_read_permno (crspnum, stk, setid, permno, permno_select, wanted, status)**
stk_read_cusip	Loads Wanted Stock Data Using Current CUSIP as the Key	Page 157: **stk_read_cusip (crspnum, stkstk, setid, cusip, cusip_select, wanted, status)**
stk_read_permco	Loads Wanted Stock Data Using CRSP PERMCO as the Key	Page 157: **stk_read_permco (crspnum, stk, setid, permco, permco_select, wanted, status)**
stk_read_hcusip	Loads Wanted Stock Data Using Historical CUSIP as the Key	Page 158: **stk_read_hcusip (crspnum, stk, setid, hcusip, hcusip_select, wanted, status)**
stk_read_siccd	Loads Wanted Stock Data Using Historical SIC Code as the Key	Page 158: **stk_read_permco (crspnum, stk, setid, permco, permco_select, wanted, status)**
stk_read_ticker	Loads Wanted Stock Data Using Current Ticker Symbol as the Key	Page 159: **stk_read_siccd (crspnum, stk, setid, siccd, siccd_select, wanted, status)**
stock_close	Closes a Stock Set	Page 159**:stock_close (crspnum, setid)**

`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
	`status` – returned value indicating success/failure (`CRSP_SUCCESS`/`CRSP_FAIL`) of `stock_open()`

Return Values:	`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 indices 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	Prototype
index_open	Opens an Existing Index Set in an Existing CRSPAccess Database	Page 160: **index_open (ind_data, user_path, crspnum, setid, wanted, status)**
ind_read_indno	Loads Wanted Data For a CRSP INDNO	Page 161:
index_close	Closes an Indices Set	Page 161: **index_close (crspnum, setid)**

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.

Function Group	Description	Prototype
`crsp_allocate_unit`	Allocates Unused Unit for FORTRAN-95-95 I/O	Page 162: **`crsp_allocate_unit()`**
`crsp_deallocate_unit`	Deallocates Unit Allocated by `crsp_allocate_unit()`	Page 162: **`crsp_deallocate_unit(unit)`**
`crsp_free_all_units`	Deallocates All Units Currently Allocated by `crsp_allocate_unit()`	Page 162: **`crsp_free_all_units`**

`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.

Function	Description	Prototype
cal_index	Finds CRSP Calendar Index of Date	page 163: **cal_index (cal, date)**
date_index	Finds CRSP Calendar Index of Date	page 163: **date_index (cal, date, option)**
stk_usdate	Subscript of Calendar Trading Date	page 163: **stk_usdate(stk, datein, dateout, position)**

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 backwared, 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	Prototype
stk_comp_ret	Compound Returns	Page 164: **compret=stk_compret (retv, begind, endind)**
stk_curdis	Finds Distributions Between Specified Dates	Page 165: **stk_curdis (stk, dist_type, begdt, enddt)**
stk_curnam	Finds Name Data on Specified Date	Page 165: **curnam=stk_curnam (stk, date)**
stk_curndi	Finds Effective NASDAQ Information Structure on Specified Date	Page 165: **stk_curndi (stk, date)**
stk_curshr	Finds Shares Outstanding on Specified Date and Calendar Index	Page 165: **shares=stk_curshr (stk, date)**
stk_exrdat	Restricts Real Array Data Between Selected Dates and by Exchange	Page 166: **stk_exrdat (stk, excode, begind, endind, array, missval)**
stk_exrinf	Restricts Event Data Between Selected Dates and by Exchange	Page 166: **stk_exrinf (stk, excode)**
stk_exrint	Restricts Integer Array Data Between Selected Dates and by Exchange	Page 166: **stk_exrint (stk, excode, begind, endind, array, missval)**
stk_loadba	Loads Bid and Ask Data to Price Arrays	Page 166: **stk_loadba (stk)**
stk_loadhl	Loads Trade Only Data to Price Arrays	Page 167: **stk_loadhl (stk)**
stk_namrng	Finds Calendar Index Ranges Corresponding to a Name Structure	Page 167: **stk_namrng (stk, ind, bind, eind)**
stk_valexc	Determines if Exchange Code is Valid	Page 167: **valid=stk_valexc (exhave, exwant)**
xs_ret_calc	Calculates a Stock Excess Return Over an Index	Page 168: **xs_ret_calc (ret_ts, indtret_ts, xs_ret, MISSFLAG, STATUS)**
comp_ind_calc	Calculates a Composition Return	Page 168: **comp_ind_calc (ind, stk, port_type, composite_index_return, status)**
stk_ret_calc	Calculates a Return Based on Trade-Only Prices	Page 169: **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)**

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=AMEX 3=NASDAQ 4=ARCA 31=NYSE when-issued 32=AMEX 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=AMEX 4=NASDAQ 8=ARCA
Return Values:	`.TRUE.` – if `exhave` is valid according to `exwant` `.FALSE.` – if `exhave` is not valid according to `exwant`
Side Effects:	none
Preconditions:	none

CRSPAccess Excess Return Functions

`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 indices 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 = AMEX, 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	Prototype
stk_outdat	Outputs Price, Volume, and Return Data	Page 170: **stk_outdat (stk, unit, bind, eind, step)**
stk_outdel	Outputs Delisting Data	Page 170: **stk_outdel (stk, unit, first, last)**
stk_outdis	Outputs Distribution Data	Page 171: **stk_outdis (stk, unit, first, last)**
stk_outhdr	Outputs Header Data	Page 171: **stk_outhdr (stk, unit)**
stk_outint	Outputs Data for One Integer Array	Page 171: **stk_outint (stk, unit, title, array, bind, eind, step)**
stk_outnam	Outputs Name Data	Page 171: **stk_outnam (stk, unit, first, last)**
stk_outndi	Outputs NASDAQ Information Data	Page 172: **stk_outndi (stk, unit, first, last)**
stk_outnms	Outputs NASDAQ Time Series Data	Page 172: **stk_outnms (stk, unit, bind, eind, step)**
stk_outone	Outputs Data for One Real Array	Page 172: **stk_outone (stk, unit, title, array, bind, eind, step)**
stk_outshr	Outputs Shares Data	Page 172: **stk_outshr (stk, unit, first, last)**
stk_outyr	Outputs Year and Portfolio Data	Page 173: **stk_out_port (stk, unit, bind, eind, port_num)**

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()`

C

D

E

F

H

I

S

T