CRISP®

WORF ™Communications

User's

Guide

SQUARE D COMPANY
CRISP AUTOMATION SYSTEMS

CRISP®
WORF™ Communications User's Guide

CRISP® is a registered trademark of Square D Company

I/ONYX® is a registered trademark of Square D Company

WORF™ is a registered trademark of Square D Company

Dedicated to Growth
Committed to Quality

CRISP®
WORF™ Communications User's Guide

Square D Company
5160 Paul G. Blazer Memorial Parkway
Dublin, Ohio 43017
USA

(614) 764-4200

Dedicated to Growth
Committed to Quality

CRISP*connect*™

WORF ™ Network API

User's

Guide

CRISP *Software* *Products*

SQUARE D

GROUPE SCHNEIDER

CRISP*connect* ™

WORF™ Network API
User's Guide

Document number: 500 049 - 002, Rev. 1

Document History

Revision	Date	Pages affected/Description of change
1	11/19/93	Initial Release. ECN 4324

Software Version WORF™ Rev 3.0
CRISP® /32 Rev. 3.0 and later

Table of Contents

Data Format

Symbol Name Syntax

DSL Infomation

Data Source List Functions

Advanced Real-Time Functions

Notes:

Introduction

Intended Audience

This manual is for experienced C or Fortran programmers having extensive experience in using the CRISP®/32 system.

WORF™ users should be knowledgeable about the CRISP/32 language, HISTORIAN system, and at least one current CRISP® workstation product.

Another way to access CRISP/32 real-time data is with the Database Access functions.

Refer to the appropriate CRISP/32 system documentation for more details.

General

This document describes how to use the WORF layer to access CRISP data.

This manual is divided into the following sections.

Section	Description
Operation *(page 5)*	This section defines the basic operational details required for WORF communication, compatibility, and capabilities matrix.
Real-Time Data *(page 7)*	This section defines the functions called to read and modify real-time data.
Trend Data *(page 9)*	This section defines the functions called to read trend data.
HISTORIAN Data *(page 11)*	This section defines the functions called to read and write HISTORIAN and SPC data.
Initialization and Configuration *(page 13)*	This section defines the calls required to allocate memory for default information, create the required internal data structures, and open the necessary communication channels. This section also defines the entries in the configuration table.
Cleanup *(page 17)*	This section defines the functions called and the proper procedures to perform when exiting from WORF.

(Continued on next page.)

Section	Description
Timeout Recovery *(page 19)*	This section defines the methods of timeout recovery used by WORF.
Reading Data *(page 21)*	This section defines the function called to perform Data Source List reads.
Writing Data *(page 23)*	This section defines the function called to perform Data Source List writes.
Data Format *(page 25)*	This section defines the information contained in the data buffers.
Symbol Name Syntax *(page 27)*	This section defines the character strings required by the Data Source List.
DSL Information *(page 31)*	This section defines the primary methods of communicating with WORF.
Data Source List Functions *(page 45)*	This section defines the Data Source List functions and the uses of each function.
Advanced Real-Time Functions *(page 49)*	This section defines the procedures required to read or write data in the Database List.
Miscellaneous *(page 51)*	This section provides miscellaneous information for ASTs, special files, buffer trimming, and node lists.
Compiling and Linking *(page 55)*	This section defines the text and run-time libraries required to compile and link programs using WORF.
Function Descriptions *(page 57)*	This section defines the WORF functions. Each function contains a description, format, operation, and example section.

(Continued on next page.)

Section	Description
Appendix A - Glossary *(page A - 1)*	This section contains definitions of terms used in this manual.
Appendix B - Sample Program C *(page B - 1)*	This section contains a sample program using WORF communications written in C. This example also includes descriptive notes of key elements.
Appendix C - Sample Program Fortran *(page C - 1)*	This section contains a sample program using WORF communications written in Fortran.. This example also includes descriptive notes of key elements.
Appendix D - Status Values *(page D - 1)*	This section defines the WORF message format and the unique returned status values.
Appendix E - Status Messages *(page E - 1)*	This section defines the WORF message format, and the status values that are returned when the debugging output is enabled.
Appendix F - WORF Internal Structure *(page F - 1)*	This section defines the internal structures of WORF. This information is necessary when reading a WORF dump.
Appendix G - Reading a WORF Dump *(page G - 1)*	This section contains information that is helpful when reading a WORF dump. This appendix defines the dump structure and contains an example of a WORF dump.

Notes:

Philosophy

The WORF system gathers information from CRISP/32 systems. It reduces the amount of bookkeeping the client must do and provides great flexibility. The WORF layer communicates with the Database Access Server (DBASRV) and CRISP Access Server (CASRV) on one or more CRISP/32 systems.

The WORF layer provides user access to CRISP data. WORF can access the following three types of information.

Real-time data Real-time data is stored in a CRISP/32 database on a CPU that is running CRISP. WORF accesses real-time data through the Database Access Server (DBASRV) on a CPU that is running CRISP.

Trend data The CWS trend server creates trends on a CPU that is running CRISP. WORF accesses the data by communicating with the DBASRV on a CPU this is running CRISP. WORF does not support trend data from the Basic Workstation trend process.

HISTORIAN data and SPC data HISTORIAN and SPC data is stored in disk files known as 'point files'. WORF accesses this data through the CRISP Access Server (CASRV) on a CPU this is running CRISP. The SPC data must have been stored in HISTORIAN point file format.

WORF is a set of functions that are called by the user. WORF runs as a subroutine to the user's code, and not as a separate process. Each WORF user maintains a separate context.

The user builds a list of symbols known as a Data Source List (DSL). The DSL is a structure internal to WORF; however, the user can examine many of the fields maintained by WORF. The DSL can be read or written by calling various WORF functions.

When data structures or character strings are inputs to WORF, WORF copies the information into its own internal structures. All character strings are folded to uppercase for maximum performance. This allows the user to modify their own data without affecting the operation of WORF. Functions are available to change the operation of WORF in a controlled fashion.

Throughout WORF, the concept of 'handles' is applied. When more than one instance of a particular data structure is possible, such as Data Source Lists, WORF returns a handle to the user so that it can be uniquely identified. Future calls to WORF to manipulate that particular structure require the corresponding handle.

Compatibility and Capabilities Matrix

WORF Version

The version number of WORF is returned by the function WORF_VERSION_NUMBER. Because WORF may change in future releases, users are cautioned to check the version number against the expected value.

Matrix

	WORF Version Number	CRISP/32 Version	Comments
Document # 500049-001	1	2.7	Initial Release of WORF. SPC files accessed by HISTORIAN file access.
	2	2.8	Addition of WORF_CHECK_CRISP_RUNNING and WORF_NL_GET_NODE_STATS_LIST functions. Can retrieve data from both CRISP/32 V2.7 and V2.8 systems.
	3	2.8-21	A node name of "0" is now valid, and refers to the local node.
	WORF Version Number	**CRISP/32 Version**	**Comments**
Document # 500049-002	4	3.0	Addition of WORF_HS_GET_FILE_LIST and WORF_TRANSLATE_ALIAS functions.

NOTE

When running against a version of CRISP/32 before V2.8-29, Real-Time WORF_DSL and WORF_DSL_WRITE operations will operate incorrectly if one or more DSL elements has an `rt.subscript` value that is not valid. Data for those DSL elements will be improperly handled on the CRISP/32 end. The work-around for this problem is to reverse the DSL and mark as invalid all DSL elements with a `status` bit of WORF_DSL_M_BOUND.

General

Real-time data is stored in CRISP/32 databases. The WORF user can read the current values in the database and modify the current values. The functions that perform the I/O are as follows: WORF_DSL_READ, WORF_DSL_WRITE, WORF_DSL_SYMBOL_RAMP, and WORF_DSL_SYMBOL_WRITE.

Resolving

Each symbol in a CRISP/32 database is assigned a record number beginning with 1. It is possible to specify the record number of a desired symbol when adding the symbol to a Data Source List. If the DSL is completely specified, WORF can read or write the data immediately.

Each CRISP/32 database has a 'signature'. To completely specify the DSL, use the function WORF_DBL_ADD_DB to identify the signature value for each database for WORF.

If the WORF user does not know the record number of the desired symbols, the record numbers must be set to zero when the symbols are added to the DSL. This forces a 'resolve'.

If the database has changed, the record numbers of the symbols may have changed. This is detected by comparing the signature declared through WORF_DBL_ADD_DB with the signature of the actual database. If there is a mismatch, this forces a resolve.

Time can be saved by avoiding a resolve. Resolving forces WORF to obtain the record number of each symbol in the entire Data Source List. Minimizing this time is especially important for human-interface products. But, resolving is sometimes unavoidable.

WORF checks the database signatures at the start of each I/O operation. If there is a mismatch, a status of WORF_RESOLVE is returned. WORF_RESOLVE is also returned if DSL contains a record number that is zero.

The user should call WORF_DSL_RESOLVE when an I/O operation returns WORF_RESOLVE. This is not automatic, since the user may want to signal the person or process waiting on the resolve that some time may elapse before normal operation can continue.

Users that do not care to avoid resolving can call WORF_DSL_RESOLVE after adding the last symbol to the DSL.

After a DSL is resolved, WORF_DSL_GET_SYMBOL and WORF_DBL_GET_DB_LIST can be used to retrieve the record numbers and signatures that were located.

Notes:

General

Some CRISP/32 systems have a Color Workstation (CWS) Trend process. The Trend process is selected at system configuration time. The Trend process reads real-time data at regular intervals and builds a short-term history of the data.

To access trend data, the user builds a DSL, specifying the database, symbol name, subscript value, sampling period, and number of samples to average. The user reads the trend data by using WORF_DSL_READ. If the desired trend does not already exist, it is created.

The data returned from a trend is always floating-point, since averaging may occur. At most, TR_MAX_POINTS values will be returned (refer to header file WORF_DEF_USER_x). The first value returned is the most recent point, with successive points going farther into the past.

Trends share data if the specifications of one client match another. The client attempting to add a trend must be aware that the trend may already exist, as well as be prepared if the trend is newly created. Newly created trends have zero elements.

There is no concept of deleting a trend. Trends exist until the Trend process stops, or until the trend region fills up. When the trend region is full, the oldest trend is deleted to make room. Some trends may be made permanent in the Trend process configuration and will never be deleted.

Notes:

General

The CRISP/32 HISTORIAN reads real-time data and builds a long-term history of the data. Collection of data can be time-based, event-based, or both. The Statistical Process Control (SPC) layered product may also create HISTORIAN point files.

HISTORIAN data can be read and modified via WORF. The data is a set of files, specified by a database name and a point name. These usually correspond to the database name and symbol name of the real-time CRISP/32 data source.

Because HISTORIAN data resides on disk, operations may be much slower than accessing real-time or trend data. If the data files are accessed across the network or if the node is heavily loaded, HISTORIAN operations may take minutes to complete.

HISTORIAN data may be floating point, longword, or unsigned longword. The type of data found is available to the user by calling WORF_DSL_GET_SYMBOL after the operation finishes. Longword and unsigned longword data are generally not accessible from CRISP/32 systems prior to CRISP version V2.8-13.

It is not possible to create HISTORIAN point files or create new records in those files using WORF.

The following operations can be performed on HISTORIAN data.

Data-by-Time The user specifies a starting time and a time interval. The data records are interpolated to provide an array of values, uniformly spaced.

Data-at-Times The user specifies an array of times that do not have to be uniformly spaced. The data from the specified records is returned. The times must correspond to actual records in the point file.

Data-by-Record The user specifies a starting time and a number of records. The data records are read, and both the times and values of those records are returned to the user.

Data-Match The user specifies a data value and criterion. The data records are scanned until a match is located. The time and value of the match are returned.

Update-at-Times The user specifies an array of times and data values. The data records at those times are updated with the specified values. Generally, Update-at-Times is performed after a Data-by-Record operation.

General (cont)

Event-Driven The user specifies a starting time, optional data value, and criterion. Beginning at the specified time and/or match, the data records are read, and the times and values of those records are returned to the user. The values from another point file, known as the secondary point file, are returned for the times retrieved from the first point file. The secondary point file must have records of the times from the first point file.

Translate-Root The value of logical name CRISP$HIST_ROOT on the target node is translated and returned.

Get-First-Point Reads the time and value of the first record in the point file.

Get-Last-Point Reads the time and value of the last record in the point file.

Skip-Records The user specifies the time of a record in the point file and a record count. The time and value of a record is returned, the record being the record count away from the specified record.

Get-File-List Obtains list of historian point files.

SPC Data

SPC is a CRISP/32 layered product, separately licensed from CRISP/32. SPC captures statistical information. One option of SPC is to store the information as HISTORIAN point files.

If the HISTORIAN option is chosen, the data collected is indistinguishable from HISTORIAN data. The same operations can be performed on SPC data as on HISTORIAN data.

⚠ CAUTION

It is recommended that the user should only read the SPC data. Updating SPC data (operation Update-at-Times) may cause improper operation of SPC.

SPC produces 29 different 'databases', SPC010 through SPC290. In each database, SPC produces a 'symbol' for each SPC characteristic. The sampled data, as captured by SPC, is database SPC290. Other databases hold internal SPC information and are subject to change without notice.

Initialization

The very first call to any routine in the WORF layer is WORF_CONFIG. If the user does not explicitly call WORF_CONFIG, WORF_INIT calls it. WORF_CONFIG allocates memory for a configuration table and then fills it in with default information. The user specifies a bit mask of options.

After calling WORF_CONFIG and before calling any other functions, WORF_INIT must be called. This function creates the necessary internal data structures and opens necessary channels. WORF_INIT must be called only once by the user.

WORF has the capability of communicating with CRISP/32systems (via the DBASRV and CASRV processes) over two paths: the Software Bus and IEEE 802.3 (Ethernet). When both paths are available, WORF automatically selects the required path.

The following resources are required.

2 Event flags
3 Timer Queue Entries
2 AST quota

Software Bus

WORF_INIT connects to the Software Bus using information built by WORF_CONFIG. If the CRISP Software Bus run-time library (CRISPSWBRTL) is not located or if attempting to connect to the Software Bus returns a *no such global section* error ,WORF does not attempt to use the Software Bus path.

Other errors from the attempt to connect to the Software Bus cause WORF_INIT to return a warning code as its status code.

Connecting the software bus requires privileges of SYSLCK and SYSGBL. If the client process is in the same group as CRISP, GROUP privilege is also required. If the groups are different, WORLD privilege is required.

The following additional resources are required for Software Bus Communications.

2 Timer Queue Entries
2 AST Quota

IEEE 802.3 (Ethernet)

The IEEE 802.3 communications path is enabled by having a read-ahead buffer count of greater than zero (specified as a configuration parameter in the structure returned by WORF_CONFIG). WORF creates channels to logical names CRISP$NET00 and CRISP$NET01 (if available).

The number of read-ahead buffers should be proportionate to the highest number of databases of all the Data Source Lists used. WORF communicates with servers in parallel, one read-ahead buffer for each database.

Having fewer read-ahead buffers will degrade performance slightly since it will cause unnecessary retries. In severe cases, the server may appear to have timed out. No advantage is gained from having more read-ahead buffers than necessary.

Each read-ahead buffer deducts from the BYTLM quota of the caller. If both network devices are available, the number of read-aheads actually used is twice the number specified in the call to WORF_INIT. Currently, each read-ahead buffer consumes 1648 bytes of BYTLM.

If an error condition occurs while WORF is creating a channel to a network device, WORF does not attempt to use that device. If channels cannot be created to either network device, WORF_INIT returns a warning status value.

If one or more channels are already open on a network device (channels that the user opened), WORF_INIT returns a warning status value. WORF will still work. Any messages received that are not from pending WORF transactions are passed to the user's AST routine, if one was specified when the network was created.

⚠ CAUTION

Since WORF uses asynchronous communications, any use that the client makes of the network channels must also be asynchronous.

Using the network requires a privilege of NETMBX.

The following additional resources are needed for IEEE 802.3 communications.

Event Flags:	1 per network
Timer Queue Entries:	`init.ieee_buffers - 1`
AST Quota:	`2X (init.ieee_buffers - 1)`
BIOLM Quota:	`init.ieee_buffers + 1`

Advanced Information

WORF_CONFIG returns the address of the configuration table.

Knowledgeable users can modify the table to change the action of WORF. Entries in the table are as follows.

`init.options`	Configuration options bits. None are defined.
`init.timeout`	If a server does not respond within 'timeout' milliseconds, it has timed out. A value of zero means that no timeout can occur. Default: 1000 milliseconds.
`init.retries`	The number of attempts after the initial attempt to transact with a server. After the number of retries has been reached, the server is marked as timed out. Default: 3.
`init.connect_name_dx`	Software Bus connect name. If NULL or length 0, the process ID is used. Default: NULL.
`init.totalsize`	Total size of SWB message area, in 512-byte pages. Default: 20.
`init.staticsize`	Size of SWB static message area, in 512-byte pages. Default: 0.
`init.num_queues`	Number of SWB queues to open. If < 2, WORF will not connect to the Software Bus. Default: 2.
`init.condx`	Address of the Software Bus connect descriptor. Set by WORF_INIT and WORF_SWB_RECONNECT. If NULL, WORF did not connect to the SWB.
`init.ieee_buffers`	Number of IEEE 802.3 read-ahead buffers to allocate per channel that is opened. Default: 2.

This is the absolute minimum number for communications. Improved performance may be possible with a larger value, if sufficient BYTLM quota is available.

`init.dbasrv_sap`	IEEE 802.3 SAP to use in communicating with the Database Access Server (real-time and trend data). Default: 12.
`init.casrv_sap`	IEEE 802.3 SAP to use in communicating with the CRISP Access Server (HISTORIAN and SPC data). Default: 4.

Advanced Information (cont)

`init2.alias_file_name_dx`

Name of WORF alias file. If null or length 0, CRISP$CFG:WORF_ALIAS.DAT is used. (Available starting in V2.8-23).

`dynamic.recovery_time`

The number of seconds that should elapse between automatic attempts to contact timed-out servers. Set to 0 to disable automatic timeout recovery. Default: 0.

`dynamic.debug_output` If non-zero, special status messages are signalled to the output. When zero, transactions with multiple servers are performed in parallel. When non-zero, only one transaction at a time is performed. Default: 0.

`dynamic.dump_output` If non-zero, all internal WORF structures are dumped to a file when WORF_STOP is called or when the program exits. Default: 0.

The 'init' entries can only be modified by the user after calling WORF_CONFIG and before calling WORF_INIT. Changes made by the user after calling WORF_INIT, for the Software Bus entries, will only be made effective by calling WORF_SWB_RECONNECT. Changes to `init.ieee_buffers` can be made effective by calling WORF_NETWORK_RECONNECT. Changes to `init.timeout` and `init.retries` can be made effective by calling WORF_CONFIG_TIMEOUT.

The 'dynamic' entries may be changed 'on-the-fly' at any time. The current value is checked as required.

The client may share use of the Software Bus with WORF. The user may use any queue other than 2 for their own purposes. WORF uses queue 2 for its communications; therefore, the user may not use queue 2.

The size of the SWB area (`totalsize - staticsize`) should be at least 20 pages for normal operation of WORF.

Because the client may be hibernating, expecting to be awakened when an SWB message is received, WORF scans the other queues before returning from each of its functions that access the Software Bus. If messages are pending, WORF will perform a SYS$WAKE before returning.

The user can enable or disable the communications paths of WORF dynamically using WORF_PATH_SELECT. This function can force WORF to use a specific IEEE 802.3 network, for example.

Cleanup

Calling WORF_STOP aborts all pending I/O on the network channels, deletes all data structures, closes all the network channels, and disconnects from the Software Bus. WORF_STOP is automatically called by an exit handler established by WORF_INIT.

Exit handlers are processed in reverse order of declaration. Therefore, if some part of the user's exit handler requires WORF, the exit handler must be declared after calling WORF_INIT.

Processes connected to the Software Bus are expected to respond to exit messages received on the Software Bus. Users of WORF must either check for exit messages in Software Bus queue 1, or call WORF_CHECK_FOR_EXIT. A process that is 'hibernating' is 'awakened' when a Software Bus message has been received. This is an indication to the client to call WORF_CHECK_FOR_EXIT. The client must check for exit messages on a regular basis and not wait a long time between checks.

Notes:

Timeout Detection

The status value WORF_CHECK from functions such as WORF_DSL_READ (but not WORF_DSL_RESOLVE) indicates that one or more symbols in a DSL have a problem. Traversing the list (using WORF_DSL_TRAVERSE) might reveal symbols with the WORF_ERR_M_TIMEOUT bit set. This indicates that the associated node is timed out.

Timeouts occur for many reasons, including the following.

- The communications path required is not available or was disabled by WORF_PATH_SELECT.
- The server on the CRISP system is heavily loaded and cannot respond in the timeout period.
- The server has stopped operating for some reason (for example, CRISP/32 may have been stopped).
- The node name was specified incorrectly.
- The Data Source List may require more data to be sent than the server can send. Refer to the DSL Size subsection of the Miscellaneous section.

If the timeout time specified in the WORF configuration (`init.timeout`) is zero, no timeout can occur. WORF will wait forever for the server to respond.

The timeout time should be chosen carefully. A short timeout time with a large number of retries is generally less successful than a long timeout time with a few retries.

Although HISTORIAN and SPC DSLs may take a long time to complete, the timeout value does not have to be adjusted. WORF queries the server once per second for completed requests.

Node Not Located

A special condition occurs when a node could not be located. The first I/O operation attempts to locate the node. If the locate fails, the DSL elements are marked as invalid, with an `error_code` of WORF_ERR_M_NONODE. Further I/O operations do not attempt to locate the node, which improves performance.

If the node becomes available, WORF will not detect this condition. Any symbols marked with WORF_ERR_M_NONODE will not be updated during I/O operations.

To recover from this condition, the nodes that have not been located must be located in a separate operation. This can happen automatically or manually, as shown on the next page.

Automatic Recovery

The user may enable and disable automatic node timeout recovery by setting configuration value `dynamic.recovery_time`. A changed value of `dynamic.recovery_time` is detected by WORF when a node times out during communications, or when timeout recovery occurs.

If `dynamic.recovery_time` is not zero, WORF causes an AST to occur every 'n' seconds, where 'n' is the value of `dynamic.recovery_time`. Nodes that are timed out are located during the AST.

The automatic timeout recovery time should be on the order of one minute. Very short times may cause a high level of multicast traffic on a IEEE 802.3 network.

Manual Recovery

If the user sets the value of `recovery_time` to zero, automatic node timeout recovery is disabled. In this case, the user will want to cause recovery from timeout conditions by calling WORF_TIMEOUT_RECOVERY. This function can be called at AST level, even while I/O is in progress.

Post-Recovery Actions Real-Time Data

If symbols are not resolved and the node they refer to recovers from a timeout, call WORF_DSL_RESOLVE before those symbols are read or written. Not resolving causes the read or write to return a status of WORF_RESOLVE.

If symbols are resolved and the node they refer to recovers from a timeout, the next read or write operation will update the symbols if the database signatures match. If the signatures do not match, the operation will return a status of WORF_RESOLVE.

Post-Recovery Actions Other Data

No special actions are required. The operation updates the symbols.

Reading Data

The user calls WORF_DSL_READ with a Data Source List handle. WORF begins transactions with servers as required, collecting all the data for the DSL.

The data collected is put into the user's buffers.

The values of `status` and `error_code` for each DSL element are updated as appropriate.

If one or more symbols in the DSL are invalid or have overrun data buffers, the function returns a status of WORF_CHECK. The user can mark entries as invalid (using WORF_DSL_SET_STATUS). Entries marked, do not cause WORF_CHECK status values to be returned by future calls to WORF_DSL_READ.

The following bits of `status` being set cause a WORF_CHECK status.

WORF_DSL_M_INVALID
WORF_DSL_M_OVERRUN
WORF_DSL_M_BOUND
WORF_DSL_M_BOTH
WORF_DSL_M_NOUPDATE
WORF_DSL_M_NULL

NOTE

This function does not return control to the caller until the data has been read or until the function is unable to continue its operations.

Real-Time DSLs

If any symbols have subscripts that are out of bounds, WORF_DSL_READ returns a status of WORF_CHECK.

A status of WORF_RESOLVE indicates that WORF_DSL_RESOLVE must be called.

Trend DSLs

For trend DSLs, the first call to WORF_DSL_READ causes the trends to be created, unless they had already been created by another client. The number of points returned may be zero or non-zero.

HISTORIAN and SPC DSLs

The operations to be performed are queued to the server(s). Because disk access is required at each server, the time to complete may be long.

WORF_DSL_READ returns a status of WORF_BADOPS if any of the DSL elements have an operation type of HS_UPDATE_AT_TIMES.

Notes:

Writing Data

The user calls WORF_DSL_WRITE with a Data Source List handle. WORF begins transactions with servers as required, sending all the data for the DSL.

The data is taken from the user's buffers.

The values of `status` and `error_code` for each DSL element are updated as appropriate.

If one or more symbols in the DSL are invalid or have overrun data buffers, the function returns a status of WORF_CHECK. The user can mark entries as invalid (using WORF_DSL_SET_STATUS). Entries marked do not cause WORF_CHECK status values to be returned by future calls to the function.

The following bits of `status` being set cause a WORF_CHECK status.

WORF_DSL_M_INVALID
WORF_DSL_M_OVERRUN
WORF_DSL_M_BOUND
WORF_DSL_M_BOTH
WORF_DSL_M_NOUPDATE
WORF_DSL_M_NULL

NOTE

This function does not return control to the caller until the data has been written or until the function is unable to continue its operations.

Real-Time DSLs

If any symbols have subscripts that are out of bounds, the function returns a status of WORF_CHECK.

A status of WORF_RESOLVE indicates that WORF_DSL_RESOLVE must be called.

Because WORF may sort the Data Source List into a different order than the symbols were added by the user, users must not create Data Source Lists that are order-dependent. For example, the user creates a DSL with the following elements.

```
DB_1:SUB_1
DB_1:X_ARRAY(SUB_1)    (SUB_1 is a variable subscript)
```

When writing this list with WORF_DSL_WRITE, WORF makes no guarantees that `SUB_1` is written before `X_ARRAY`. The results of writing this list are undefined.

Real-Time DSLs (cont)

The user may call WORF_DSL_SYMBOL_WRITE on symbol `SUB_1` and then `X_ARRAY(SUB_1)`. This causes the subscripting variable to be updated first and then the array element is updated.

Trend DSLs

Trend DSLs cannot be written and return a status of WORF_WRONGTYPE.

HISTORIAN and SPC DSLs

The operations to be performed are queued to the server(s). Because disk access is required at each server, the time to complete may be long.

WORF_DSL_WRITE returns a status of WORF_BADOPS unless all the DSL elements have an operation type of HS_UPDATE_AT_TIMES.

⚠ CAUTION

The order in which the update is performed is not guaranteed.

Writing Individual Symbols

Functions WORF_DSL_SYMBOL_RAMP and WORF_DSL_SYMBOL_WRITE are available to modify the value of an individual symbol in a real-time DSL. The functions return an error code for other DSL types.

Because several different clients and processes can be modifying CRISP data simultaneously, these two functions act as indivisible operations. WORF_DSL_SYMBOL_WRITE writes a single value to the CRISP database. WORF_DSL_SYMBOL_RAMP adds or subtracts a value from the symbol, while checking that the resulting value is within an allowable range.

The value of `rt.transfer_count` is ignored for these functions. One and only one element is transferred.

Real-Time DSLs

The information in the user's data buffers depends on the data type of the symbol. The structures located in DBA$DBDEF_USER_x are as follows.

Type Value from CSP$VARTYPEDEF_USER_X	Structure Name from DBA$DBDEF_USER_X	Structure
DSC$_STRING	(None)	2-byte unsigned string length Body of string
DSC$_LOGICAL	LOGICAL	1-byte value: low-order bit of 1 indicates true, a low-order bit of 0 indicates false. 1-byte status value
DSC$_NUMERIC	NUMERIC	2-byte integer value
DSC$_LONGWORD	LONGWORD	4-byte integer value
DSC$_FLOAT	F_FLOAT	4-byte value, VAX floating-point format
DSC$_TIMER	TIMER	2-byte reset value 2-byte tickdown value 1-byte countdown value 1-byte status value
DSC$_COUNTER	COUNTER	2-byte reset value 2-byte tickdown value

When `rt.transfer_count` is greater than 1, the data buffer contains an array of values. The values are consecutive in the buffer with no intervening spaces or padding.

HISTORIAN/SPC DSLs

HISTORIAN times are in seconds past midnight, January 1, 1970.

The values in `hs.value_array_dx` and `hs.secondary_value_array_dx` depend on the type of point files accessed. The value of `hs.data_type` indicates the type of data that was accessed. The user cannot specify the data type in advance.

Structure HIST_VALUE (refer to header file WORF_DEF_USER_x) has been defined so that one of the interpretations can be selected as required.

Trend DSLs

The information placed into the user's data buffers are arrays of floats, representing the desired averaging. There can be a maximum of 512 (TR_MAX_POINTS) data values returned.

Notes:

General

Each Data Source List entry requires several character strings that is passed by the descriptor. To simplify the creation of these strings, function WORF_DSL_PARSE can be called by the user to break a single character string into the parts required by WORF_DSL_ADD_SYMBOL.

Real-Time DSLs

The syntax expected by WORF_DSL_PARSE is as follows.

```
[node-1[,node-2]::][db[(S)]:]symbol[(subscript)]
```

Where:

node-n	is the node name of the machine that has the data.
db	is the name of the CRISP database.
symbol	is the name of the symbol in the database.
subscript	is an integer, or symbol name in the database, that is used as a subscript value.

(S) If two nodes are specified:
(S) is present, track the standby database.
(S) is not present, track the active database.

If a single node name is listed, data from that node is taken. If two nodes are specified, tracking of active/standby databases occurs. If *(S)* follows the database name, data from the standby database is used. Otherwise, data from the active database is used.

If the optional *subscript* is a number, that is the subscript value used. If *subscript* is not a number, it is assumed to be a symbol name. The database supplying the data for *symbol* is used for the data for *subscript*.

Real-Time DSLs Examples

```
CRISP:SYSTAT_B_ACTIVE
```

The CRISP/32 system database `CRISP` on the default node, symbol `SYSTAT_B_ACTIVE`.

```
SFQ::FRED:GEORGE
```

Symbol `GEORGE` from database `FRED` on node `SFQ`.

```
SFQ,CJP::IDCTST:ABC(14)
```

Both nodes `SFQ` and `CJP` have database `IDCTST`. Use the active database, operating on the fifteenth element of array `ABC`.

HISTORIAN and SPC DSLs

The syntax expected by WORF_DSL_PARSE is as follows.

```
[node::][rootspec]pointspec
```

Where:

node is the node name of the machine that is running the CASRV to be used.

rootspec is the VMS directory at which the search for *pointspec* is anchored. *rootspec* is of the following form.

```
[node::][device_name:] "[" vms_directory_spec. "]"
```

Note that the square brackets are required for WORF_DSL_PARSE to recognize the rootspec.

pointspec is the point file name, in the following form.

```
[node::][dir_name:]point_name(subscript)
```

The value of *node* determines which machine is used for HISTORIAN and SPC functions. If no *rootspec* is defined, the value of the system logical name CRISP$HIST_ROOT at *node* is used.

If the *pointspec* contains a `dir_name`, *rootspec* must have a trailing period inside of its angle brackets. For example, a *pointspec* of `test:fl1` requires the following *rootspec*.

```
DISK$USER:[CRISP.HIST.ELAB.]
```

If the *rootspec* is specified, it is passed unchanged to the CASRV on *node*. The *rootspec* can contain a node name that causes the CASRV to retrieve the data from a remote machine by way of DECnet.

The value of `subscript` must be a number.

HISTORIAN and SPC DSLs Examples

```
HIST_MACHINE::PROC4:TEMP
```

From node HIST_MACHINE, use database PROC4, symbol TEMP. CRISP$HIST_ROOT on HIST_MACHINE is used.

```
DISK$HIST:[CRISP.HIST.]PROC2:PRESS(2)
```

Override CRISP$HIST_ROOT on the default node. Use database PROC2, symbol PRESS(2).

Trend DSLs

The syntax expected by WORF_DSL_PARSE is:

```
[node-1[,node-2]::][db:]symbol[(subscript)]
```

Where:

node-n	is the node name of the machine that has the data.
db	is the name of the CRISP database.
symbol	is the name of the symbol in the database.
subscript	is an integer that is used as a subscript value.

Trend DSLs Examples

```
PROCESS:VOLTS
```

Creates a trend of real-time symbol VOLTS from database PROCESS on the default node.

```
SFQ,CJP::TEST:MOX(4)
```

Both nodes SFQ and CJP have database TEST. Create a trend on the fifth element of symbol MOX from the active database.

Notes:

General

WORF_DSL_ADD_SYMBOL and WORF_DSL_GET_SYMBOL are the user's primary way of communicating with WORF.

The user loads a structure of type SYMBOL_RECORD and then calls WORF_DSL_ADD_SYMBOL to add the new element to a Data Source List. WORF_DSL_GET_SYMBOL copies the internal SYMBOL_RECORD of WORF into the user's SYMBOL_RECORD for examination and modification. Each of these fields are described in this section.

Node Name

The user must always set `node_name_dx` before calling WORF_DSL_ADD_SYMBOL.

A descriptor with a zero length or a NULL pointer defaults to the node that is running WORF.

Any communications necessary are conducted with that node.

The maximum length of a node name is six characters.

Beginning with V2.8-21, node name of "0" means the local node.

Status

WORF sets bits in `status` to indicate the status of a DSL element.

Status Bits	Description
WORF_DSL_M_RESOLVED	This element has been resolved successfully. Refer to Notes ‹ and › .
WORF_DSL_M_ALTERNATE	This element is the alternate of an active/standby pair. Refer to Notes ‹ and › .
WORF_DSL_M_ARRAY	This element refers to an array. Refer to Note ‹ .
WORF_DSL_M_VARSUB	This element has a variable subscript. Refer to Note ‹ .
WORF_DSL_M_INVALID	This element is not valid. Refer to `error_code`.
WORF_DSL_M_OVERRUN	The user's buffer was overrun by a read or write operation.
WORF_DSL_M_NULL	Buffer of the symbol is zero length or is a NULL pointer.
WORF_DSL_M_BOUND	The subscript value is out of bounds. Refer to Note ‹ .
WORF_DSL_M_BOTH	Both servers in an active/standby pair responded. Refer to Notes ‹ , › , and fl .
WORF_DSL_M_NOUPDATE	This entry was not updated by the last operation. Refer to Notes ‹ , › , and fl .

Status (cont)

The user can set or clear these bits in `status` using WORF_DSL_SET_STATUS and WORF_DSL_CLEAR_STATUS.

Status Bits	Description
WORF_DSL_M_INVALID	Stop all update operations on this element.
WORF_DSL_M_BOUND	The subscript value is out of bounds.
WORF_DSL_M_BOTH	Both servers responded in an active/standby pair.

Error Code

When bit WORF_DSL_M_INVALID is set, the following bits in `error_code` are meaningful.

Error Code Bits	Description
WORF_ERR_M_NONODE	The named node was not located.
WORF_ERR_M_NODB	The named database was not located. Refer to Note ‹ .
WORF_ERR_M_NOSYM	The named symbol was not located. Refer to Notes ‹ and › .
WORF_ERR_M_CLIENT	The user requested that this entry be marked invalid.
WORF_ERR_M_BADSUB	A variable subscript was specified. The type of the variable was not numerical. Refer to Note ‹ .
WORF_ERR_M_SUBNF	A variable subscript was specified, but the symbol for the variable subscript was not located. Refer to Note ‹ .
WORF_ERR_M_NOTSUB	A subscript was specified for a non-array variable. Refer to Note ‹ .
WORF_ERR_M_TIMEOUT	Timeout to the server.
WORF_ERR_M_BADTYPE	Type code returned from the server was not valid. Refer to Note ‹ .
WORF_ERR_M_ CANTPROC	Server cannot process request. Refer to Note ´ .
WORF_ERR_M_FNF	File not located. Refer to Note ´ .

Error Code (cont)

Error Code Bits	Description
WORF_ERR_M_ TNDFULL	Trend area cannot hold another trend. Refer to Note › .
WORF_ERR_M_ TNDNOTAVAIL	Trend area not available to server. Refer to Note › .
WORF_ERR_M_ NOTREND	Trend not recognized by server. This means that the trend was deleted by the trend process. The next attempt to read the trend should succeed. Refer to Note › .

Note ‹ = Applicable to real-time DSLs.
Note › = Applicable to trend DSLs.
Note ´ = Applicable to HISTORIAN and SPC DSLs.
Note fl = Before V2.8-36, these bits were sct by WORF_DSL_READ only. Later versions of WORF can set these bits in WORF_DSL_WRITE as well.

User Data

The user may enter four bytes of data in `user_data`. This information is copied from the SYMBOL_RECORD by WORF_DSL_ADD_SYMBOL and can be modified by WORF_DSL_MODIFY_SYMBOL.
For example, `user_data` may contain the address of a data structure associated with the DSL element.

Real-Time

The user must set the following elements of the SYMBOL_RECORD structure before calling WORF_DSL_ADD_SYMBOL.

`rt.alt_node_name_dx`

The name of a second node. Used for tracking active/standby status. Ignored unless the tracking option is WORF_TRACK_ACTIVE or WORF_TRACK_STANDBY.

`rt.database_name_dx`

The name of the database containing the symbol.

`rt.symbol_name_dx` The name of the symbol.

(Continued on next page.)

Real-Time (cont)

`rt.subscript_dx` The subscript to use for the symbol. A zero length or NULL pointer indicates that no subscript value is to be used.

`rt.tracking_options`

A value of how the symbol is to be tracked.

WORF_NO_TRACK - No tracking - collect data from the specified system.

WORF_TRACK_ACTIVE - Track the database that is active.

WORF_TRACK_STANDBY - Track the database that is standby.

`rt.transfer_count` The number of consecutive elements to transfer during read/write operations. A value of <= 1 causes one element to be transferred.

If the symbol is an array, `transfer_count` is checked to ensure that the transfer does not exceed the bounds of the array. If the symbol is **not** an array, `transfer_count` must be 1.

`rt.buffer_dx` Descriptor for a buffer in the client's area where the data is read or written. The type and class of the descriptor are ignored. The site of the buffer must be at least `rt.transfer_count * rt.data_length` bytes, or the WORF_DSL_M_OVERRUN bit will be set in `status`.

The following are set by WORF_DSL_RESOLVE.

`rt.subscript` The subscript value or the record number of the subscript symbol, depending on bit WORF_DSL_M_VARSUB in `status`.

`rt.symbol_type` The symbol type code, as defined in CSP$VARTYPEDEF_USER_x.

`rt.dimension` The number of elements in the array. If the symbol is not an array, the dimension is zero.

Real-Time (cont)

`rt.data_length` The minimum size of data buffer required to one element of the data, in bytes. For strings, this is 2 bytes larger than the maximum string size.

`rt.record_number` The record number of the symbol.

To modify the values of `rt.subscript` and `rt.transfer_count` once a symbol has been added to a Data Source List, call WORF_DSL_MODIFY_SYMBOL.

NOTE

If a variable subscript goes beyond the bounds of an array, the user will not be notified. If the user has to monitor the value of the subscript, add an entry to the Data Source List that returns the subscript value actually used.

To avoid unnecessary resolving, load the following fields of the symbol with the proper values. If both the `rt.record_number` and `rt.symbol_type` fields are non-zero, it is assumed that all the fields have been loaded properly and that resolving of the symbol is unnecessary. 'Trash' values in the fields may adversely affect the operation of WORF.

`status` (bits WORF_DSL_M_ARRAY and WORF_DSL_M_VARSUB)
`rt.subscript`
`rt.symbol_type`
`rt.dimension`
`rt.data_length`
`rt.record_number`

Data is transferred under the following values of `rt.tracking_options`.

- WORF_NO_TRACK - No tracking -- Transfer data for entry always.
- WORF_TRACK_ACTIVE -- Track the database that is active.
- WORF_TRACK_STANDBY -- Track the database that is standby.

HISTORIAN and SPC

When setting up an HISTORIAN or SPC Data Source List, the user must specify the type of operation to be performed and various related values. Which values are required depends on the type of operation to be performed. Some fields are used as both inputs and outputs.

Required variables are as follows.

`hs.operation_type` The operation to be performed. Valid operation types are as follows (refer to header file WORF_DEF_USER_x).

HS_DATA_AT_TIMES
HS_DATA_BY_RECORD
HS_DATA_BY_TIME
HS_DATA_MATCH
HS_EVENT_DRIVEN
HS_GET_FIRST_POINT
HS_GET_LAST_POINT
HS_SKIP_RECORDS
HS_TRANSLATE_ROOT
HS_UPDATE_AT_TIMES

`hs.point_name_dx` The point file name defines which file is to be operated upon. The point file name has the format: [dir_name:]point_name

`hs.root_name_dx` A VMS path has begun the search for `hs.point_name_dx`. If the length of this descriptor is zero, the value of logical name CRISP$HIST_ROOT at the server (if any) is used.

Other variables that might have to be set are as follows.

`hs.point_count` The number of points to read or write. Defines the sizes of arrays. The maximum value is defined in HS_MAX_POINTS (refer to WORF_DEF_USER_x). Not used for DATA_MATCH.

HISTORIAN and SPC (cont)

`hs.time_array_dx` Descriptor for a buffer in the client's area where the HISTORIAN time data is read or written. The type and class of the descriptor are ignored.

The client loads this buffer with HISTORIAN time values, sorted in ascending time values. (DATA_AT_TIMES, UPDATE_AT_TIMES)

WORF loads this buffer with HISTORIAN time values read. (DATA_BY_RECORD, DATA_BY_TIME, DATA_MATCH, EVENT_DRIVEN, GET_FIRST_POINT, GET_LAST_POINT, SKIP_RECORDS)

`hs.start_time` The HISTORIAN time beginning at which records are retrieved. (DATA_BY_RECORD, DATA_BY_TIME, SKIP_RECORDS)

The HISTORIAN time at which searching is to begin. (DATA_MATCH, EVENT_DRIVEN)

`hs.time_inverval` Time interval, in seconds, between desired data values. (DATA_BY_TIME)

`hs.compression_factor`

Compression factor to use (DATA_BY_TIME). Values are as follows.

0 No expansion or compression.

x > 0 For each output point, x + 1 file points are read and averaged.

x < 0 Each file point is expanded into 1 - x output points.

`hs.record_count` Number of records to skip (SKIP_RECORDS).

< 0 Move backwards in time.

= 0 Do not move; retrieve information at specified time.

> 0 Move forward in time.

HISTORIAN and SPC (cont)

`hs.interpolation_option`

Interpolation option code to use (DATA_BY_TIME). Valid values are as follows.

0 Sample the file with interpolation between points.

1 Sample the file, but return the lower value when between points.

`hs.search_criteria`

Search code to use (DATA_MATCH, EVENT_DRIVEN). The following are valid values (refer to header file WORF_DEF_USER_x).

HS_LESS_THAN To match, the record in the point file must be less than `hs.search_value`.

HS_LESS_THAN_OR_EQUAL

To match, the record in the point file must be less than or equal to `hs.search_value`.

HS_GREATER_THAN To match, the record in the point file must be greater than `hs.search_value`.

HS_GREATER_THAN_OR_EQUAL

To match, the record in the point file must be greater than or equal to `hs.search_value`.

HS_EQUAL_TO To match, the record in the point file must be equal to `hs.search_value`.

HS_NOT_EQUAL_TO To match, the record in the point file must not be equal to `hs.search_value`.

If searching is not desired for the EVENT_DRIVEN operation, use a `hs.search_criteria` of zero.

HISTORIAN and SPC (cont)

`hs.search_value`

Value to search for. (DATA_MATCH, EVENT_DRIVEN)

`hs.end_time` The HISTORIAN time at which searching is to end. (DATA_MATCH, EVENT_DRIVEN)

`hs.value_array_dx`

Descriptor for a buffer in the client's area where the data is read or written. The type and class of the descriptor are ignored.

The client loads this buffer with data values to be written. (UPDATE_AT_TIMES)

WORF loads this buffer with data values as read. (DATA_AT_TIMES, DATA_BY_RECORD, DATA_BY_TIME, DATA_MATCH, EVENT_DRIVEN, GET_FIRST_POINT, GET_LAST_POINT, SKIP_RECORD). For floating point values, a value of WORF_FLT_INVALID indicates an invalid value.

`hs.secondary_value_array_dx`

Descriptor for a buffer in the client's area where the data is read or written. The type and class of the descriptor are ignored.

WORF loads this buffer with data values as read. (EVENT_DRIVEN). For floating point values, a value of WORF_FLT_INVALID indicates an invalid value.

`hs.second_point_dx`

The secondary point file name to be operated upon. The name has the following format: `[dir_name:]point_name`. (EVENT_DRIVEN)

`hs.actual_root_dx`

Descriptor for a buffer in the client's area where the historian root actually used by the server is stored. The root name is loaded into the buffer, and the descriptor length is modified. If the buffer points to NULL, or has a length of zero, the root name is not written. The type and class of the descriptor are ignored.

HISTORIAN and SPC (cont)

The user may examine the following by calling WORF_DSL_GET_SYMBOL.

`hs.points_processed`

The number of points processed or returned.

`hs.match_time` The HISTORIAN time at which a match was located. (DATA_MATCH, EVENT_DRIVEN) This value is copied into the `hs.time_array_dx` buffer (DATA_MATCH).

`hs.match_value` The point file value that matched. (DATA_MATCH, EVENT_DRIVEN). For floating point values, a value of WORF_FLT_INVALID indicates an invalid value. This value is copied into the `hs.value_array_dx` buffer (DATA_MATCH).

`hs.search_status` A VMS status code relating to the operation.

`hs.secondary_status`

A VMS status code relating to EVENT_DRIVEN operations only.

`hs.secondary_data_type`

When the data is returned from the operation, the data type is indicated. This field will be HS_FLOAT, HS_LONG, or HS_UNSIGNED_LONG (refer to header file WORF_DEF_USER_x).

`hs.queue_position` Before the CRISP Access Server begins processing the operation, this number will hold the position that the request holds relative to all other pending requests on that node.

`hs.data_type` When the data is returned from the operation, the data type will be indicated. This field will be HS_FLOAT, HS_LONG, or HS_UNSIGNED_LONG (refer to header file WORF_DEF_USER_x).

HISTORIAN and SPC (cont)

`hs.search_status` and `hs.secondary_status` may have one of the following values:

`HSUB_CRITERION`	Criteria passed to DATA_MATCH not valid (DATA_MATCH, EVENT_DRIVEN).
`HSUB_ENDOFDATA`	End of last file encountered in processing point file records (DATA_AT_TIMES, DATA_BY_RECORD, DATA_MATCH, EVENT_DRIVEN, UPDATE_AT_TIMES).
`HSUB_NOMATCH`	No record satisfying match conditions located in point file (DATA_MATCH, EVENT_DRIVEN).
`HSUB_RECMSSNG`	Record for specified time is missing from point file (DATA_AT_TIMES, SKIP_RECORDS, UPDATE_AT_TIMES).
`HSUB_TOOEARLY`	Starting time is earlier than first recorded history point. (DATA_BY_RECORD, DATA_BY_TIME, SKIP_RECORDS)
`HSUB_TOOLATE`	Starting time is later than last recorded history point (DATA_BY_RECORD, DATA_BY_TIME, DATA_MATCH, EVENT_DRIVEN, SKIP_RECORDS).
`RMS$_FNF`	File not located.
`SS$_NORMAL`	Normal status.
`WORF_CANTPROC`	Refer to the Status Values appendix.
`WORF_NOTQUEUED`	Refer to the Status Values appendix.
`WORF_QUEUED`	Refer to the Status Values appendix.
`WORF_SERVER`	Refer to the Status Values appendix.

Trend

The user must set the following as appropriate before calling WORF_DSL_ADD_SYMBOL.

`tr.alt_node_name_dx`

The name of a second node. Used for tracking active/standby status. Ignored unless the tracking option is 'WORF_TRACK_ACTIVE'.

Trend (cont)

`tr.database_name_dx`

The name of the database containing the symbol.

`tr.symbol_name_dx` The name of the symbol.

`tr.subscript` The subscript to use for the symbol. This value is a number.

`tr.point_count` The number of points in the trend. Maximum value: TR_MAX_POINTS (refer to header file WORF_DEF_USER_x).

`tr.buffer_dx` Descriptor for a buffer in the client's area where the data is written. The type and class of the descriptor are ignored. This buffer holds an array of floating point values. A value of WORF_FLT_INVALID indicates an invalid (missing) point.

`tr.period` The time between samples, in seconds.

`tr.average` The number of consecutive samples that make one trend point.

`tr.tracking_options`

A value of how the symbol is to be tracked.

WORF_NO_TRACK - No tracking--collect data for entry always.

WORF_TRACK_ACTIVE - Track the database that is active.

NOTE
It is not possible to track a standby database.

The following are set by WORF_DSL_READ.

`tr.points_returned`

The actual number of points returned in the trend. This may be less than `tr.point_count`.

Status and Error Bits

The `status` and `error_code` bits for symbols in a DSL are set or cleared in the following routines.

Status Bit — Routine

	A_S	C_S	M_S	Read (R.T.)	Read (T.R.)	Read (H.S.)	Resolve	S_S	S_R	S_W	Write (R.T.)	Write (H.S.)	N_S
WORF_DSL_M_RESOLVED	S/C			S/C		S/C							
WORF_DSL_M_ALTERNATE	S/C												
WORF_DSL_M_ARRAY	S/C					S/C						S/C	
WORF_DSL_M_VARSUB	S/C					S/C							
WORF_DSL_M_INVALID	S/C	C	S/C	S/C	S/C	S/C	S/C	S	S/C	S/C	S/C	S/C	S/C	S/C	
WORF_DSL_M_OVERRUN	S/C		S/C	S/C	S/C	S/C			S/C	S/C	S/C	S/C	S/C	
WORF_DSL_M_NULL	S/C		S/C										S/C	
WORF_DSL_M_BOUND	S/C	C	C				S/C	S	S/C	S/C				
WORF_DSL_M_BOTH		C		S/C	S/C			S			S/C*			
WORF_DSL_M_NOUPDATE				S/C	S/C							S/C*		

Error Code Bit — Routine

	A_S	C_S	M_S	Read (R.T.)	Read (T.R.)	Read (H.S.)	Resolve	S_S	S_R	S_W	Write (R.T.)	Write (H.S.)	N_S
WORF_ERR_M_NONODE	S/C		C	C		S/C							
WORF_ERR_M_NODB	S/C		C	C		S/C							
WORF_ERR_M_NOSYM	S/C		C	S/C			S/C						
WORF_ERR_M_CLIENT	S/C	C	S/C					S					
WORF_ERR_M_BADSUB							S/C						
WORF_ERR_M_SUBNF	S/C						S/C						
WORF_ERR_M_NOTSUB	S/C						S/C						
WORF_ERR_M_TIMEOUT				S/C	S/C	S/C	S/C	S/C	S/C	S/C	S/C		
WORF_ERR_M_BADTYPE							S/C					S/C	
WORF_ERR_M_CANTPROC						S/C						S/C	
WORF_ERR_M_FNF						S/C						S/C	
WORF_ERR_M_TNDFULL					S/C								
WORF_ERR_M_TNDNOTAVAIL				S/C									
WORF_ERR_M_NOTREND					S/C								

S = Set
C = Cleared
S/C = Set or Cleared

A_S	WORF_DSL_ADD_SYMBOL
C_S	WORF_DSL_CLEAR_STATUS
M_S	WORF_DSL_MODIFY_SYMBOL
Read	WORF_DSL_READ
Resolve	WORF_DSL_RESOLVE
S_S	WORF_DSL_SET_STATUS
S_R	WORF_DSL_SYMBOL_RAMP
S_W	WORF_DSL_SYMBOL_WRITE
Write	WORF_DSL_WRITE
N_S	WORF_RT_NEXT_SYMBOL

* Beginning with V2.8-36

WORF_DSL_CREATE

To create a Data Source List, the user calls WORF_DSL_CREATE. The user passes a value indicating which type of Data Source List to create (valid options are CRISP_REAL_TIME, CRISP_TREND, and CRISP_HISTORIAN). SPC data is specified as a type of CRISP_HISTORIAN. WORF_DSL_CREATE returns a handle value corresponding to the Data Source List created.

WORF_DSL_ADD_SYMBOL

To add a single element or symbol to a Data Source List, the user calls WORF_DSL_ADD_SYMBOL. The user specifies the DSL handle and the address of an element of type SYMBOL_RECORD. The function returns a handle for the element.

WORF_DSL_GET_SYMBOL

To examine an element in a Data Source List, the user calls WORF_DSL_GET_SYMBOL. The user specifies the DSL handle, symbol handle, and the address of an element of type SYMBOL_RECORD. WORF copies the information about that element from the DSL into the specified area. When communications begins (reading, writing, or resolving), the fields are updated with information about the element. The user may examine fields depending on the DSL type.

WORF_DSL_SET_STATUS

After examining the information about an element in a Data Source List, the client may want to disable updating of that element during further communications. The user calls WORF_DSL_SET_STATUS, specifying the DSL handle, symbol handle, and a status value of WORF_DSL_M_INVALID. Other status bits may be affected, depending on the DSL type.

WORF_DSL_CLEAR_STATUS

To re-enable updating an element during further communications, the user calls WORF_DSL_CLEAR_STATUS, specifying the DSL handle, symbol handle, and a status value of WORF_DSL_M_INVALID. If there are no other reasons why the symbol is invalid, it will be made valid.

WORF_DSL_TRAVERSE

Function WORF_DSL_TRAVERSE is called to process all the elements in a Data Source List. WORF_DSL_TRAVERSE calls a user-specified function once per DSL entry. This function is useful, for example, in preparing a list of symbols with problems if a function returns WORF_CHECK.

WORF_DSL_MODIFY_SYMBOL

Various pieces of a Data Source List element can be modified by WORF_DSL_MODIFY_SYMBOL. Get the current information on the Data Source List element, modify the appropriate information, and then call WORF_DSL_MODIFY_SYMBOL. Only the pieces that are user-modifiable are changed.

WORF_DSL_DELETE

An entire Data Source List can be deleted by calling WORF_DSL_DELETE. The user must specify the DSL handle that was returned from the corresponding call to WORF_DSL_CREATE. All data structures allocated for the Data Source List are freed. Communications related to that DSL are terminated.

WORF_DSL_DELETE_SYMBOL

A single element can be removed from a Data Source List by calling WORF_DSL_DELETE_SYMBOL. The user specifies both the DSL handle and symbol handle to be deleted.

WORF_DSL_GET_STATUS_STRING & WORF_DSL_GET_STATUS_STRING_NUM

The user can fetch a string describing the status of a Data Source List element by calling WORF_DSL_GET_STATUS_STRING. This string describes the highest-priority problem affecting that element.

The user may call WORF_DSL_GET_STATUS_STRING_NUM multiple times to create a set of strings that describe the status of a symbol.

WORF_DSL_ABORT_IO

All WORF I/O functions retain control until the I/O completes. This can take a long time for HISTORIAN/SPC DSLs. The user can call WORF_DSL_ABORT_IO (from AST level) to abort any operations in progress. The WORF operation in progress terminates within a short time and returns a status of WORF_ABORTED.

WORF_DSL_GET_SYMBOL, WORF_DSL_GET_STATUS_STRING, WORF_DBL_GET_DB_LIST, & WORF_NL_GET_NODE_STATS_LIST

The user can call the following functions from normal or AST code to obtain a snapshot of the WORF state: WORF_DSL_GET_SYMBOL, WORF_DSL_GET_STATUS_STRING, WORF_DBL_GET_DB_LIST, WORF_NL_GET_NODE_STATS_LIST. The user **must not** modify any of the structures of WORF while I/O is in progress or from AST code.

Notes:

Database List

Only real-time data has a user-accessible Database List. The DBL is the mapping between the Data Source List and the nodes running CRISP. A Database List is always associated with a particular Data Source List. A Data Source List may have at most one associated Database List.

When data is to be read or written, status information in the Database List is compared with the actual status of the databases on the CPUs containing the data. If no Database List exists, the read/write operation fails with a status value indicating that resolving is required (WORF_RESOLVE). If the DBL exists but the status in the list do not match those of the databases, the operation fails with a status value of WORF_RESOLVE.

Resolving of symbols can take significant time. Unnecessary resolving can be eliminated by creating a Database List. This is especially important in a process with a human interface. By not automatically resolving, the process can display a warning or determine whether the resolve is immediately necessary.

When a new Data Source List is created and information on signatures and record numbers is unavailable, resolving is unavoidable. However, the information generated in the Database List can be stored for the next time that the Data Source List is created. The user calls WORF_DBL_GET_DB_LIST, which calls the user's action routine for each Database List entry. The action routine receives the node and database name and the database signature. The database signature is the VMS-format absolute time at which the database was last compiled. The user then stores this information.

When re-using a Data Source List, the data read by WORF_DBL_GET_DB_LIST can be restored by calling WORF_DBL_ADD_DB. The user specifies the DSL handle, the node and database name and the database signature. The function adds the information to the end of the Database List. The user calls WORF_DBL_ADD_DB once for each Data Base List entry.

Resolving

When a read or write operation returns the status code WORF_RESOLVE, indicating that resolving is required, WORF_DSL_RESOLVE should be called. The user specifies the DSL handle of the Data Source List entry upon which resolving is to commence. WORF_DSL_RESOLVE retains control of the process until resolving is finished.

The resolve operation updates various information in the Data Source List.

Resolving is never necessary for HISTORIAN/SPC and Trend DSLs.

WORF_RT_GET_DB_LIST

The user can get a list of available databases on a particular node by calling function WORF_RT_GET_DB_LIST. The function communicates with the appropriate server and calls a user's action routine once for each database.

WORF_RT_GET_SYMBOL_LIST

The user can get a list of symbols matching a specified pattern by calling WORF_RT_GET_SYMBOL_LIST. The function calls a user's action routine once for each symbol in the database that matches the pattern.

WORF_RT_GET_NODE_LIST

The user can get a list of nodes that have a particular database by calling WORF_RT_GET_NODE_LIST. The function multicasts to all Database Access Servers and calls the user's action routine once for each server having the desired database.

WORF_RT_NEXT_SYMBOL

After resolving a symbol, the user can locate adjacent symbols in the database by calling WORF_RT_NEXT_SYMBOL. The user passes the handle of the symbol to the function, and WORF transacts with the specified server to locate the next symbol. WORF builds the information into a SYMBOL_RECORD structure so that the new symbol can be easily added to a Data Source List.

WORF_RT_OPTIMIZE_DSL

The DBASRV has the capability to optimize the transmit list to reduce the CPU requirements of the server. After resolving, the user can call WORF_RT_OPTIMIZE_DSL to change the internal DSL ordering of WORF. Performance may improve, but shall be no worse than prior to calling WORF_RT_OPTIMIZE_DSL.

WORF_RT_SYMBOL_LOGGING_NAME

If logging is specified when WORF_DSL_SYMBOL_RAMP or WORF_DSL_SYMBOL_WRITE is called, the DBASRV displays a message on CRISP$TT. The 'username' displayed on these messages may be changed by calling WORF_RT_SYMBOL_LOGGING_NAME, which permits a more meaningful name to be logged.

ASTs

ASTs must be enabled for WORF to communicate over IEEE 802.3. The functions that perform communications immediately return with an error status code if ASTs are disabled but network channels are open.

Unless documented otherwise, WORF functions must not be called from the AST level. WORF cannot protect the user against these errors.

Dump File

If dump output is enabled, a complete dump of all WORF internal structures is written to a file when WORF_STOP or WORF_DUMP is called. The default file name is WORF_DUMP.DMP, in the current directory of the program running WORF. By defining logical name WORF_DUMP, the user can override the default location and name of the file. Refer to the Initialization and Configuration section, subsection Advanced Information, of this manual and Appendix G for more details.

Alias File

When WORF_INIT is called, WORF opens and reads file CRISP$CFG:WORF_ALIAS.DAT. The file should contain pairs of names. When a node name is passed to WORF, the name is translated using the table into an actual node name to be used. This allows the operation of WORF to be modified without programs having to be re-written. For example, a program could be written to use production system node names, but have them aliased to non-production system node names for debugging.

A special 'actual' node name of "0" specifies the local node.

By defining logical name WORF_ALIAS, the user can override the default location and name of the file. For example, performing the following, before starting WORF, changes where WORF looks for the alias file.

```
$DEFINE/JOB WORF_ALIAS DISK$USER:MY_ALIAS.DAT
```

A different alias file can be specified before WORF_INIT is called by defining configuration value `init2.alias_file_name_dx` (V2.8-23 and later).

Buffer Trimming

WORF allocates various internal buffers. During normal operation, WORF never releases these buffers. This improves performance of WORF; fewer memory-allocation operations means better operation. However, this can cause WORF to keep more memory than is really required.

Calling WORF_TRIM_BUFFERS relieves this problem. All WORF internal buffers are freed. The next time a buffer is required, WORF allocates one of the required size.

NOTE

If WORF has to allocate buffers, this will degrade performance. It is important that the user choose a proper place to call WORF_TRIM_BUFFERS. For example, it might be appropriate to call the function after resolving all Data Source Lists.

Node List

Each Data Source List entry contains a node name. For each different node, WORF keeps status information about that node in its internal Node List. Users may want to examine the Node List to display those statistics.

To examine the Node List, call WORF_NL_GET_NODE_STATS_LIST, reading out one Node List entry at a time. The function returns the node name and the statistics.

DSL Size

There is no set limit to the size of a Data Source List. Very large real-time DSLs can cause the Database Access Server to consume excessive amounts of CPU time. This CPU loading may cause unacceptable performance for other processes.

Although WORF 'hides' communication paths from the user, the Software Bus path has strict limitations on message size. WORF_DSL_WRITE on a large real-time DSL may return a status of WORF_LIST2BIG. This indicates that the message sent to the DBASRV exceeds the largest message it can accept. The user can reduce the size of the DSL. The system manager can perform the following to increase the size of the message that the DBASRV can accept.

- Create a file called CRISP$CFG:USER_SETUP_DBASRV.COM.
- Enter the following line:

```
$ DEFINE /SYSTEM CRISP$DBASRV_SWB_SIZE  "nn"
```

  where *"nn"* is the number of pages that the server needs to connect with.

DSL Size (cont)

- `CRSTOP`
- `@CRISP$:CRISP_SETUP`
- `CRSTART`

A similar problem can occur with WORF_DSL_READ. In this case, WORF resolves symbols properly, but always times out on the read. The DBASRV signals a DBASRV-E-XMITSWB message with a second line of SWB-E-NOPOOL.

WORF_SWB_SIZE can be called to determine the number of pages needed for the largest DSL. If the problem is with WORF_DSL_READ, `init.totalsize` can be modified and WORF_SWB_RECONNECT be called.

Notes:

Compiling

The WORF development kit contains a text library (CRISPUSERLIB.TLB).

When compiling a program using WORF, the Fortran and C compilers require access to CRISPUSERLIB.TLB. Define the appropriate logical name as follows.

Non-CRISP/32 Systems

```
$ DEFINE/JOB C$LIBRARY SYS$LIBRARY:CRISPUSERLIB.TLB                    (C)
$ DEFINE/JOB FORT$LIBRARY SYS$LIBRARY:CRISPUSERLIB.TLB                 (Fortran)
```

Systems with CRISP/32

```
$ DEFINE/JOB C$LIBRARY CRISP$LIB:CRISPUSERLIB.TLB                      (C)
$ DEFINE/JOB FORT$LIBRARY CRISP$LIB:CRISPUSERLIB.TLB                   (Fortran)
```

Linking

The WORF development kit contains two run-time libraries (CRISPRTL.EXE and CRISPWORFRTL.EXE). The program must be linked against both CRISPRTL and CRISPWORFRTL, as well as the VAX CRTL (even for Fortran programs). A typical link statement is as follows.

```
$ LINK program.obj, SYS$INPUT/OPTIONS
                    SYS$LIBRARY:CRISPWORFRTL.EXE /SHAREABLE
                    SYS$LIBRARY:CRISPRTL.EXE /SHAREABLE
                    SYS$LIBRARY:VAXCRTL.EXE /SHAREABLE
```

Note that three lines ending in SHAREABLE are not continuations of the LINK statement, but are option file entries. If this command is executed interactively, CTRL-Z must be pressed to end the options input. Alternatively, an options text file can be created with these three lines.

Text Library

The text library (CRISPUSERLIB.TLB) includes the following files.

CSP$TIMEDEF_USER_C CSP$TIMEDEF_USER_FOR	Definitions of structures for VMS quadword time values.
CSP$VARTYPEDEF_USER_C CSP$VARTYPEDEF_USER_FOR	Definitions of the values for real-time symbol types.
DBA$DBDEF_USER_C DBA$DBDEF_USER_FOR	Definitions of the structures of real-time data values.
DBA$DEF_USER_C DBA$DEF_USER_FOR	Definition of DBNAME_MAXLENGTH, the largest real-time database name.
DESCRIPDEF_USER_C DESCRIPDEF_USER_FOR	Definitions of the various descriptor structures, and macros for initializing them.
MACRODEF_USER	Define C macros SUCCESS and FAILURE. Used for checking VMS status values.
SWB$DEF_USER_C SWB$DEF_USER_FOR	Software Bus definitions.
WORF_DEF_USER_C WORF_DEF_USER_FOR	WORF-specific structures and definitions.

For details, use the LIBRARY/EXTRACT command to retrieve the desired module.

Sample Programs

Sample programs are included in the WORF development kit. On CRISP/32 systems, they are installed in CRISP$HLP. On non-CRISP/32 systems, they are in installed in SYS$EXAMPLES.

The programs available are as follows.

WORF_EXAMPLE_C.C	Real-time sample program, written in C
WORF_EXAMPLE_FOR.FOR	Real-time sample program, written in Fortran
WORF_EXAMPLE_HS_C.C	HISTORIAN/SPC sample program, written in C
WORF_EXAMPLE_HS_FOR.FOR	HISTORIAN/SPC sample program, written in Fortran
WORF_EXAMPLE_TR_C.C	Trend sample program, written in C
WORF_EXAMPLE_TR_FOR.FOR	Trend sample program, written in Fortran

General

This section defines the WORF functions. Each subsection contains a description of the function, the format of the function, operational details, returns, and, where applicable, an example of the function.

This section defines the following functions.

Function	Description
WORF_CHECK_CRISP_RUNNING *(page 61)*	Determines if CRISP/32 is running on the local system.
WORF_CHECK_FOR_EXIT *(page 63)*	Determines if an exit message has been received.
WORF_CONFIG *(page 65)*	Configures WORF communications.
WORF_CONFIG_TIMEOUT *(page 67)*	Changes timeout and retry values.
WORF_DBL_ADD_DB *(page 69)*	Adds a database to the Database List.
WORF_DBL_GET_DB_LIST *(page 73)*	Reads all the Database List entries associated with a Data Source List.
WORF_DSL_ABORT_IO *(page 79)*	Aborts a pending WORF I/O operation.
WORF_DSL_ADD_SYMBOL *(page 81)*	Adds a symbol to a Data Source List.
WORF_DSL_CLEAR_STATUS *(page 85)*	Clears the status bits of a variable in a Data Source List.
WORF_DSL_CREATE *(page 87)*	Creates a Data Source List.
WORF_DSL_DELETE *(page 89)*	Deletes a Data Source List.
WORF_DSL_DELETE_SYMBOL *(page 91)*	Deletes a symbol from a Data Source List.

Function	Description
WORF_DSL_GET_NODE_AND_DB *(page 93)*	Returns information related to a symbol in a Data Source List.
WORF_DSL_GET_PAIR_HANDLE *(page 97)*	Returns the other handle of a Data Source List entry pair.
WORF_DSL_GET_STATUS_STRING *(page 99)*	Returns a text string describing a problem with the specified symbol.
WORF_DSL_GET_STATUS_STRING_NUM *(page 103)*	Returns a text string describing a problem with a symbol.
WORF_DSL_GET_SYMBOL *(page 107)*	Returns information about a symbol in a Data Source List.
WORF_DSL_MODIFY_SYMBOL *(page 111)*	Modifies a Data Source List entry.
WORF_DSL_PARSE *(page 115)*	Parses a symbol specification and fills in the appropriate pieces in a Data Source List entry.
WORF_DSL_READ *(page 119)*	Reads data for all the symbols in a Data Source List.
WORF_DSL_RESOLVE *(page 121)*	Resolves all symbols in a Data Source List.
WORF_DSL_SET_STATUS *(page 123)*	Sets the status bits of a variable in a Data Source List.
WORF_DSL_SYMBOL_RAMP *(page 124)*	Updates a single symbol in a real-time CRISP/32 database.
WORF_DSL_SYMBOL_WRITE *(page 129)*	Updates a single symbol in a real-time CRISP/32 database.

Function	Description
WORF_DSL_TRAVERSE *(page 133)*	Traverses all the entries in a Data Source List.
WORF_DSL_WRITE *(page 137)*	Writes data for all the symbols in a Data Source List.
WORF_DUMP *(page 141)*	Dumps all WORF internal structures.
WORF_HS_GET _FILE_LIST *(page 143)*	Gets a list of available Historian point files on any node.
WORF_INIT *(page 151)*	Initializes the WORF communications system.
WORF_NETWORK_RECONNECT *(page 153)*	Closes then re-opens network channels.
WORF_NL_GET_NODE_STATS_LIST *(page 155)*	Reads the statistics from the Node List.
WORF_PATH_SELECT *(page 159)*	Enables or disables the use of various communications paths by WORF.
WORF_RT_GET_DB_LIST *(page 163)*	Gets a list of available databases on a node.
WORF_RT_GET_NODE_LIST *(page 167)*	Gets a list of nodes that have the specified real-time database.
WORF_RT_GET_SYMBOL_LIST *(page 171)*	Gets a list of available symbols in a real-time database.
WORF_RT_NEXT_SYMBOL *(page 177)*	Locates adjacent symbol names from a CRISP real-time database.
WORF_RT_OPTIMIZE_DSL *(page 183)*	Optimizes a Data Source List.

Function	Description
WORF_RT_SYMBOL_LOGGING_NAME *(page 185)*	Enables the user to specify the name with which the Database Access Server logs a WORF_DSL_SYMBOL_RAMP or WORF_DSL_SYMBOL_WRITE.
WORF_STOP *(page 187)*	Stops all WORF communication activities.
WORF_SWB_RECONNECT *(page 189)*	Connects or reconnects to the Software Bus.
WORF_SWB_SIZE *(page 193)*	Computes the longest Software Bus message required.
WORF_TIMEOUT_RECOVERY *(page 197)*	Attempts to recover from node timeouts.
WORF_TIMEOUT_RECOVERY_CHECK *(page 199)*	Checks to see if timeout recovery is in progress.
WORF_TIME_HS_TO_VMS *(page 201)*	Converts Historian time to a VMS quadword time.
WORF_TIME_VMS_TO_HS *(page 203)*	Converts a VMS quadword time to a Historian time.
WORF_TRANSLATE_ALIAS *(page 205)*	Translates a WORF alias node name into an actual node name.
WORF_TRIM_BUFFERS *(page 207)*	Trims all WORF internal buffers.
WORF_VERSION_NUMBER *(page 209)*	Returns the version number of WORF.

WORF_CHECK_CRISP_RUNNING

Description

Sees if CRISP/32 is running on the local system.

Format

WORF_CHECK_CRISP_RUNNING ()

Operation

Checks to determine if CRISP/32 is running on the local system.

This function may be called at any time, including before WORF_INIT and WORF_CONFIG.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion (CRISP is running).

Warning:

WORF_CRSTOP	CRISP is stopped.

Error:

CSP_NOTIMPL	Image linked against WORF RTL is running against an incompatible version of WORF RTL.

Examples

```
#include worf_def_user_c

globalvalue const int SS$_NORMAL;

WORF_CONFIG *config_ptr;

long systat;

/* get WORF configuration information */
config_ptr = worf_config (0);

/* connect to Software Bus only if CRISP is running */
systat = worf_check_crisp_running ();
if (systat != SS$_NORMAL)
    config_ptr->init.num_queues = 0;

systat = worf_init (0);
```

WORF_CHECK_FOR_EXIT

Description

Determines if an exit message has been received.

Format

WORF_CHECK_FOR_EXIT ()

Operation

Processes connected to the Software Bus should exit when an exit message is received. To insulate the user of WORF from knowing about the Software Bus, this function checks for an exit message.

If an exit message has been received, the function returns a status value of WORF_EXITMSG.

If WORF is not connected to the Software Bus, no exit message has been received, or the user has disabled use of the Software Bus by WORF, the function returns a status of SS$_NORMAL.

NOTE

Removes all messages on Software Bus queue 1 that are not exit messages (V2.7 WORF only).

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Exit message was not received.

Informational:

WORF_EXITMSG	Exit message received.

Error:

WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include worf_def_user_c

globalvalue const int SS$_NORMAL;

long systat;

/* exit program unless no exit message has been received */
do
{
    systat = worf_check_for_exit ();
    if (systat == WORF_INPROGRESS)
    {
        lib$wait (&1.0);
        continue;
    }
    else if (systat != SS$_NORMAL)
        exit (systat);
}
while (systat != SS$_NORMAL);
```

WORF_CONFIG

Description

Configures WORF communications.

Format

WORF_CONFIG (*options*)

Arguments

options

Usage: Options bit mask.
Type: Unsigned longword.
Access: Read Only.
Mechanism: By value.

Operation

Allocates a table that holds parameters related to communications used by WORF. Fills in default values.

This function is optional, and does not have to be called except by users planning to manipulate the information that WORF_CONFIG builds. The function returns the address of the configuration information.

None of the options bits are defined.

Calling this function repeatedly causes no ill effects. The address of the configuration information already established is returned.

NOTE

If WORF cannot be configured, a NULL address is returned.

Returns

Usage: Parameter Table.
Type: Structure WORF_CONFIG (refer to WORF_DEF_USER_x (where, x is 'C' or 'FOR')).
Access: Write Only.
Mechanism: By reference.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c

WORF_CONFIG *config_ptr;

config_ptr = worf_config (0);
if (config_ptr == NULL)
    exit (WORF_NOCONFIG);                    /* unable to configure WORF */

/* enable automatic recovery from timeouts */
config_ptr->dynamic.recovery_time = 10;      /* every 10 seconds */
```

WORF_CONFIG_TIMEOUT

Description

Changes timeout and retry values.

Format

WORF_CONFIG_TIMEOUT (*options*)

Arguments

options

Usage: Options bit mask.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

Options to use. No options are currently defined.

Operation

The user may modify the values of `init.timeout` and `init.retries` and then call this routine to put the new values into effect.

This routine MUST NOT be called while at AST level. No error message can be returned to the user in this case, however the routine will not function properly.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

`SS$_NORMAL`	Normal successful completion.

Error:

`WORF_INPROGRESS`	WORF I/O in progress
`WORF_NOINIT`	WORF layer was not initialized.

Example

```
#include worf_def_user_c

main ()
{
    int retstat;

    WORF_CONFIG *config_ptr;

    config_ptr = worf_config (0);

    config_ptr->init.timeout = 1500;            /* 1.5 second timeout */
    config_ptr->init.retries = 5;               /* 5 retries */

    retstat = worf_init (0);

    /* change communications setup */
    config_ptr->init.timeout = 1000;            /* 1.0 second timeout */
    config_ptr->init.retries = 3;               /* 3 retries */

    retstat = worf_config_timeout (0);
}
```

WORF_DBL_ADD_DB

Description

Adds a database to the Database List.

Format

WORF_DBL_ADD_DB (*id*, *node*, *dbname*, *signature*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List with which the Database List is to be associated.

node

Usage: Node Name.
Type: Character String.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the DECnet node.

dbname

Usage: Database Name.
Type: Character String.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP database.

signature

Usage: Database signature.
Type: Quadword.
Access: Read only.
Mechanism: By reference.

The expected database signature.

Operation

To eliminate unnecessary resolving, the client can specify the databases that are used by the Data Source List.

During a call to WORF_DSL_READ, WORF_DSL_WRITE, WORF_DSL_SYMBOL_RAMP, or WORF_DSL_SYMBOL_WRITE, the actual database signature is compared with the expected signature. If the two do not match, the returned status code indicates that resolving is required.

Operation (cont)

Calling this function for a Data Source List that is not CRISP_REAL_TIME type is an error.

Calling WORF_DBL_ADD_DB more than once for the same node and database is not an error. It just changes WORF's record of the expected signature.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_DBLEXISTS	Data Base List entry already exists. Signature updated.

Error:

WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_MAXDSLS	Too many Data Source Lists referenced one node (maximum of 65535 per node).
WORF_NOINIT	WORF layer was not initialized
WORF_TRUNC	The length of *node* or *dbname* was not valid
WORF_WRONGTYPE	DBL operations not permitted on this Data Source List type.

Example

```
#include worf_def_user_c
#include descripdef_user_c

static const $DESCRIPTOR (node_dx, "PRC12");       /* node name */
static const $DESCRIPTOR (db_dx, "PROC_DB");       /* database name */
long signature[2] = {0x153233, 0x19823};           /* expected signature */

long systat;
long dsl_id;

/* create a Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* add a data base to the Data Base List */
systat = worf_dbl_add_db (dsl_id, &node_dx, &db_dx, &signature[0]);
```

Notes:

WORF_DBL_GET_DB_LIST

Description

Reads all the Database List entries associated with a Data Source List.

Format

WORF_DBL_GET_DB_LIST (*id*, *argument*, *action*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List.

argument

Usage: Argument to *action* routine
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the *action* routine.

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By reference.

The *action* routine is called once for each entry in the Database List.

Operation

To generate the list of data bases and signatures required to prevent unnecessary resolving in the future, the client can read out the Database List.

The signatures in the Database List are updated by WORF_DSL_RESOLVE. Entries can be added or modified by WORF_DBL_ADD_DB.

No guarantee is made as to the order in which Database List entries will be passed to the user's action routine.

WORF_DBL_GET_DB_LIST checks the returned value from the user's action routine. If the value is non-zero, WORF_DBL_GET_DB_LIST stops processing the Database List, and returns the value to the caller.

A common use of *argument* is to pass an address to the *action* routine. In this case, the *action* routine receives the address of a longword that contains the desired address.

Operation (cont)

The *action* routine may call any WORF function other than WORF_DSL_DELETE, WORF_DSL_ADD_DB, or WORF_STOP.

AST re-entrant: This routine can be called from either AST level or normal user mode.

NOTE

Earlier versions of WORF returned WORF_WRONGTYPE if this function was used on other than a real-time DSL. The current version of this function works with all DSL types, and adds the new argument *stats* when calling the *action* routine.

Action Routine

For each Database List entry, the *action* routine is called with the following arguments. After all DBL entries have been processed, the *action* routine is called once again, with NULL pointers for the *node* and *dbname* descriptors. Any changes made by the user to *ID*, *argument*, *number*, *node*, *dbname*, *signature*, and *stats* are ignored.

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By reference.

The handle value of the Data Source List.

argument

Usage: Argument to action routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The argument that was passed to WORF_DBL_GET_DB_LIST.

number

Usage: Item number of symbol.
Type: Longword.
Access: Read only.
Mechanism: By reference.

Each successive call of the user's action routine is passed a number from 1 to the number of DBL entries. A value of 0 signals that the previous call to the action routine was the last DBL entry located.

Action Routine (cont)

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the node from the Database List entry. For convenience of C programs, it is also null-terminated.

dbname

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the database from the Database List entry. For convenience of C programs, it is also null-termined.

signature

Usage: Database signature.
Type: Quadword.
Access: Read only.
Mechanism: By reference.

The database signature (Real-time DSLs only).

stats

Usage: Statistics for the database
Type: Structure of type DATABASE_STATISTICS
Access: Read only
Mechanism: By reference

The statistics for the database, including the following fields.

status A bit mask of status information, as defined in the following.

WORF_DB_M_LOCATED	The database has been located.
WORF_DB_M_ACTIVE	The database is active.

server_number The database number at the server (Real-time DSLs only).

Action Routine (cont)

server_status A bit mask of database status as reported by the server (Real-time DSLs only).

WORF_SERVER_DB_M_ACTIVE	The database is active.
WORF_SERVER_DB_M_STANDBY	The database is standby.
WORF_SERVER_DB_M_SYSTEM	The database is a system database.
WORF_SERVER_DB_M_LOCAL	The database is a local database.
WORF_SERVER_DB_M_PUBLIC	The database is a public database.

signature The database signature (Real-time DSLs only).

Returns

Usage: Condition code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Any non-zero status from the user's *action* routine.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized.

Example

DISPLAY ALL DATABASE LIST ENTRIES

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;

printf ("\n\nData Base List:");
systat = worf_dbl_get_db_list (dsl_id, 0, &dbl_action);
if (FAILURE (systat))
    lib$signal (systat);



static long dbl_action (long *dsl_id_ptr, long *argument_ptr,
    long *number_ptr, DSC$DESCRIPTOR *node_dx_ptr,
    DSC$DESCRIPTOR *dbname_dx_ptr, long *sig_ptr,
    DATABASE_STATISTICS *stats_ptr)
{
    if (*number_ptr != 0)
    {
        printf ("\n%3d %s::%s %08X%08X", *number_ptr,
            node_dx_ptr->dsc$a_pointer, dbname_dx_ptr->dsc$a_pointer,
            sig_ptr[1], sig_ptr[0]);
    }

    return 0;                /* keep going */
}
```

Notes:

WORF_DSL_ABORT_IO

Description

Aborts a pending WORF I/O operation.

Format

WORF_DSL_ABORT_IO ()

Operation

WORF operations, especially those dealing with Historian or SPC data, can take a large amount of time to complete. Historian data retrieval may take minutes, which means that the user calls WORF_DSL_READ or WORF_DSL_WRITE and does not receive control back for minutes.

The user can establish a cover timer and call WORF_DSL_ABORT_IO at AST level. This causes I/O to cease almost immediately (a wait comparable to the timeout interval may be expected).

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_NOINIT	WORF layer was not initialized.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c

static const $DESCRIPTOR (time_dx, "0 00:01:00");

long time_interval[2];
long systat;


/* convert time string into VMS internal format */
sys$bintim (&time_dx, &time_interval[0]);

/* start the timer -- do AST to "cancel_ast" when it expires */
sys$setimr (0, &time_interval[0], &cancel_ast, (unsigned long)&time_interval[0], 0);

/* do a read -- if not complete in 1 minute, it will be cancelled */
systat = worf_dsl_read (dsl_id);
if (systat == WORF_ABORTED)
    printf ("I/O was aborted\n");

/* cancel the timer */
sys$cantim ((unsigned long)&time_interval[0], 0);


/* function to cancel pending i/o */
void cancel_ast ()
{
    worf_dsl_abort ();
}
```

WORF_DSL_ADD_SYMBOL

Description

Adds a symbol to a Data Source List.

Format

WORF_DSL_ADD_SYMBOL (*id*, *symbol*, *handle*, *alt_recnum*, *alt_handle*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

symbol

Usage: Information about symbol.
Type: Structure SYMBOL_RECORD.
Access: Read only.
Mechanism: By reference.

The address of a structure of type SYMBOL_RECORD. Refer to header file WORF_DEF_USER_x (where, x is 'C' or 'FOR') for more information.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The handle value of the symbol that was added to the Data Source List.

alt_recnum

Usage: Record number.
Type: Unsigned word.
Access: Read only.
Mechanism: By value.

The record number of the symbol on the alternate node (used only for real-time active/standby tracking).

alt_handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The handle value of the symbol that was added to the Data Source List. (Used only for real-time and trend active/standby tracking.)

Operation

Adds a symbol to the specified Data Source List.

If the value of *status* in the input structure has the invalid bit set (WORF_DSL_M_INVALID), the resulting symbol is set as invalid, as requested by the client.

The handle value that is returned is used to manipulate the information about the symbol.

To easily load the fields of *symbol*, refer to function WORF_DSL_PARSE.

To remove a symbol from a Data Source List, call function WORF_DSL_DELETE_SYMBOL.

If the SYMBOL_RECORD specifies active/standby tracking (the alternate node name is not zero length), two DSL elements are added to the Data Source List. The second DSL element is a copy of the first, with the node name and alternate node name swapped.

The *alt_recnum* parameter is required if resolving is to be avoided for real-time DSLs, since the record number on the alternate node may not be the same as the first node.

A *symbol.node_name_dx* descriptor with a zero length or a NULL pointer defaults to the node on which WORF is running.

For real-time DSLs, it is highly recommended that `rt.transfer_count` be set to 1 until the Data Source List is resolved.

A status of WORF_INVALID is returned if a string descriptor points to a location that is not readable.

Violating limits on the following shall cause a return status of WORF_INVALID.

- The node name is longer than 6 characters.
- For real-time DSLs: The size of the database name being greater than DBNAME_MAXLENGTH (defined in DBA$DEF_USER_x (where, x is 'C' or 'FOR')).

  The size of the symbol name or subscript greater than SYMNAME_MAXLENGTH (defined in DBA$DBDEF_USER_x (where, x is 'C' or 'FOR')).

  The tracking options are WORF_NO_TRACK, WORF_TRACK_ACTIVE, or WORF_TRACK_STANDBY.

Operation (cont)

- For Historian/SPC DSLs: The value of `symbol.hs.operation_type` is not valid.

  The value of `hs.point_count` is greater than HS_MAX_POINTS.

  The length of strings `hs.root_name_dx`, `hs.point_name_dx`, or `hs.second_point_dx` is greater than 256 characters.

- For trend DSLs: The size of the database name being greater than DBNAME_MAXLENGTH (defined in DBA$DEF_USER_x (where, x is 'C' or 'FOR')).

  The size of the symbol name being greater than SYMNAME_MAXLENGTH (defined in DBA$DBDEF_USER_x (where, x is 'C' or 'FOR')).

  The value of `symbol.tr.point_count` is greater than TR_MAX_POINTS.

  The tracking options are WORF_NO_TRACK or WORF_TRACK_ACTIVE.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Error codes from memory allocation routines, plus the following condition values.

Success:

`SS$_NORMAL`	Normal successful completion.

Error:

`WORF_BADID`	Data Source List ID was not valid
`WORF_INPROGRESS`	WORF I/O in progress
`WORF_MAXDSLS`	Too many Data Source Lists referenced one noc (maximum of 65535 per node)
`WORF_INVALID`	Information in the entry was not valid
`WORF_NOINIT`	WORF layer was not initialized
`WORF_NULLPTR`	A NULL pointer was passed for *symbol*
`WORF_MISSING`	Missing database or symbol name.

Fatal:

`WORF_UNIMPL`	Unimplemented code path.

Example

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle;

SYMBOL_RECORD record;

static const $DESCRIPTOR (symbol_dx, "PROC12::PROC_DB:INPUT_3");


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* parse the symbol name, and clear fields in "record" */
systat = worf_dsl_parse (dsl_id, &symbol_dx, &record);

/* add the symbol */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);
if (FAILURE (systat))
    exit (systat);
```

WORF_DSL_CLEAR_STATUS

Description

Clears the status bits of a variable in a Data Source List.

Format

WORF_DSL_CLEAR_STATUS (*id*, *handle*, *status*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be modified.

status

Usage: New status value.
Type: Longword.
Access: Read only.
Mechanism: By value.

The status bits of the symbol to be clear.

Operation

Clears the status bits of a specific symbol.

The input value of *status* is searched for bits that are on (1). The corresponding bits of the status field of the symbol are then cleared. Bits in *status* that are off (0) cause no change to the status field of the symbol.

Bits (refer to the DSL Information section, subsection Status) that can be specified by the client are as follows.

WORF_DSL_M_INVALID
WORF_DSL_M_BOUND
WORF_DSL_M_BOTH

This function enables the client to disable read/write of specific symbols. If the client had previously disabled updating of a variable by calling WORF_DSL_SET_STATUS, the client could enable updating of the variable by clearing bit WORF_DSL_M_INVALID.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_IGNORED	Change to 'invalid' bit ignored. Symbol must remain invalid because of information discovered by WORF.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include macrodef_user

long systat;

/* attempt to enable updating of symbol */
systat = worf_dsl_clear_status (dsl_id, symbol_handle, WORF_DSL_M_INVALID)

if (systat == WORF_IGNORED)
    printf ("Cannot enable symbol -- problems in symbol status\n");
else if (FAILURE (systat))
    exit (systat);
else
    printf ("Symbol updating now enabled\n");
```

WORF_DSL_CREATE

Description

Creates a Data Source List.

Format

WORF_DSL_CREATE (*type*, *id*)

Arguments

type

Usage: Type of DSL.
Type: Longword.
Access: Read only.
Mechanism: By value.

The type number of Data Source List to be created.

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The handle value of the Data Source List that was created.

Operation

After calling WORF_INIT and before calling WORF_DSL_ADD_SYMBOL, a Data Source List must be created. WORF_DSL_CREATE initializes the structures internal to the WORF layer.

A client may have multiple Data Source Lists. All manipulations of DSL entries must specify the handle passed back from this routine. Valid *type* arguments are as follows.

CRISP_REAL_TIME
CRISP_HISTORIAN
CRISP_TREND

Symbols for the types of Data Source List are located in header file WORF_DEF_USER_x (where, x is 'C' or 'FOR').

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_INPROGRESS	WORF I/O in progress
WORF_INVTYPE	Type number of DSL was not valid
WORF_NOINIT	WORF layer was not initialized.

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include worf_def_user_c
#include macrodef_user

long systat;
long dsl_id[3];


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id[0]);
if (FAILURE (systat))
    exit (systat);

/* create an Historian Data Source List */
systat = worf_dsl_create (CRISP_HISTORIAN, &dsl_id[1]);
if (FAILURE (systat))
    exit (systat);

/* create a trend Data Source List */
systat = worf_dsl_create (CRISP_TREND, &dsl_id[2]);
if (FAILURE (systat))
    exit (systat);
```

WORF_DSL_DELETE

Description

Deletes a Data Source List.

Format

WORF_DSL_DELETE (*id*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be deleted.

Operation

Deletes the specified Data Source List, all symbols, and any associated Database Lists. Frees all memory for this DSL.

To add or remove individual symbols from a Data Source List, refer to WORF_DSL_ADD_SYMBOL and WORF_DSL_DELETE_SYMBOL.

⚠ CAUTION

Using the handle of a deleted DSL or symbol is an error, and may cause an access violation.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include worf_def_user_c
#include macrodef_user

int idx;

long systat;
long dsl_id[3];


/* create Data Source Lists */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id[0]);
systat = worf_dsl_create (CRISP_HISTORIAN, &dsl_id[1]);
systat = worf_dsl_create (CRISP_TREND, &dsl_id[2]);


/* delete the Data Source Lists */
for (idx = 0; idx < 3; idx++)
{
    systat = worf_dsl_delete (dsl_id[idx]);
    if (FAILURE (systat))
       exit (systat);
}
```

WORF_DSL_DELETE_SYMBOL

Description

Deletes a symbol from a Data Source List.

Format

WORF_DSL_DELETE_SYMBOL (*id*, *handle*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be deleted from the Data Source List.

Operation

Deletes a symbol from the specified Data Source List.

To add a symbol to a Data Source List, call function WORF_DSL_ADD_SYMBOL.

NOTE

For active/standby tracking pairs, deletes both DSL entries.

⚠ CAUTION

Using the handle of a deleted symbol is an error, and may cause an access violation.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Returns (cont)

Condition Values Returned Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include worf_def_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle[3];

SYMBOL_RECORD record;


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* add symbols */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[0], 0, NULL);
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[1], 0, NULL);
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[2], 0, NULL);


/* delete the second symbol */
systat = worf_dsl_delete_symbol (dsl_id, symbol_handle[1]);
if (FAILURE (systat))
    exit (systat);
```

WORF_DSL_GET_NODE_AND_DB

Description

Returns information related to a symbol in a Data Source List.

Format

WORF_DSL_GET_NODE_AND_DB (*id, handle,* [*node*], [*database*])

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List holding the desired symbol.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol in the Data Source List.

[node]

Usage: Statistics for the node.
Type: Structure of type NODE_STATISTICS.
Access: Read only.
Mechanism: By reference.

The statistics for the node, including the following fields. This argument is optional.

casrv_count The total number of operations that failed with the CRISP Access Server of the node.

dbasrv_count The total number of operations that failed with the Database Access Server of the node.

status A bit mask of status information, defined as follows.

WORF_NODE_M_LOCATED	The node has been located.
WORF_NODE_M_SWB	The node is reachable over the Software Bus.
WORF_NODE_M_IEEE802	The node is reachable over the IEEE 802.3 network.

Arguments

status (cont)

WORF_NODE_M_TMO_CASRV	The CRISP Access Server is currently timed out.
WORF_NODE_M_TMO_DBASRV	The Database Access Server is currently timed out.

[database]

Usage: Statistics for the database.
Type: Structure of type DATABASE_STATISTICS.
Access: Read only.
Mechanism: By reference.

The statistics for the database, including the following fields. This argument is optional.

status A bit mask of status information, as defined in the following.

WORF_DB_M_LOCATED	The database has been located.
WORF_DB_M_ACTIVE	The database is active.

server_number The database number at the server (Real-time DSLs only).

server_status A bit mask of database status as reported by the server (Real-time DSLs only).

WORF_SERVER_DB_M_ACTIVE	The database is active.
WORF_SERVER_DB_M_STANDBY	The database is standby.
WORF_SERVER_DB_M_SYSTEM	The database is a system database.
WORF_SERVER_DB_M_LOCAL	The database is a local database.
WORF_SERVER_DB_M_PUBLIC	The database is a public database.

signature The database signature (Real-time DSLs only).

Operation

The node and database information for the symbol that is stored in the Data Source List is copied to the specified locations.

If a *handle* of zero is specified, a symbol in the DSL is chosen by WORF's internal ordering of the DSL. If the DSL is empty, no information is returned. Specifying a *handle* of zero is only useful if all the DSL entries are from the same node and database.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_DSLEMPTY	Data Source List is empty.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized.

Example

```
    #include worf_def_user_c

    main ()
    {
        long dsl_id;
        long symbol_handle;
        long systat;

        SYMBOL_RECORD record;

        NODE_STATISTICS node;

        DATABASE_STATISTICS db;


        /* set up situation */
        systat = worf_init (0);
        systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);
        systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);
        systat = worf_dsl_resolve (dsl_id);

        /* get statistics */
        systat = worf_dsl_get_node_and_db (dsl_id, symbol_handle, &node, &db);



        if (!(node.status & WORF_NODE_M_LOCATED))
            printf ("Node was not located\n");
        if (node.status & WORF_NODE_M_TMO_CASRV)
            printf ("CRISP Access Server is timed out\n");
        if (node.status & WORF_NODE_M_TMO_DBASRV)
            printf ("Database Access Server is timed out\n");

        if (!(db.status & WORF_DB_M_LOCATED))
            printf ("Database was not located\n");
        if (db.status & WORF_DB_M_ACTIVE)
            printf ("Database was active\n");
        else
            printf ("Database was standby\n");
}
```

WORF_DSL_GET_PAIR_HANDLE

Description

Get the other handle of a Data Source List entry pair.

Format

WORF_DSL_GET_PAIR_HANDLE (*id, handle, pair*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List that contains ***handle*** and *pair*.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol.

pair

Usage: Identification handle of the other symbol.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The handle value of the DSL symbol that is paired with ***handle***. If ***handle*** is not an active/idle tracking symbol, zero is returned.

Operation

When a symbol specifies two nodes and active/standby tracking, WORF creates two DSL elements.

This function returns the handle value of the other symbol in the pair. If ***handle*** specifies a symbol that is not part of an active/standby pair, zero is returned.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Success:

SS$_NORMAL	Normal successful completion

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized
WORF_NULLPTR	A NULL pointer was passed for *pair*.

Example

Create an active/idle tracking pair, and then get the other handle.

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

static const $DESCRIPTOR (symbol_name_dx, "ABC,DEF::CRISP:CSP_S_TIME");

SYMBOL_RECORD record;
long id, handle, other_handle;


systat = worf_dsl_parse (id, &symbol_name_dx, &record):

systat = worf_dsl_add_symbol (id, &record, &handle, 0, NULL);

systat = worf_dsl_get_pair_handle (id, handle, &other_handle);

printf ("\nDSL ID=%d, Handles=%d, %d", id, handle, other_handle);
```

WORF_DSL_GET_STATUS_STRING

Description

Returns a text string describing a problem with the specified symbol.

Format

WORF_DSL_GET_STATUS_STRING (*id*, *handle*, *string*, *length*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List containing the desired symbol.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol in the Data Source List.

string

Usage: Description of problem.
Type: Character string.
Access: Modify.
Mechanism: By descriptor -- fixed-length string descriptor.

The status of the symbol.

length

Usage: Length of string.
Type: Unsigned word.
Access: Write only.
Mechanism: By reference.

The length of the status string in bytes.

Operation

This function returns a character string that describes the status of the specified symbol.

The user supplies a descriptor that points into the user's data space. The function loads the buffer ***string*** with the string, up to the maximum length set by the incoming descriptor.

Operation (cont)

Symbols can have multiple problems. This function returns a string for the most critical problem. Refer to function WORF_DSL_GET_STATUS_STRING_NUM for a list of priorities.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	This symbol did not have an error or warning status.

Warning:

WORF_CHECK	This symbol has an error or warning status.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle;

char string[512];

DSC$DESCRIPTOR string_dx;


/* initialize descriptor */
INIT_DX_PTR (&string_dx, &string[0], sizeof (string));

/* get highest priority status string for symbol */
systat = worf_dsl_get_status_string (dsl_id, symbol_handle, &string_dx,
                                     &string_dx.dsc$w_length);
if (FAILURE (systat))
    exit (systat);

/* null-terminate string */
string[string_dx.dsc$w_length] = '\0';

printf ("Symbol status:  %s\n", string);
```

Notes:

WORF_DSL_GET_STATUS_STRING_NUM

Description

Returns a text string describing a problem with a symbol.

Format

WORF_DSL_GET_STATUS_STRING_NUM (*status*, *error*, *string*, *length*)

Arguments

status

Usage: Status value from the symbol.
Type: Unsigned longword.
Access: Modify.
Mechanism: By reference.

The status value from the SYMBOL_RECORD of the symbol.

error

Usage: Error value from the symbol.
Type: Unsigned longword.
Access: Modify.
Mechanism: By reference.

The error code value from the SYMBOL_RECORD of the symbol.

string

Usage: Description of problem.
Type: Character string.
Access: Modify.
Mechanism: By descriptor -- fixed-length string descriptor.

The status of the symbol.

length

Usage: Length of string.
Type: Unsigned word.
Access: Write only.
Mechanism: By reference.

The length of the status string in bytes.

Operation

This function returns a character string that describes the status values passed to the function.

The user supplies a descriptor that points into the user's data space. The function loads the buffer ***string*** with the string, up to the maximum length set by the incoming descriptor.

Because one symbol can have multiple problems, the string reflects the highest of the following priorities.

Operation (cont)

Return status WORF_CHECK

Server timeout
Node not found
Server cannot process request ‹
File not found ‹
Trend region not available ›
No such trend ›
Trend region full ›
Bad type code from server ´
Subscript specified for non-subscripted variable ´
Variable subscript symbol was not found ´
Variable subscript symbol was not a valid type ´
Database not found › ´
Symbol not found › ´
Subscript value out of bounds ´
Both servers of an active/standby pair responded › ´
This entry was not updated › ´
Client requested symbol be marked invalid
Data overruns buffer
Buffer pointer NULL, or length is zero

Return status SS$_NORMAL

Symbol has been resolved › ´
Symbol is alternate of an active/standby tracking pair › ´
Symbol has a variable subscript ´
Symbol is an array ´
Symbol has a normal status

‹ = Historian and SPC DSLs
› = Trend DSLs
´ = Real-time DSLs

NOTE

This function turns off the bit of *status* or *error* that produced the particular message. This allows a user to call this function multiple times, and display multiple messages in priority order.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	This symbol did not have an error or warning status.

Warning:

WORF_CHECK	This symbol has an error or warning status.

Example

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;
char string[512];
unsigned long status;
unsigned long error_code;
SYMBOL_RECORD record;
DSC$DESCRIPTOR string_dx;

/* get the current information about a DSL entry previously declared */
systat = worf_dsl_get_symbol (id, handle, &record);
if (FAILURE (systat))
    exit (systat);
status = record.status;
error_code = record.error_code;

/* get each string until the status bits are exhausted */
/* test is at end of loop, since all bits=0 produces a message */
do
{   /* initialize descriptor with buffer size each time */
    INIT_DX_PTR (&string_dx, &string[0], sizeof (string));

    systat = worf_dsl_get_status_string_num (&status, &error_code,
        &string_dx, &string_dx.dsc$w_length);
    if (FAILURE (systat) && systat != WORF_CHECK)
        exit (systat);
```

Example (cont)

```
    /* null-terminate */
    string[string_dx.dsc$w_length] = '\0';

    printf ("%s\n", string);
}
while (status != 0 || error_code != 0);
```

WORF_DSL_GET_SYMBOL

Description

Returns information about a symbol in a Data Source List.

Format

WORF_DSL_GET_SYMBOL (*id*, *handle*, *symbol*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List containing the desired symbol.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol in the Data Source List.

symbol

Usage: Information about symbol.
Type: Structure SYMBOL_RECORD.
Access: Write only.
Mechanism: By reference.

The address of a structure of type SYMBOL_RECORD. Refer to header file WORF_DEF_USER_x for more information.

Operation

The current information about the symbol that is stored in the Data Source List is copied to the specified structure.

If this function is called after symbols are resolved (refer to WORF_DSL_RESOLVE), the information returned will include the resolve results.

AST re-entrant: This routine can be called from either AST level or normal user mode.

(Continued on next page.)

Operation (cont)

⚠ CAUTION

Many of the descriptors in the SYMBOL_RECORD point to internal WORF structures. The user must not modify the buffers pointed to by these descriptors. The user may modify the descriptors that point to buffers in the user's area. The descriptors that point to internal WORF structures are as follows.

```
node_name_dx
rt.alt_node_name_dx
rt.database_name_dx
rt.symbol_name_dx
rt.subscript_dx
tr.alt_node_name_dx
tr.database_name_dx
tr.symbol_name_dx
hs.point_name_dx
hs.root_name_dx
hs.second_point_dx
```

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized
WORF_NULLPTR	A NULL address was passed for *symbol*.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;

char string[512];

SYMBOL_RECORD record;

/* get information on previously-defined symbol */
systat = worf_dsl_get_symbol (dsl_id, symbol_handle, &record);
if (FAILURE (systat))
    exit (systat);

/* print status and error numbers */
printf ("Status code = %08X, error code = %08X", record.status,
    record.error_code);

/* copy node name */
strncpy (string, record.node_name_dx.dsc$a_pointer,
    record.node_name_dx.dsc$w_length);

/* null-terminate name */
string[record.node_name_dx.dsc$w_length] = '\0';

/* print node name */
printf ("Node:  %s\n", string);
```

Notes:

WORF_DSL_MODIFY_SYMBOL

Description

Modifies a Data Source List entry.

Format

WORF_DSL_MODIFY_SYMBOL (*id*, *handle*, *symbol*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be modified.

symbol

Usage: Information about symbol.
Type: Structure SYMBOL_RECORD.
Access: Modify.
Mechanism: By reference.

The address of an element of type SYMBOL_RECORD. Refer to header file WORF_DEF_USER_x (where, x is 'C' or 'FOR') for more information.

Operation

Modifies allowed fields of a Data Source List element. The fields modified depends on the type of DSL. The value of the WORF_DSL_M_INVALID bit is set or cleared, if possible, depending on the corresponding bit in ***symbol*** . Any fields that cannot be changed are simply ignored. The WORF_ERR_M_CLIENT bit is set or cleared to reflect this change.

The best way to use this function is to get a copy of the Data Source List element using WORF_DSL_GET_SYMBOL, and then modify the appropriate fields.

Operation (cont)

After the internal information of WORF is updated, it is copied back into the user's ***symbol*** structure. This allows immediate checking of the information set.

Updates the `user_data` field in the internal structures of WORF with the value from symbol.user_data.

NOTE

The function modifies only the specified symbol. To modify the other symbol in an active/standby pair, this function must be performed on the other symbol separately.

Real-Time DSLs

Fields that can be changed are as follows.

- `symbol.rt.buffer_dx`
- `symbol.rt.subscript`
- `symbol.rt.transfer_count`

If `symbol.rt.transfer_count` is <= 0, 1 is used.

If the symbol has not been resolved, the transfer count can be set to any value, although 1 is recommended. If the symbol has been resolved, WORF_DSL_MODIFY_SYMBOL enforces the following restrictions.

- If the symbol is an array with a variable subscript:
  - `symbol.rt.subscript` is ignored
  - `symbol.rt.transfer_count` is used
  - `symbol.rt.transfer_count` is checked to determine if it is less than or equal to the array dimension.
- If the symbol is an array with a numerical subscript:
  - `symbol.rt.subscript` is used
  - `symbol.rt.transfer_count` is used

Operation

Real-Time DSLs (cont)

- `symbol.rt.subscript` is checked to determine if it is less than the array dimension

- `symbol.rt.subscript` plus `symbol.rt.transfer_count` are checked to determine if it is less than or equal to the array dimension.

• If the symbol is not an array:

- `symbol.rt.subscript` is ignored
- `symbol.rt.transfer_count` must be 1. If this is violated, a return status of WORF_INVALID is returned.

Historian/SPC DSLs

Fields that can be changed are as follows:

- `symbol.hs.point_name_dx`
- `symbol.hs.root_name_dx`
- `symbol.hs.second_point_dx`
- `symbol.hs.operation_type`
- `symbol.hs.point_count`
- `symbol.hs.time_array_dx`
- `symbol.hs.value_array_dx`
- `symbol.hs.start_time`
- `symbol.hs.time_interval`
- `symbol.hs.compression_factor`
- `symbol.hs.interpolation_option`
- `symbol.hs.search_criteria`
- `symbol.hs.search_value`
- `symbol.hs.end_time`
- `symbol.hs.secondary_value_array_dx`
- `symbol.hs.actual_root_dx`
- `symbol.hs.record_count`.

Violating the following causes a return status of WORF_INVALID:

- The value of `symbol.hs.operation_type` is not valid
- The value of `symbol.hs.point_count` is greater than 512.

Trend DSLs

The only field that can be changed is `symbol.tr.buffer_dx`.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_BOUNDS	Transfer outside bounds of array or database
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_INVALID	Information in the entry was not valid
WORF_NOINIT	WORF layer was not initialized
WORF_NULLPTR	A NULL pointer was passed for *symbol*.

Example

Change the value of `transfer_count` for a Real-Time DSL.

```
#include worf_def_user_c
#include macrodef_user

SYMBOL_RECORD record;

systat = worf_dsl_get_symbol (id, handle, &record);

record.rt.transfer_count = 20;

systat = worf_dsl_modify_symbol (id, handle, &record);
if (FAILURE (systat))
   exit (systat);
```

WORF_DSL_PARSE

Description

Parses a symbol specification and fills in the appropriate fields in a DSL entry.

Format

WORF_DSL_PARSE (*id*, *string*, *record*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to which this symbol will be added.

string

Usage: The string to be parsed.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

record

Usage: DSL entry.
Type: Structure SYMBOL_RECORD.
Access: Write only.
Mechanism: By reference.

The pieces parsed from the string are placed into the appropriate descriptors in the symbol.

Operation

WORF_DSL_PARSE parses the input string into several fields, depending on the type of the Data Source List. The fields in ***record*** are loaded, as appropriate. The string descriptors in ***record*** point into ***string***. If the buffer referenced by ***string*** is changed after calling WORF_DSL_PARSE, but before calling WORF_DSL_ADD_SYMBOL, the fields in ***record*** will be wrong.

The syntax expected by WORF_DSL_PARSE is defined in the Symbol Name Syntax section of this manual.

Fields in ***record*** affected by this routine are as follows.

All DSL Types

`record.node_name_dx`	Set by the input string.
`record.status`	Cleared.
`record.error_code`	Cleared.
`record.user_data`	Not changed.

(Continued on next page.)

Operation (cont)

Real-Time Data

`record.rt.alt_node_name_dx`	Set by the input string.
`record.rt.database_name_dx`	Set by the input string.
`record.rt.symbol_name_dx`	Set by the input string.
`record.rt.subscript_dx`	Set by the input string.
`record.rt.tracking_options`	Set by the input string.
`record.rt.buffer_dx`	Cleared.
`record.rt.subscript`	Cleared.
`record.rt.transfer_count`	Cleared.
`record.rt.dimension`	Cleared.
`record.rt.data_length`	Cleared.
`record.rt.record_number`	Cleared.
`record.rt.symbol_type`	Cleared.

Historian/SPC Data

`record.hs.root_name_dx`	Set by the input string.
`record.hs.point_name_dx`	Set by the input string.
`record.hs.operation_type`	Set to HS_DATA_BY_TIME.
`record.hs.search_criteria`	Cleared.
`record.hs.actual_root_dx`	Cleared.
`record.hs.point_count`	Cleared.
`record.hs.time_array_dx`	Cleared.
`record.hs.value_array_dx`	Cleared.
`record.hs.start_time`	Cleared.
`record.hs.time_interval`	Cleared.
`record.hs.compression_factor`	Cleared.
`record.hs.interpolation_option`	Cleared.
`record.hs.search_value`	Cleared.
`record.hs.end_time`	Cleared.
`record.hs.second_point_dx`	Cleared.
`record.hs.points_processed`	Cleared.
`record.hs.match_time`	Cleared.
`record.hs.match_value`	Cleared.
`record.hs.secondary_status`	Cleared.
`record.hs.secondary_value_array_dx`	Cleared.
`record.hs.queue_position`	Cleared.
`record.hs.record_count`	Cleared.
`record.hs.data_type`	Cleared.
`record.hs.secondary_data_type`	Cleared.

(Continued on next page.)

Operation (cont)

Trend Data

`record.tr.alt_node_name_dx`	Set by the input string.
`record.tr.database_name_dx`	Set by the input string.
`record.tr.symbol_name_dx`	Set by the input string.
`record.tr.subscript`	Set by the input string.
`record.tr.tracking_options`	Set by the input string.
`record.tr.buffer_dx`	Cleared.
`record.tr.point_count`	Cleared.
`record.tr.period`	Cleared.
`record.tr.average`	Cleared.
`record.tr.points_returned`	Cleared.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

`SS$_NORMAL`	Normal successful completion.

Error:

`WORF_BADID`	Data Source List ID was not valid
`WORF_NOINIT`	WORF layer was not initialized
`WORF_NULLSTR`	Null string was passed to parser
`WORF_PARSERR`	Error parsing string
`WORF_WRONGTYPE`	Wrong type of Data Source List.

Fatal:

`WORF_UNIMPL`	Unimplemented code path.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle;

SYMBOL_RECORD record;

static const $DESCRIPTOR (symbol_dx, "PROC12::PROC_DB:INPUT_3");


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);


/* test to see if WORF_DSL_PARSE clears fields in the record */
record.rt.record_number = 12;


/* parse the symbol name, and clear fields in "record" */
systat = worf_dsl_parse (dsl_id, &symbol_dx, &record);
if (FAILURE (systat))
    exit (systat);


printf ("Record number = %d\n", record.rt.record_number);
```

WORF_DSL_READ

Description

Reads data for all the symbols in a Data Source List.

Format

WORF_DSL_READ (*id*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be used.

Operation

Causes transactions to occur with servers as required to read the latest values of the symbols in the Data Source List.

This function MUST NOT be called while at AST level. No error message can b returned to the user in this case, but the I/O may not occur correctly.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_DSLEMPTY	Data Source List is empty.

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	Check for DSL entries with problems.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADID	Data Source List ID was not valid
WORF_BADOPS	Invalid mixture of operations on Historian/SPC DSL

Condition Values Returned (cont)

Error:

WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_RESOLVE	Resolving of symbols is required. Call WORF_DSL_RESOLVE
WORF_SERVER	Error code returned from server

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include macrodef_user
#include descripdef_user_c
#include dba$dbdef_user_c

long systat;
long dsl_id;
long symbol_handle[3];

LONGWORD long_array[3];      /* defined in dba$dbdef_user_c */

SYMBOL_RECORD record;

/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* add symbols */
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[0], sizeof (long_array[0]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[0], 0, NULL);
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[1], sizeof (long_array[1]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[1], 0, NULL);
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[2], sizeof (long_array[2]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[2], 0, NULL);
systat = worf_dsl_resolve (dsl_id);

/* read the latest values of the symbols */
systat = worf_dsl_read (dsl_id);
if (FAILURE (systat))
    exit (systat);

printf ("First symbol:  %d", long_array[0].value);
printf ("Second symbol:  %d", long_array[1].value);
printf ("Third symbol:  %d", long_array[2].value);
```

WORF_DSL_RESOLVE

Description

Resolves all symbols in a Data Source List.

Format

WORF_DSL_RESOLVE (*id*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be used.

Operation

Causes symbol resolve to occur. All information in the Data Source List is updated with the resolve results.

Resolving is not necessary if the signature of the databases referenced in the Data Source List match the signatures in the Database List. Resolving can take a long time (seconds, perhaps).

Call WORF_DSL_RESOLVE if WORF_DSL_READ, WORF_DSL_WRITE, WORF_DSL_SYMBOL_RAMP, or WORF_DSL_SYMBOL_WRITE returns a status of WORF_RESOLVE.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

Calling this function for a Data Source List that is not CRISP_REAL_TIME does not cause any I/O to take place. If the DSL has nodes that are timed out, WORF_CHECK is returned. Otherwise, SS$_NORMAL is returned.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

(Continued on next page.)

Condition Values Returned (cont)

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	One or more nodes were not located.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_SERVER	Error code returned from server.

Fatal:

WORF_UNIMPL	Unimplemented code path.

WORF_DSL_SET_STATUS

Description

Sets the status bits of a variable in a Data Source List.

Format

WORF_DSL_SET_STATUS (*id*, *handle*, *status*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be modified.

status

Usage: New status value.
Type: Longword.
Access: Read only.
Mechanism: By value.

The status bits of the symbol to be set.

Operation

Sets the status bits of a specific symbol.

The input value of ***status*** is searched for bits that are on (1). The corresponding bits of the status field of the symbol are then set. Bits in ***status*** that are off (0) cause no change to the status field of the symbol.

The following bits can be specified by the client (refer to the DSL Information section, subsection Status).

WORF_DSL_M_INVALID
WORF_DSL_M_BOUND
WORF_DSL_M_BOTH

This function enables the client to disable read/write of specific symbols without deleting them from the Data Source List.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_IGNORED	Change to 'invalid' bit ignored. Symbol was already invalid.

Error:

WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include macrodef_user

long systat;

/* disable updating of symbol */
systat = worf_dsl_set_status (dsl_id, symbol_handle, WORF_DSL_M_INVALID)

if (systat == WORF_IGNORED)
    printf ("Symbol was already marked as invalid\n");
else if (FAILURE (systat))
    exit (systat);
else
    printf ("Symbol updating now disabled\n");
```

WORF_DSL_SYMBOL_RAMP

Description

Updates a single symbol in a real-time CRISP/32 database.

Format

WORF_DSL_SYMBOL_RAMP (*id*, *handle*, *options*, *ramp*, *low*, *high*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be modified.

options

Usage: Options mask.
Type: Longword.
Access: Read only.
Mechanism: By value.

A mask that specifies the behavior of the server as it writes the symbol.

ramp

Usage: Change in value.
Type: Depends on the CRISP/32 symbol type.
Access: Read only.
Mechanism: By descriptor--fixed-length string descriptor.

The increment.

low

Usage: Low limit.
Type: Depends on the CRISP/32 symbol type.
Access: Read only.
Mechanism: By descriptor--fixed-length string descriptor.

The low limit. The variable is not incremented beyond this value.

Arguments (cont)

high

Usage: High limit.
Type: Depends on the CRISP/32 symbol type.
Access: Read only.
Mechanism: By descriptor--fixed-length string descriptor.

The high limit. The variable is not incremented beyond this value.

Operation

This function modifies the value of a single real-time CRISP/32 symbol. The value of `rt.transfer_count` is ignored. Only one element is changed.

This operation allows the client to modify a single variable in a non-interruptible fashion. When the variable is updated, all other clients are prevented from updating the same variable. The current value of the symbol is read, the value of ***ramp*** is added, and the new value is written to the database.

The server may log the change to its system console, depending on the options bits specified. To change the logging name, refer to WORF_RT_SYMBOL_LOGGING_NAME.

This operation is not valid on character strings.

Arguments ***ramp***, ***low***, and ***high*** must be a structure of the type of the symbol to be modified. Refer to the Data Format section. The descriptors pointed to by ***ramp***, ***high***, and ***low*** must be large enough for the type of symbol being updated, or the overrun status bit will be set.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

Bits that can be specified for ***options*** are as follows.

- WORF_OPT_M_UPDATE_IDLE Have active side update idle side CPU.
- WORF_OPT_M_SYNCH Synchronize update to end of logic.
- WORF_OPT_M_LOG Log operation to CRISP$TT:.
- WORF_OPT_M_NO_LIMITS The result of the ramping is not checked against the high and low limits. The *low* and *high* arguments are not used.
- WORF_OPT_M_TICKDOWN For timers, update the 'tickdown' field. For counters, update the 'count' field. If not specified, update the 'reset' field. The corresponding field from *ramp*, *low*, and *high* must be loaded by the user.

Operation (cont)

NOTE

The following fields are not changed by this operation:

- **Logicals -- 'status'**
- **Timers -- 'countdown' and 'status'**

If the symbol specified is one of a pair of active/standby tracking DSL elements, the server that is in the state being tracked is affected.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	Check for DSL entries with problems
WORF_LIMITS	Attempt to ramp beyond limits; clamped at limit value.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_RESOLVE	Resolving of symbols is required. Call WORF_DSL_RESOLVE
WORF_SERVER	Error code returned from server
WORF_STRING	Attempt to ramp character string
WORF_WRONGTYPE	Data Source List was not real-time CRISP/32 data.

Example

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user
#include dba$dbdef_user_c

long systat;
long dsl_id;
long symbol_handle;

LONGWORD increment;        /* defined in dba$dbdef_user_c */
LONGWORD high_limit;
LONGWORD low_limit;

DSC$DESCRIPTOR low_dx, high_dx, increment_dx;

SYMBOL_RECORD record;

static const $DESCRIPTOR (symbol_dx, "PROC12::PROC_DB:INPUT_3");


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* parse the symbol name, and clear fields in "record" */
systat = worf_dsl_parse (dsl_id, &symbol_dx, &record);

/* add the symbol */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);
systat = worf_dsl_resolve (dsl_id);


increment.value = 1        /* add one */
low_limit.value = 0;
high_limit.value = 100;    /* limit from 0 to 100 */

/* build descriptors to values */
INIT_DX_PTR (&increment_dx, (char *)&increment, sizeof (increment));
INIT_DX_PTR (&low_dx, (char *)&low_limit, sizeof (low_limit));
INIT_DX_PTR (&high_dx, (char *)&high_limit, sizeof (high_limit));

/* increment the symbol's value by one -- log to console */
systat = worf_dsl_symbol_ramp (dsl_id, symbol_handle, WORF_OPT_M_LOG,
                                       &increment_dx, &low_dx, &high_dx);
if (FAILURE (systat))
    exit (systat);
```

WORF_DSL_SYMBOL_WRITE

Description

Updates a single symbol in a real-time CRISP/32 database.

Format

WORF_DSL_SYMBOL_WRITE (*id*, *handle*, *options*, [*value*])

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be modified.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol to be modified.

options

Usage: Options mask.
Type: Longword.
Access: Read only.
Mechanism: By value.

A mask that specifies the behavior of the server as it writes the symbol.

value

Usage: New value.
Type: Depends on the CRISP/32 symbol type.
Access: Read only.
Mechanism: By descriptor -- fixed-length descriptor.

The new value.

Operation

Modifies the value of a single real-time CRISP/32 symbol. The value of `rt.transfer_count` is ignored. Only one element is changed.

This operation enables the client to modify a single variable in a non-interruptible fashion. When the variable is updated, all other clients are prevented from updating the same variable.

The server may log the change to its system console, depending on the options bits specified. To change the logging name, refer to WORF_RT_SYMBOL_LOGGING_NAME.

Operation (cont)

Argument ***value*** must be a structure of the type of the symbol to be modified. Refer to the Data Format section. The descriptor pointed to by ***value*** must be large enough for the type of symbol being updated, or the overrun status bit will be set. If ***value*** is NULL, the new value is taken from the normal data buffer of the symbol.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

The following bits can be specified for ***options***.

- WORF_OPT_M_UPDATE_IDLE — Have active side update idle side CPU.
- WORF_OPT_M_SYNCH — Synchronize update to end of logic.
- WORF_OPT_M_LOG — Log operation to CRISP$TT:.
- WORF_OPT_M_TICKDOWN — For timers, update the 'tickdown' field. For counters, update the 'count' field. If not specified, update the 'reset' field. The corresponding field from ***value*** must be loaded by the user.

NOTE

The following fields are not changed by this operation.

- **Logicals -- 'status'**
- **Timers -- 'countdown' and 'status'**

If the symbol specified is one of a pair of active/standby tracking DSL elements, the server that is in the state being tracked is affected.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Condition Values Returned (cont)

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	Check for DSL entries with problems.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_RESOLVE	Resolving of symbols is required. Call function WORF_DSL_RESOLVE
WORF_SERVER	Error code returned from server
WORF_WRONGTYPE	Data Source List was not real-time CRISP/32 data.

Example

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user
#include dba$dbdef_user_c

long systat;
long dsl_id;
long symbol_handle;

LONGWORD value;            /* defined in dba$dbdef_user_c */

DSC$DESCRIPTOR value_dx;

SYMBOL_RECORD record;

static const $DESCRIPTOR (symbol_dx, "PROC12::PROC_DB:INPUT_3");


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* parse the symbol name, and clear fields in "record" */
systat = worf_dsl_parse (dsl_id, &symbol_dx, &record);

/* add the symbol */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);
systat = worf_dsl_resolve (dsl_id);
```

Example (cont)

```
value.value = 1234;        /* set to 1234 */

/* build descriptor to values */
INIT_DX_PTR (&value_dx, (char *)&value, sizeof (value));

/* set value -- log to console */
systat = worf_dsl_symbol_write (dsl_id, symbol_handle, WORF_OPT_M_LOG,
                                &value_dx);
if (FAILURE (systat))
    exit (systat);
```

WORF_DSL_TRAVERSE

Description

Traverse all the entries in a Data Source List.

Format

WORF_DSL_TRAVERSE (*id*, *argument*, *action*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be traversed.

argument

Usage: Argument to action routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the ***action*** routine.

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The ***action*** routine is called once for each entry in the Data Source List.

Operation

WORF maintains complete information on all symbols in a Data Source List. Rather than duplicating this information, the user can cause WORF to traverse the Data Source List, calling a user's ***action*** routine for each entry in the list.

There is no specific order in which DSL entries are passed to the user's ***action*** routine.

WORF_DSL_TRAVERSE checks the returned value from the user's ***action*** routine. If the value is non-zero, WORF_DSL_TRAVERSE stops traversing the list, and returns the value to the caller.

For active/standby tracking pairs, the user's ***action*** routine is called twice. One of the entries has the WORF_DSL_M_ALTERNATE status bit set.

A common use of ***argument*** is to pass an address to the ***action*** routine. In this case, the ***action*** routine receives the address of a longword that contains the desired address.

(Continued on next page.)

Operation (cont)

⚠ CAUTION

Calling functions that change the Data Source List ordering (WORF_DSL_ADD_SYMBOL, WORF_DSL_DELETE, WORF_DSL_DELETE_SYMBOL, WORF_RT_OPTIMIZE_DSL, WORF_STOP) from the *action* routine may cause erratic operation or even access violations.

The *action* routine may call any other WORF function.

Action Routine

The ***action*** routine is called with the following arguments. Any changes made by the user to ***id***, ***handle***, ***argument***, and ***symbol*** are ignored.

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By reference.

The handle value of the Data Source List.

handle

Usage: Identification handle of the symbol.
Type: Longword.
Access: Read only.
Mechanism: By reference.

The handle value of the ***symbol***.

argument

Usage: Argument to ***action*** routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The address of the argument that was passed to WORF_DSL_TRAVERSE.

Action Routine (cont)

symbol

Usage: Information about symbol.
Type: Structure SYMBOL_RECORD.
Access: Read only.
Mechanism: By reference.

The address of an element of type SYMBOL_RECORD. The user may change any entry in this structure: any changes are ignored. If the user wants to make a change in the copy of the structure of WORF, the user can call WORF_DSL_MODIFY_SYMBOL.

Returns

Usage: Value from user's action routine.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

A zero value means that WORF_DSL_TRAVERSE traversed the entire list, or that an error occurred internal to WORF_DSL_TRAVERSE.

⚠ CAUTION

Many of the descriptors in the SYMBOL_RECORD point to internal WORF structures. The user must not modify the buffers pointed to by these descriptors. The user may modify the descriptors to point to buffers in the user's area. The descriptors that point to internal WORF structures are as follows.

```
node_name_dx
rt.alt_node_name_dx
rt.database_name_dx
rt.symbol_name_dx
rt.subscript_dx
tr.alt_node_name_dx
tr.database_name_dx
tr.symbol_name_dx
hs.point_name_dx
hs.root_name_dx
hs.second_point_dx
```

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

SYMBOL_RECORD matching_record;

long systat;

/* traverse an already-created Data Source List called "dsl_id" */
systat = worf_dsl_traverse (dsl_handle, &matching_record, &traverse_routine);
if (systat != 0)
    printf ("WORF_DSL_TRAVERSE returned value of %d\n", systat);


long traverse_routine (long *dsl_id_ptr, long *symbol_handle_ptr,
                                    long *argument_ptr, SYMBOL_RECORD *record_ptr)
{
    char string[512];

    /* print node name for each symbol */
    strncpy (string, record_ptr->node_name_dx.dsc$a_pointer,
       record_ptr->node_name_dx.dsc$w_length);
    string[record_ptr->node_name_dx.dsc$w_length] = '\0';
    printf ("Node name:  %s\n", string);

    if (strcmp (string, "ND1234") == 0)
    {
       /* copy matching symbol record */
       *(SYMBOL_RECORD *) *argument_ptr = *record_ptr;
       return 1;                /* stop traversing */
    }
    else
       return 0;                /* continue on */
}
```

WORF_DSL_WRITE

Description

Writes data for all the symbols in a Data Source List.

Format

WORF_DSL_WRITE (*id*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be used.

Operation

Causes transactions to occur with servers as required to write the latest values of the symbols in the Data Source List.

This function is valid for Data Source Lists of real-time CRISP/32 data and DSLs of Historian/SPC data that all specify operation types of HS_UPDATE_AT_TIMES. Other DSL types or Historian/SPC functions are an error.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

DSL entries that specify active/standby tracking only update the system that is in the proper state.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Informational:

WORF_DSLEMPTY	Data Source List is empty.

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	Check for DSL entries with problems.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADID	Data Source List ID was not valid
WORF_BADOPS	Invalid mixture of operations on Historian/SPC DSL
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_RESOLVE	Resolving of symbols is required. Call WORF_DSL_RESOLVE
WORF_SERVER	Error code returned from server
WORF_WRONGTYPE	Attempted operation on wrong DSL type.

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include worf_def_user_c
#include macrodef_user
#include dba$dbdef_user_c

long systat;
long dsl_id;
long symbol_handle[3];

LONGWORD long_array[3];    /* defined in dba$dbdef_user_c */

SYMBOL_RECORD record;


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* add symbols */
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[0], sizeof (long_array[0]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[0], 0, NULL);
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[1], sizeof (long_array[1]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[1], 0, NULL);
INIT_DX_PTR (&record.rt.buffer_dx, (char *)long_array[2], sizeof (long_array[2]));
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[2], 0, NULL);
systat = worf_dsl_resolve (dsl_id);


/* modify the local values of the symbols */
long_array[0].value = 30000;
long_array[1].value = -1437;
long_array[2].value = 42;


/* write the values into the database */
systat = worf_dsl_write (dsl_id);
if (FAILURE (systat))
    exit (systat);
```

Notes:

WORF_DUMP

Description

Dumps all WORF internal structures.

Format

WORF_DUMP ()

Operation

Dumps all WORF internal data structures to file `WORF_DUMP.DMP`. Defining logical name `"WORF_DUMP"` overrides the default location and name of the file.

> **⚠ CAUTION**
>
> **This function takes quite a while to do everything. ASTs are disabled during this time.**

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

`SS$_NORMAL`	Normal successful completion.

Error:

`WORF_CORRUPT`	Internal data structures were corrupted.

Notes:

WORF_HS_GET_FILE_LIST

Description

Gets a list of available Historian Point Files (or other files) that match a specified file pattern.

Format

WORF_HS_GET_FILE_LIST (*search_type*, *file_pattern_dx_ptr*, *action_routine*, *argument*)

Arguments

search_type

Usage: Search Type.
Type: Int.
Access: Read only.
Mechanism: By Value.

The type of search to be performed which is either 1) HS_DIR_HPF_MODE which performs a directory search for Historian Point Files (HPF) or 2) HS_DIR_DCL_MODE which performs a DCL-type directory search.

file_pattern_dx_ptr

Usage: File Search Pattern to Match.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

A search pattern that supports wildcard characters (* and %) as a filter for the file specifications that are to be returned in the action routine. The search pattern may contain the following.

- Letters, digits, and underscores (_); must match exactly. (Case is ignored).
- Asterisk (*); matches zero or more characters.
- Percent sign (%); matches exactly one character.

For the Search_Type of HS_DIR_DCL_MODE, the search pattern can be any valid VMS file specification and may be a full file spec consisting of "NODE::DEVICE:[DIRECTORY...]FILE.EXT". It can also be a logical name where the logical name may be defined as a search list.

For the Search_Type of HS_DIR_HPF_MODE, the search pattern may be one of the following format-methods, where the format-method is determined by the syntax.

***file_pattern_dx_ptr* (cont.)**

1. The "DEFAULT" format-method builds the file specification using the default CRISP$HIST_ROOT directory logical:

(ex. `<node::>CRISP$HIST_ROOT:[<database>]symbol_name.HP*`**)**

to search for point file names specified as follows.

Point Name Formats node::database:symbol_name
node::symbol_name
database:symbol_name
symbol_name

2. The "USER-SPECIFIED" format-method uses the point name format directly as specified thus allowing the caller to search for historian point files that reside in directories other than in the CRISP$HIST_ROOT directory path. In all cases, the file extension used for the search is ".HP*".

Point Name Formats:
node::device:[directory]symbol_name
device:[directory]symbol_name
node::logical_directory:symbol_name
logical_directory:symbol_name

action_routine

Usage: Action Routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The action routine is called once for each file found that matches the file search pattern.

argument

Usage: Argument to action routine
Type: Unsigned Int
Access: Read only
Mechanism: By value

An argument that is passed unchanged to the action routine.

Operation

This function transacts with the CRISP Access Server that resides on the first node specified in the file search pattern and gets a list of files that match the specified file search pattern. If a node is not specified in the file search pattern, the search is performed on the local node.

Operation (cont)

It is possible to specify two node names in the file search pattern where the first node name specifies the node that the CRISP Access Server resides on and the second node name is the node specified in the Point Name Format. Example: "node1::node2::database:symbol_name"

The user's action routine is called once for each filename found that matches the file search pattern.

WORF_HS_GET_FILE_LIST checks the returned value from the user's action routine. If the value is non-zero, WORF_HS_GET_FILE_LIST stops transacting with the server, and returns the value to the caller.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case and the I/O may not occur correctly.

Returns

Usage: Condition Code
Type: Unsigned Longword
Access: Write only
Mechanism: By value

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_FILNOTFOUND	File Not Found.

Error:

WORF_ASTDIS	AST's were disabled
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized
WORF_NULLPTR	Null pointer passed for required argument
WORF_NULLSTR	Null string passed for required argument
WORF_SERVER	Error code returned from server
WORF_TRUNC	Node name too long in file pattern
WORF_UNIMPL	Unimplemented code path.

Action Routine

For each file found, the user's action routine is called with the arguments shown below. After all files have been found (or if no files were found), the action routine is called a final time with the "file_counter" argument set to zero.

Action Routine (cont)

Successive calls to the user's action routine will be made for each file found as long as the action routine returns a zero condition code. The user's action routine may stop the transaction process at any time by returning a non-zero condition code, whereupon the WORF_HS_GET_FILE_LIST function call will cease its operation and return the non-zero condition code to its caller.

Format

(action_routine) (*search_type*, *casrv_node_dx_ptr*, *file_pattern_dx_ptr*, *argument*, *file_counter*, *file_found_rec_ptr*)

Arguments

search_type

Usage: Search Type.
Type: Int.
Access: Read only.
Mechanism: By Value.

The type of search performed. This is the value that was passed to WORF_HS_GET_FILE_LIST.

casrv_node_dx_ptr

Usage: CRISP Access Server Node used.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The Node Name of the CRISP Access Server (CASRV) that was accessed.

file_pattern_dx_ptr

Usage: File Search Pattern Matched.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The pattern that filenames must match. This is the address of the descriptor that was passed to WORF_HS_GET_FILE_LIST. If the programmer wants the string to be null-terminated, he should do that before calling WORF_HS_GET_FILE_LIST.

argument

Usage: Argument for user's action routine.
Type: Unsigned Int.
Access: Read only.
Mechanism: By value.

The argument that was passed to WORF_HS_GET_FILE_LIST.

file_counter

Usage: Count of filenames passed.
Type: Int.
Access: Read only.
Mechanism: By value.

Each successive call of the user's action routine is passed a number from 1 to the number of filenames found. A value of 0 signals that no more files were found.

file_found_rec_ptr

Usage: File Found Record.
Type: FILE_FOUND_REC (defined in WORF_DEF_USER_C.H).
Access: Read only.
Mechanism: By reference.

The file found record contains the filename that matched the file search pattern and additional file-info descriptors. For convenience of C programs, the filename is null-terminated.

The additional file-info descriptors can be used by the user to display their own variation of the returned file name. Each file-info descriptor describes a parsed piece of the filename. If a parsed filename does not contain a particular piece of data (ie, database), the associated descriptor length (dsc$w_length) will contain a value of zero.

Returns

Usage: Condition Code.
Type: Int.
Access: Write only.
Mechanism: By value.

Condition Values Returned

User can return the following.

0 - Continue file search.
1 - Stop file search.

Example

```
/* Obtain list of Historian Point Files (HPF) generated from the PAINT  *
 * database for all AREA1_FLOW_RED through AREA6_FLOW_YELLOW tagnames.  */
```

Example (cont)

```
#include stdio
#include stdlib
#include descripdef_user_c
#include macrodef_user
#include worf_def_user_c

main ()
{

    int  systat
    int display_parsed_file = TRUE;

    $DESCRIPTOR (hpf_pattern_dx, "PAINT:*FLOW*");

    /* Initialize WORF */
    systat = worf_init (0);

    /* Get list; DISPLAY_FILE_NAME called for each HPF found */
    systat = worf_hs_get_file_list (HS_DIR_HPF_MODE, &hpf_pattern_dx,
                              &display_file_name, display_parsed_file);

}

/*  ------------------------------------------------------------------------
    DISPLAY_FILE_NAME: Action Routine for "WORF_HS_GET_FILE_LIST" Function Call

    Called by "WORF_HS_GET_FILE_LIST" for each file found that matches the
    file search pattern.  This example routine displays each file found.
    ------------------------------------------------------------------------ */

int display_file_name (int search_type, DSC$DESCRIPTOR *casrv_node_dx_ptr,
    DSC$DESCRIPTOR *file_pattern_dx_ptr, unsigned int argument,
    int file_counter, FILE_FOUND_REC *file_found_rec_ptr)

{
    static int             files_displayed = 0; /* Files Displayed Count */

    /* Print header if this is first file found */
    if (file_counter == 1)
    {
```

Example (cont)

```
printf ("\nHPF-MODE DIRECTORY LIST USING SEARCH SPEC '%.*s'",
   file_pattern_dx_ptr->dsc$w_length,
   file_pattern_dx_ptr->dsc$a_pointer);
printf ("\n\tCRISP Access Server (CASRV) Node : %.*s",
   casrv_node_dx_ptr->dsc$w_length,
   casrv_node_dx_ptr->dsc$a_pointer);
if (file_found_rec_ptr->hist_default_dir_used)
    printf ("\n\tCRISP$HIST_ROOT default directory: 1 (Used)");
else
    printf ("\n\tCRISP$HIST_ROOT default directory: 0 (Not-Used)");
 }

 /* Print contents of FILE_FOUND_REC */
 if (file_counter > 0)
 {
printf ("\n%d. FullFileSpec: %.*s", file_counter,
   file_found_rec_ptr->full_file_spec_dx.dsc$w_length,
   file_found_rec_ptr->full_file_spec_dx.dsc$a_pointer);

/* Print parsed pieces of file if "argument" set to TRUE */
if (argument)
{
    printf ("\n\tNode     : %.*s",
      file_found_rec_ptr->node_spec_dx.dsc$w_length,
      file_found_rec_ptr->node_spec_dx.dsc$a_pointer);
    printf ("\n\tDirectory: %.*s",
      file_found_rec_ptr->dir_spec_dx.dsc$w_length,
      file_found_rec_ptr->dir_spec_dx.dsc$a_pointer);
    printf ("\n\tFilename : %.*s",
      file_found_rec_ptr->file_spec_dx.dsc$w_length,
      file_found_rec_ptr->file_spec_dx.dsc$a_pointer);
    printf ("\n\tDatabase : %.*s",
      file_found_rec_ptr->db_spec_dx.dsc$w_length,
      file_found_rec_ptr->db_spec_dx.dsc$a_pointer);
    printf ("\n\tSymbol   : %.*s",
      file_found_rec_ptr->sym_spec_dx.dsc$w_length,
      file_found_rec_ptr->sym_spec_dx.dsc$a_pointer);
    printf ("\n\tSubscript: %.*s",
      file_found_rec_ptr->sub_spec_dx.dsc$w_length,
      file_found_rec_ptr->sub_spec_dx.dsc$a_pointer);
    printf ("\n\tDataType : %.*s",
      file_found_rec_ptr->data_type_dx.dsc$w_length,
      file_found_rec_ptr->data_type_dx.dsc$a_pointer);
    printf ("\n\tStartTime: %.*s",
      file_found_rec_ptr->start_time_dx.dsc$w_length,
      file_found_rec_ptr->start_time_dx.dsc$a_pointer);
}
files_displayed = file_counter;
```

Example (cont)

```
    }
    else  /* (file_counter <= 0) */
    {
   if (files_displayed > 0) {
       printf ("\n");
       files_displayed = 0;
   }
   else
       printf ("\n** No Files Found **\n");
    }

    /*   ------------------------------------------
   Return "0" to enable additional file searches
   Return "1" to terminate file search
   ------------------------------------------ */
    return (0);

}
```

WORF_INIT

Description

Initializes the WORF communications system.

Format

WORF_INIT (*options*)

Arguments

options

Usage: Options bit mask.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

Passed unchanged to WORF_CONFIG.

Operation

WORF_CONFIG may be called before this routine. If WORF_CONFIG has not previously been called, WORF_INIT will call WORF_CONFIG and pass it the value of ***options***. None of the ***options*** are defined.

WORF_INIT must be called before using any other WORF routines.

The WORF layer connects to the Software Bus using the configuration information set up by WORF_CONFIG. If WORF cannot connect to the SWB using the configuration values, Software Bus communications will not be attempted. The Software Bus connect name is specified in the configuration information; if zero length, the Process ID in hexadecimal is used. If CRISP is not installed on the local machine, WORF_INIT does not connect to the Software Bus.

The WORF layer opens channels to IEEE 802.3 network numbers 0 and 1. If there are channels already opened on those networks, a warning is returned. The memory used for read-ahead buffers is taken from the client's BYTLM quota. If the configuration value of `init.ieee_buffers` is zero, no network channels are created.

The file CRISP$CFG:WORF_ALIAS.DAT is opened and read. The table is assumed to contain pairs of names: the first name is an 'alias', and the second name is an actual node name of a system running CRISP/32. References to the alias value are translated to the actual node name used internally by WORF. Defining logical name WORF_ALIAS overrides the default location and name of the file. Beginning with V2.8-21, an 'actual' node name of "0" is allowed. This node name always refers to the local node. A different alias file can be specified by defining configuration value `init2.alias_file_name_dx` (V2.8-23 and later).

Establishes an exit handler so that the user does not have to explicitly call WORF_STOP.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_NONETWORK	No network channels could be created
WORF_STEALING	WORF will be using channels already opened for IEEE 802 communications.

Error:

WORF_ALRINIT	WORF layer already initialized
WORF_CORRUPT	Internal data structures were corrupted.

Fatal:

WORF_BADALIAS	Format of the alias table was not valid
WORF_NOCOMM	Neither Software Bus nor IEEE 802 communications were established
WORF_NOCONFIG	Unable to configure WORF layer.

WORF_NETWORK_RECONNECT

Description

Closes then re-opens network channels.

Format

WORF_NETWORK_RECONNECT (*options*)

Arguments

options

Usage: Options bit mask.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

Options to use during channel initialization. No options are currently defined.

Operation

The IEEE 802.3 channels opened by WORF are closed and then re-opened. The user may modify the value of `init.ieee_buffers` and then call this function to put the new value into effect.

Because WORF can transact with multiple network nodes simultaneously, increasing the number of read-aheads improves performance. Generally, the optimum number of read-heads is one greater than the number of nodes in a Data Source List.

An I/O channel will be opened on each Ethernet line in the system. The memory used for read-ahead buffers is taken from the client's BYTLM quota. If the configuration value of `init.ieee_buffers` is zero, no network channels are opened.

Channels 'borrowed' (that is, already open when WORF_INIT was called) are not affected by this operation.

⚠ CAUTION

It is not possible to guarantee that the IEEE 802.3 SAP will remain the same when the channel is closed and then opened. This function will request the same SAP, but the results of the request cannot be guaranteed.

ASTs must be enabled for this function to work properly.

This function MUST NOT be called while at AST level. No error message can be returned to the user in the event of an error and the function will not operate properly.

To change `init.retries` and `init.timeout`, refer to function WORF_CONFIG_TIMEOUT. To change the Software Bus configuration, refer to function WORF_SWB_RECONNECT.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Failure statues from CSP802$CREATE and CSP802$ELIMINATE

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_NONETWORK	No network channels could be created.

Error:

WORF_ASTDIS	AST's were disabled
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized.

Example

```
#include worf_def_user_c
#include macrodef_user

main ()
{
    WORF_CONFIG *config_ptr;

    config_ptr = worf_config (0);

    systat = worf_init (0);


    /* close all network channels and don't re-open */
    config_ptr->init.ieee_buffers = 0;
    systat = worf_network_reconnect (0);
    if (FAILURE (systat) && systat != WORF_NONETWORK)
        exit (systat); /* expect WORF_NONETWORK in this case */


    /* open all network channels with 10 read-aheads */
    config_ptr->init.ieee_buffers = 10;
    systat = worf_network_reconnect (0);
    if (FAILURE (systat))
        exit (systat);
}
```

WORF_NL_GET_NODE_STATS_LIST

Description

Reads the statistics from the Node List.

Format

WORF_NL_GET_NODE_STATS_LIST (*action, argument*)

Arguments

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The action routine is called once for each Node List entry.

argument

Usage: Argument to *action* routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the ***action*** routine.

Operation

This function allows the user to retrieve information about nodes in the network. The nodes for which information is available are those that have been specified by at least one element in a Data Source List.

The statistics are updated by each operation such as WORF_DSL_READ and WORF_DSL_WRITE; therefore, the information on a given node may not be very current.

The user's ***action*** routine is called once for each node.

WORF_NL_GET_NODE_STATS_LIST checks the returned value from the user's ***action*** routine. If the value is non-zero, the function stops calling the user's ***action*** routine, and returns the value to the caller.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Action Routine

For each node, the ***action*** routine is called with the following arguments. After all nodes have been processed (or if no nodes are in the list), the action routine is called once again, with a NULL pointer for the ***node*** descriptor. Any changes made by the user to ***node***, ***argument***, ***number***, or ***stats*** are ignored.

Action Routine (cont)

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the node. For convenience of C programs, is null-terminated.

argument

Usage: Argument to ***action*** routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The address of the argument that was passed to WORF_NL_GET_NODE_LIST_STATS.

number

Usage: Item number of node.
Type: Longword.
Access: Read only.
Mechanism: By reference.

Each successive call of the user's action routine is passed a number from 1 to the number of nodes in the Node List. A value of 0 signals that the previous call to the ***action*** routine was the last database locate.

stats

Usage: Statistics for the node.
Type: Structure of type NODE_STATISTICS.
Access: Read only.
Mechanism: By reference.

The statistics for the node, including the following fields.

casrv_count The total number of times that communications have failed with the CRISP Access Server of the node.

Action Routine

***stats* (cont)**

dbasrv_count The total number of times that communications havefailed with the Database Access Server of the node.

status A bit mask of status information, as defined in the following.

WORF_NODE_M_LOCATED	The node has been located.
WORF_NODE_M_SWB	The node is reachable over the Software Bus.
WORF_NODE_M_IEEE802	The node is reachable over the IEEE 802.3 network.
WORF_NODE_M_TMO_CASRV	The CRISP Access Server is currently timed out.
WORF_NODE_M_TMO_DBASRV	The Database Access Server is currently timed out.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Any non-zero status from the user's action routine.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized.

Example

DISPLAY ALL NODE LIST ENTRIES

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long node_action (DSC$DESCRIPTOR *node_dx_ptr, long **node_count_ptr2,
    long *number_ptr, NODE_LIST *stats_ptr);


main ()
{
    long systat;
    long node_count;


    / * assume WORF is initialized, DSL's have been resolved, etc. * /


    printf ("\n\nNode List:");

    systat = worf_nl_get_node_stats_list (&node_action, &node_count);

    printf ("\nThere were %d nodes in the Node List", node_count);
}

long node_action (DSC$DESCRIPTOR *node_dx_ptr, long **node_count_ptr2,
    long *number_ptr, NODE_LIST *stats_ptr)
{
     if (*number_ptr != 0)
     {
         **node_count_ptr2 = *number_ptr;

         printf
             ("\n%s:: Status Mask %08X, CASRV count = %d, DBASRV count = %d",
             node_dx_ptr->dsc$a_pointer, stats_ptr->status,
             stats_ptr->casrv_count, stats_ptr->dbasrv_count);
     }

     return 0;        / * Keep going * /
}
```

WORF_PATH_SELECT

Description

Enables or disables the use of various communications paths by WORF.

Format

WORF_PATH_SELECT (*net0*, *net1*, *swb*)

Arguments

net0

Usage: Network 0 enable.
Type: Longword.
Access: Read only.
Mechanism: By value.

If 0, use of IEEE 802.3 network 0 by WORF is disabled.

net1

Usage: Network 1 enable.
Type: Longword.
Access: Read only.
Mechanism: By value.

If 0, use of EEE 802.3 network 1 by WORF is disabled.

swb

Usage: Software Bus enable.
Type: Longword.
Access: Read only.
Mechanism: By value.

If 0, use of the Software Bus by WORF is disabled.

Operation

By default, WORF uses all communications paths available to it.

There may be cases where the user would like to force WORF to use only certain paths. For example, it may be desirable to limit WORF to IEEE 802.3 network 1 (typically the 'alternate' network) unless network 1 fails.

This function can be called at any time after WORF_INIT. The user can dynamically control which communications paths can be used.

NOTE

If a path could not be initialized by WORF_INIT, disabling or enabling it has no effect.

Operation (cont)

Passing a non-zero value for the associated 'enable' enables the path, if possible. Passing a zero value disables the path. Note that any channel opened stays open, even if the path is disabled.

It is possible to completely disconnect from the Software Bus by calling WORF_SWB_RECONNECT. All network channels can be closed by calling WORF_NETWORK_RECONNECT.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_NOINIT	WORF layer was not initialized.

Example

```
#include worf_def_user_c
#include descripdef_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle;

SYMBOL_RECORD record;

static const $DESCRIPTOR (symbol_dx, "PROC12::PROC_DB:INPUT_3");


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* parse the symbol name, and clear fields in "record" */
systat = worf_dsl_parse (dsl_id, &symbol_dx, &record);

/* add the symbol */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);


/* resolve -- this could be successful */
systat = worf_dsl_resolve (dsl_id);


/* disable use of all communications paths */
systat = worf_path_select (0, 0, 0);
if (FAILURE (systat))
    exit (systat);


/* this time, resolve will time out */
systat = worf_dsl_resolve (dsl_id);
```

Notes:

WORF_RT_GET_DB_LIST

Description

Gets a list of available databases on a node.

Format

WORF_RT_GET_DB_LIST (*node*, *action*, *argument*)

Arguments

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the VMS node.

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The *action* routine is called once for each database.

argument

Usage: Argument to action routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the ***action*** routine.

Operation

This function transacts with the specified node and returns a list of available databases.

The user's ***action*** routine is called once for each database.

WORF_RT_GET_DB_LIST checks the returned value from the user's ***action*** routine. If the value is non-zero, WORF_RT_GET_DB_LIST cancels any pending reads on the network channels and returns the value to the caller.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

(Continued on next page.)

Operation (cont)

Action Routine

For each database, the ***action*** routine is called with the following arguments. After all databases have been located (or if no databases are located), the ***action*** routine is called once again, with a NULL pointer for the ***db*** descriptor. Any changes made by the user to ***argument*** or *number* are ignored.

NOTE
The *action* routine is called with AST's disabled.

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the VMS node. This is the address of the descriptor that was passed to WORF_RT_GET_DB_LIST.

db

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP real-time database. For convenience of C programs, it is also null-terminated.

argument

Usage: Argument to action routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The argument that was passed to WORF_RT_GET_DB_LIST.

Operation

Action Routine (cont)

number

Usage: Item number of database.
Type: Longword.
Access: Read only.
Mechanism: By reference.

Each successive call of the user's ***action*** routine is passed a number from 1 to the number of databases located. A value of 0 signals that the previous call to the ***action*** routine was the last database located.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Any non-zero status from the user's ***action*** routine.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request.

Error:

SS$_TIMEOUT	Node did not respond within the timeout interval
WORF_ASTDIS	ASTs were disabled
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_SERVER	Error code returned from server
WORF_TRUNC	The length of ***node*** was too long.

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include stdio
#include stdlib
#include descripdef_user_c
#include macrodef_user
#include worf_def_user_c

static long database_count;

main ()
{
    long systat;
    long idx;

    char node[512];

    unsigned short count;

    static const $DESCRIPTOR (node_dx, "RPZ14");

    systat = worf_init (0);

    database_count = 0;

    /* get the list */
    systat = worf_rt_get_db_list (&node_dx, &display_routine, 0);
}

long display_routine (DSC$DESCRIPTOR *node_dx_ptr,
    DSC$DESCRIPTOR *db_dx_ptr,  unsigned long *argument_ptr, long *index_ptr)
{
    /* check for exit call */
    if (*index_ptr == 0 || db_dx_ptr == NULL)
    {
        printf ("\n\nThere were %d databases found", database_count);
        return 0;
    }

    ++database_count;

    printf ("\n%3d %s::%s", *index_ptr, node_dx_ptr->dsc$a_pointer,
        db_dx_ptr->dsc$a_pointer);

    return 0;
}
```

WORF_RT_GET_NODE_LIST

Description

Gets a list of nodes that have the specified real-time database.

Format

WORF_RT_GET_NODE_LIST (db, action, argument)

Arguments

db

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP real-time database.

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The ***action*** routine is called once for each node that has database ***db***.

argument

Usage: Argument to ***action*** routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the ***action*** routine.

Operation

This function uses the CRISP multicast to build a list of nodes that have the specified database.

Important Notes

- This function temporarily intercepts all messages on the network channels. If WORF is sharing network channels, the user's AST routine may not be called for the duration of this function.
- This function retains control for 1/2 second, to ensure that all responses have been received.
- Successful operation of this function may require large numbers of read-ahead buffers for the network channels if many machines have the desired database (refer to WORF_CONFIG).

(Continued on next page.)

Operation (cont)

- Responses from pre-CRISP/32 V2.7 systems are ignored. However, large numbers of pre-V2.7 systems can block responses from V2.7 systems, if not enough read-ahead buffers are allocated.

- It is not possible to locate CRISP/32 system databases with this function, since all CRISP/32 systems have such databases.

- Not all nodes with the database may be located, since responses may be delayed beyond the timeout interval.

- The user's ***action*** routine is called in AST mode for nodes located on the network. The ***action*** routine is called in normal mode if the desired database is located on the local machine (over the Software Bus).

  The user's ***action*** routine is called once for each node that has the database.

  WORF_RT_GET_NODE_LIST checks the returned value from the user's ***action*** routine. If the value is non-zero, WORF_RT_GET_NODE_LIST cancels any pending reads on the network channels and returns the value to the caller.

  This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

Action Routine

For each node located, the ***action*** routine is called with the following arguments. After all nodes are located (or if no nodes are located), the ***action*** routine is called once again, with a NULL pointer for the ***node*** descriptor. Any changes made by the user to ***argument*** or ***number*** are ignored.

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the VMS node. For convenience of C programs, it is also null-terminated.

db

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP real-time database. This is the address of the descriptor that was passed to WORF_RT_GET_NODE_LIST.

(Continued on next page.)

Action Routine (cont)

argument

Usage: Argument to ***action*** routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The argument that was passed to WORF_RT_GET_NODE_LIST.

number

Usage: Item number of ***symbol***.
Type: Longword.
Access: Read only.
Mechanism: By reference.

Each successive call of the user's action routine is passed a number from 1 to the number of nodes located. A value of signals that the previous call to the ***action*** routine was the las node located.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Any non-zero status from the user's ***action*** routine.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_SERVER	Error code returned from server
WORF_TRUNC	The length of ***db*** was not valid.

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include stdio
#include stdlib
#include descripdef_user_c
#include macrodef_user
#include worf_def_user_c

static long node_count;

main ()
{
    long systat;

    char database[512];

    WORF_CONFIG *config_ptr;

    static const int $DESCRIPTOR (database_dx, "SPD124");


    /* increase number of read-ahead buffers before opening network channels */
    config_ptr = worf_config (0);
    config_ptr->init.ieee_buffers = 10;

    systat = worf_init (0);

    node_count = 0;

    /* get the symbols */
    systat = worf_rt_get_node_list (&database_dx, &display_routine, 0);
}

long display_routine (DSC$DESCRIPTOR *node_dx_ptr,
    DSC$DESCRIPTOR *db_dx_ptr, unsigned long *argument_ptr, long *index_ptr)
{
    /* check for exit call */
    if (*index_ptr == 0 || node_dx_ptr == NULL)
    {
        printf ("\n\nThere were %d nodes found", node_count);
        return 0;
    }

    ++node_count;

    printf ("\n%3d %s::%s", *index_ptr, node_dx_ptr->dsc$a_pointer,
        db_dx_ptr->dsc$a_pointer);

    return 0;
}
```

WORF_RT_GET_SYMBOL_LIST

Description

Gets a list of available symbols in a real-time database.

Format

WORF_RT_GET_SYMBOL_LIST (*node*, *database*, *pattern*, *action*, *argument*)

Arguments

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the VMS node.

database

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP real-time database.

pattern

Usage: Symbol name pattern.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The pattern that symbols must match.

action

Usage: Action routine.
Type: Function.
Access: Read only.
Mechanism: By Reference.

The ***action*** routine is called once for each symbol that matches ***pattern***.

argument

Usage: Argument to action routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By value.

An argument that is passed unchanged to the ***action*** routine.

Operation

This function transacts with the ***node*** and gets a list of symbols that match ***pattern*** in the specified ***database***.

The user's ***action*** routine is called once for each symbol name in the database.

WORF_RT_GET_SYMBOL_LIST checks the returned value from the user's action routine. If the value is non-zero, WORF_RT_GET_SYMBOL_LIST stops transacting with the server and returns the value to the caller.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not occur correctly.

The pattern may contain any of the following.

- Letters, digits, and underscores (_). These must match exactly, except that case is ignored.
- Asterisks (*). This matches zero or more characters.
- Percent signs (%). This matches exactly one character.
- WORF_TIMEOUT_RECOVERY **must not** be called while this function is in progress.

Action Routine

For each symbol located, the ***action*** routine is called with the following arguments. After all symbols have been located (or if no symbols are located), the ***action*** routine is called once again, with a NULL pointer for the ***symbol*** descriptor. Any changes made by the user to ***argument*** or ***number*** are ignored.

NOTE

The *action* routine is called with ASTs disabled.

node

Usage: Node name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the VMS node. This is the address of the descriptor that was passed to WORF_RT_GET_DB_LIST.

Action Routine (cont)

database

Usage: Database name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The name of the CRISP real-time database. This is the address of the descriptor that was passed to WORF_RT_GET_SYMBOL_LIST. If the string is to be null-terminated, this should be done before calling WORF_RT_GET_SYMBOL_LIST.

pattern

Usage: Symbol name pattern.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The pattern that symbols must match. This is the address of the descriptor that was passed to WORF_RT_GET_SYMBOL_LIST. If the programmer wants the string to be null-terminated, it should be done before calling WORF_RT_GET_SYMBOL_LIST.

argument

Usage: Argument to ***action*** routine.
Type: Unsigned longword.
Access: Read only.
Mechanism: By reference.

The argument that was passed to WORF_RT_GET_SYMBOL_LIST.

number

Usage: Item number of symbol.
Type: Longword.
Access: Read only.
Mechanism: By reference.

Each successive call of the user's ***action*** routine is passed a number from 1 to the number of symbols located. A value of 0 indicates that the previous call to the ***action*** routine was the last symbol located.

Action Routine (cont)

symbol

Usage: Symbol name pattern.
Type: Character string.
Access: Read only.
Mechanism: By descriptor -- fixed-length string descriptor.

The symbol name that matched the pattern. For convenience of C programs, it is also null-terminated.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Any non-zero status from the user's ***action*** routine.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request.
WORF_CHECK	Database not found.

Error:

SS$_TIMEOUT	Node did not respond within the timeout interval
WORF_ASTDIS	ASTs were disabled
WORF_CORRUPT	Internal data structures were corrupted
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_SERVER	Error code returned from server
WORF_TRUNC	The length of ***node*** or ***database*** was not valid.

Fatal:

WORF_UNIMPL	Unimplemented code path.

Example

```
#include stdio
#include stdlib
#include descripdef_user_c
#include macrodef_user
#include worf_def_user_c

static long symbol_count;

main ()
{
    long systat;

    char node[512];
    char database[512];
    char pattern[512];

    static const $DESCRIPTOR (node_dx, "SUZQ19");
    static const $DESCRIPTOR (database_dx, "CRISP");
    static const $DESCRIPTOR (pattern_dx, "*");


    systat = worf_init (0);

    symbol_count = 0;

    /* get the symbols */
    systat = worf_rt_get_symbol_list (&node_dx, &database_dx, &pattern_dx,
        &display_routine, 0);
}

long display_routine (DSC$DESCRIPTOR *node_dx_ptr,
    DSC$DESCRIPTOR *db_dx_ptr, DSC$DESCRIPTOR *pattern_dx_ptr,
    unsigned long *argument_ptr, long *index_ptr,
    DSC$DESCRIPTOR *symbol_dx_ptr)
{
    if (*index_ptr == 1)
        printf ("\nPattern %s::%s:%s\n", node_dx_ptr->dsc$a_pointer,
            db_dx_ptr->dsc$a_pointer, pattern_dx_ptr->dsc$a_pointer);

    /* check for exit call */
    if (*index_ptr == 0 || symbol_dx_ptr == NULL)
    {
        printf ("\n\nThere were %d symbols found", symbol_count);
        return 0;
    }

    ++symbol_count;
```

Example (cont)

```
    printf ("\n%3d %s::%s:%s", *index_ptr, node_dx_ptr->dsc$a_pointer,
        db_dx_ptr->dsc$a_pointer, symbol_dx_ptr->dsc$a_pointer);

    return 0;
}
```

WORF_RT_NEXT_SYMBOL

Description

Locates adjacent symbol names from a CRISP real-time database.

Format

WORF_RT_NEXT_SYMBOL (*id*, *handle*, *increment*, *options*, *symbol*)

Arguments

id

Usage: Identification ***handle*** of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List.

handle

Usage: Identification ***handle*** of the ***symbol***.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the symbol at which to start.

increment

Usage: Amount and direction of change.
Type: Signed word.
Access: Read only.
Mechanism: By value.

The direction and distance to move in the database symbol table.

options

Usage: Options mask.
Type: Longword.
Access: Read only.
Mechanism: By value.

How to move in the database. A value of 0 means in alphabetical order. If WORF_OPT_M_DECL, moves in declaration order.

symbol

Usage: Information about symbol.
Type: Structure SYMBOL_RECORD.
Access: Write only.
Mechanism: By reference.

The address of an element of type SYMBOL_RECORD. Refer to header file WORF_DEF_USER_x (where, x is 'C' or 'FOR') for more information.

Operation

Using the specified real-time symbol as a starting point, moves forwards or backwards in the CRISP database. This function steps through a database in alphabetical order (the default) or in declaration order.

If any error is reported by the server (symbol not located, end of database reached, etc.), the function returns status WORF_CHECK, and ***symbol*** is not updated.

NOTE

The starting symbol must have been successfully resolved for this function to work properly.

The user must load descriptors `symbol.rt.symbol_name_dx` and `symbol.rt.buffer_dx` with valid locations in the user's data area. The function copies the symbol name and value into the user's area, to a maximum length set by the descriptors. The length field in `symbol.rt.symbol_name_dx` is then set to the length of the symbol name.

The following items are set in ***symbol*** to make adding to a DSL easier.

	`symbol.status`	Cleared, then set for the symbol.
	`symbol.error_code`	Cleared, then set for the symbol.
‹	`symbol.node_name_dx`	Copied from starting symbol.
‹	`symbol.rt.database_name_dx`	Copied from starting symbol.
‹	`symbol.rt.alt_node_name_dx`	Copied from starting symbol.
	`symbol.rt.tracking_options`	Copied from starting symbol.
	`symbol.rt.dimension`	Set for the symbol located.
	`symbol.rt.data_length`	Set for the symbol located.
	`symbol.rt.record_number`	Set for the symbol located.
	`symbol.rt.symbol_type`	Set for the symbol located.
	`symbol.rt.symbol_name_dx`	User's buffer loaded with the symbol name.
	`symbol.rt.buffer_dx`	User's buffer loaded with the value of the symbol. For subscripted variables, the first element (subscript 0) is loaded.

Operation (cont)

The other entries in the symbol record may be undefined.

⚠ CAUTION

These descriptors point to internal WORF structures. The user must not modify the buffers pointed to by these descriptors. The user may modify the descriptors to point to buffers in the user's area.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Warning:

WORF_ABORTED	I/O operation aborted at user's request
WORF_CHECK	Function was unable to complete: symbol not located, database not located, etc
WORF_TRUNC	Symbol name did not fit in the symbol.rt.symbol_name_dx descriptor.

Error:

WORF_ASTDIS	ASTs were disabled
WORF_BADHANDLE	Symbol handle was not valid
WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_LIST2BIG	Data Source List too large
WORF_NOINIT	WORF layer was not initialized
WORF_RESOLVE	Resolving of symbols is required. Call WORF_DSL_RESOLVE
WORF_SERVER	Error code returned from server
WORF_WRONGTYPE	Data Source List was not real-time CRISP/32 data.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c
#include macrodef_user
#include descripdef_user_c
#include dba$dbdef_user_c
#include dba$def_user_c

long systat;
long dsl_handle;
long symbol_handle;

short increment;

SYMBOL_RECORD rec;

static const $DESCRIPTOR (symbol_dx, "TPR123::TEST_DB:FRE0123");

char new_symbol[512];
char string[512];
char new_value[512];

systat = worf_init (0);

systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_handle);

systat = worf_dsl_parse (dsl_handle, &symbol_dx, &rec);

systat = worf_dsl_add_symbol (dsl_handle, &rec, &symbol_handle, 0, NULL);

systat = worf_dsl_resolve (dsl_handle);

systat = worf_dsl_get_symbol (dsl_handle, symbol_handle, &rec);

printf ("\nBase symbol record number = %d", rec.rt.record_number);


/* move 5 symbols down the database */
increment = 5;

/* set up descriptors */
INIT_DX_PTR (&rec.rt.symbol_name_dx, &new_symbol[0], sizeof (new_symbol));
INIT_DX_PTR (&rec.rt.buffer_dx, &new_value[0], sizeof (new_value));

systat = worf_rt_next_symbol (dsl_handle, symbol_handle, increment,
    WORF_OPT_M_DECL, &rec);
if (FAILURE (systat))
    exit (systat);
```

Example (cont)

```
strncpy (string, rec.node_name_dx.dsc$a_pointer,
    rec.node_name_dx.dsc$w_length);
string[rec.node_name_dx.dsc$w_length] = '\0';      /* null-terminate */
printf ("Node name %s\n", string);

printf ("\nStatus %08X, Error code %08X", rec.status, rec.error_code);

strncpy (string, rec.rt.database_name_dx.dsc$a_pointer,
    rec.rt.database_name_dx.dsc$w_length);
string[rec.rt.database_name_dx.dsc$w_length] = '\0';  /* null-terminate */
printf ("Database name %s\n", string);

strncpy (string, rec.rt.symbol_name_dx.dsc$a_pointer,
    rec.rt.symbol_name_dx.dsc$w_length);
string[rec.rt.symbol_name_dx.dsc$w_length] = '\0'; /* null-terminate */
printf ("Symbol name %s\n", string);
```

Notes:

WORF_RT_OPTIMIZE_DSL

Description

Optimizes a Data Source List.

Format

WORF_RT_OPTIMIZE_DSL (*id*)

Arguments

id

Usage: Identification handle of the Data Source List.
Type: Longword.
Access: Read only.
Mechanism: By value.

The handle value of the Data Source List to be optimized.

Operation

The Database Access Server (DBASRV) can optimize the list of data to be returned to a client, if the list is sorted by record number.

This function is called after the DSL is resolved. The DSL entries are then sorted by record number, causing a new list to be sent to the server. Transactions with the DBASRV might then be faster.

The degree of optimization cannot be predicted. However, transactions shall be no slower than before optimization.

This function does not change the handles of the DSL entries.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are error codes from memory allocation routines, plus the following.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_BADID	Data Source List ID was not valid
WORF_INPROGRESS	WORF I/O in progress
WORF_NOINIT	WORF layer was not initialized
WORF_WRONGTYPE	Data Source List was not real-time CRISP/32 data.

Example

```
#include worf_def_user_c
#include macrodef_user

long systat;
long dsl_id;
long symbol_handle[3];

SYMBOL_RECORD record;


/* create a real-time Data Source List */
systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);

/* add symbols */
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[0], 0, NULL);
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[1], 0, NULL);
systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle[2], 0, NULL);

/* resolve */
systat = worf_dsl_resolve (dsl_id);

/* optimize list */
systat = worf_rt_optimize_dsl (dsl_id);
if (FAILURE (systat))
    exit (systat);
```

WORF_RT_SYMBOL_LOGGING_NAME

Description

Enables the user to specify the name with which the Database Access Server logs a WORF_DSL_SYMBOL_RAMP or WORF_DSL_SYMBOL_WRITE.

Format

WORF_RT_SYMBOL_LOGGING_NAME (*name*)

Arguments

name

Usage: Logging name.
Type: Character string.
Access: Read only.
Mechanism: By descriptor (fixed-length string descriptor).

The address of a string descriptor that contains the desired name.

Operation

When logging is specified in a call to WORF_DSL_SYMBOL_RAMP or WORF_DSL_SYMBOL_WRITE, the Database Access Server (DBASRV) prints a message to the logging device. This message includes a user name.

The default user name is the node name and the Software Bus connect name. If no Software Bus connect name was specified by the user, the connect name is the VMS process name.

The specified name is truncated to 512 characters.

Passing a NULL pointer for the name, a descriptor whose length is zero, or a descriptor that has a NULL pointer, restores the default behavior.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Can be called before WORF_INIT and WORF_CONFIG are called.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Example

```
      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 RETSTAT
      INTEGER*4 DSL_ID
      INTEGER*4 SYMBOL_HANDLE


C     (Assume that DSL is already built)
C     (Assume that SYMBOL_HANDLE refers to CRISP:CSP_S_TIME, which is a string)

C     Update the symbol, with logging; use default name
      retstat = worf_dsl_symbol_write (%val (dsl_id),
     1   %val (symbol_handle), %val (WORF_OPT_M_LOG), 'HI MOM')
      if (.not.retstat) call exit (retstat)

C     Change the logging name
      retstat = worf_rt_symbol_logging_name ('LINE 12, CRT 47')
      if (.not.retstat) call exit (retstat)

C     Update the symbol, with logging; use new name
      retstat = worf_dsl_symbol_write (%val (dsl_id),
     1   %val (symbol_handle), %val (WORF_OPT_M_LOG), 'HI SON')
      if (.not.retstat) call exit (retstat)
```

If communications is over the Software Bus, the messages logged are as follows.

```
GPX1::2DC00835 [SWB 2DC00835-0001]
        modified String CRISP:CSP_S_TIME on 15-MAR-1991
11:12:13.35
        from "15-MAR-1991 11:12:12.88" to "HI MOM"
LINE 12, CRT 47 [SWB 2DC00835-0001]
        modified String CRISP:CSP_S_TIME on 15-MAR-1991
11:12:13.35
        from "HI MOM" to "HI SON"
```

If communications is over the IEEE 802.3 network, the messages logged are as follows.

```
GPX1::2DC00835 [AA-00-04-00-02-04 (FC-00)-0001]
        modified String CRISP:CSP_S_TIME on 15-MAR-1991
11:13:05.31
        from "15-MAR-1991 11:13:04.88" to "HI MOM"
LINE 12, CRT 47 [AA-00-04-00-02-04 (FC-00)-0001]
        modified String CRISP:CSP_S_TIME on 15-MAR-1991
11:13:05.31
        from "HI MOM" to "HI SON"
```

WORF_STOP

Description

Stops all WORF communications activities.

Format

WORF_STOP ()

Operation

Stops all WORF communications.

Deletes all IEEE 802.3 channels that were opened by WORF_INIT. Channels that were open prior to the call to WORF_INIT are not closed.

Disconnects from the Software Bus.

Deletes all Data Source Lists, Database Lists, and all internal structures.

If dump output is enabled (configuration parameter `dynamic.dump_output`), WORF_STOP dumps all the WORF internal structures to file WORF_DUMP.DMP. Defining logical name "WORF_DUMP" overrides the default location and name of the file.

This function is executed automatically when a WORF user process exits. However, this function may be called to stop WORF without terminating the process.

This function MUST NOT be called while at AST level. No error message can be returned to the user in this case, but the I/O may not be correctly cancelled, and may cause an access violation when ASTs are re-enabled.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

`SS$_NORMAL`	Normal successful completion.

Error:

`WORF_CORRUPT`	Internal data structures were corrupted
`WORF_NOINIT`	WORF layer was not initialized.

Notes:

WORF_SWB_RECONNECT

Description

Connects or reconnects to the Software Bus.

Format

WORF_SWB_RECONNECT (*options*, *interval*)

Arguments

options

Usage: Options bit mask.
Type: Unsigned Longword.
Access: Read only.
Mechanism: By value.

interval

Usage: Time interval, minutes.
Type: Longword.
Access: Read only.
Mechanism: By value.

Automatic re-connection time, in minutes.

Operation

This function is called to modify some of the Software Bus connection configuration. For example, the Software Bus connect size may be insufficient. Refer to DSL Size in the Miscellaneous section.

This function disconnects from the Software Bus (if connected), and then re-connects using the configuration information set by WORF_CONFIG (and optionally modified by the user).

If WORF cannot connect to the SWB using the values set by WORF_CONFIG (and optionally modified by the user), Software Bus communications will not be done.

To deliberately disconnect from the Software Bus, set configuration value `init.num_queues` to zero. Refer to the Initialization and Configuration section, subsection Advanced Information, for more details.

The Software Bus connect name is as specified by WORF_CONFIG; if zero length, uses the Process ID in hexadecimal.

If CRISP is not installed on the local machine, WORF_SWB_RECONNECT will not connect to the Software Bus.

If the connection to the Software Bus fails for any reason and ***interval*** is non-zero, an AST is scheduled to run every ***interval*** minutes to re-attempt the connection. This can be used by a client to detect when CRISP has been started on the local machine. When the connection is successful, `init.condx` will be non-zero, and then local data can be accessed.

None of the ***option*** bits are defined.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Failure status from the following.

- CSP$TRNLNM
- LIB$FIND_IMAGE_SYMBOL
- LIB$GETJPI
- SWB$CONNECT
- SWB$DISCONNECT
- SYS$SETIMR

Condition values returned are as follows.

Success:

`SS$_NORMAL`	Normal successful completion.

Warning:

`WORF_CRSTOP`	Cannot connect to the Software Bus because CRISP is stopped
`WORF_SWBCONFIG`	User's configuration prevents connecting to the Software Bus.

Error:

`WORF_CORRUPT`	Internal data structures were corrupted
`WORF_INPROGRESS`	WORF I/O in progress
`WORF_NOINIT`	WORF layer was not initialized
`WORF_SWBNOTAVAIL`	The Software Bus is not available.

Example

```
      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 CONFIG


C     (Assume that DSL's have been built already)

C     Get address of configuration information
      config = worf_config (%val (0))

C     Check configuration against requirements
C     (Pass address of configuration area to subroutine)
      call check_swb (%val (config))

      end



      SUBROUTINE CHECK_SWB (CONFIG)

      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 RETSTAT
      INTEGER*4 PAGES
      INTEGER*4 DSL_ID

      RECORD /WORF_CONFIG/ CONFIG

C     Get maximum size of Software Bus message required
      retstat = worf_swb_size (pages, dsl_id)
      if (.not.retstat) call exit (retstat)

      if (pages .gt. config.init.totalsize - config.init.staticsize) then
         print *, 'DSL ID number', dsl_id, ' requires', pages, ' pages'

C        Add required size to user's static size
         config.init.totalsize = pages + config.init.staticsize

C        Disconnect, then re-connect to Software Bus
         retstat = worf_swb_reconnect (%val (0), %val (0))
         if (.not.retstat) call exit (retstat)
      end if

      return
      end
```

Notes:

WORF_SWB_SIZE

Description

Computes the longest Software Bus message required.

Format

WORF_SWB_SIZE ([*pages*], [*id*])

Arguments

pages

Usage: Required pages.
Type: Unsigned longword.
Access: Write only.
Mechanism: By reference.

The Software Bus connect size required, in 512-byte pages. This argument is optional.

id

Usage: Identification handle.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The handle of the Data Source List with the longest Software Bus message. This argument is optional.

Operation

Scans through all Data Source Lists. Computes the longest Software Bus message necessary to support the current Data Source Lists.

The response sent by the Database Access Server (DBASRV) cannot exceed the size with which WORF connected to the Software Bus. If an attempt is made to create a DSL that is longer, an erroneous timeout condition will occur on each WORF_DSL_READ.

A similar condition can occur on WORF_DSL_WRITE, which returns a status of WORF_LIST2BIG.

Refer to DSL Size in the Miscellaneous section for more information.

After calling WORF_SWB_SIZE, the user can either reduce the size of the Data Source List or can modify the connection to the Software Bus using WORF_SWB_RECONNECT.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized.

Example

```
      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 CONFIG


C     Get address of configuration information
      config = worf_config (%val (0))

C     (Assume that DSL's have been built already)

C     Check configuration against requirements
C     (Pass address of configuration area to subroutine)
      call check_swb (%val (config))

      end



      SUBROUTINE CHECK_SWB (CONFIG)

      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 RETSTAT
      INTEGER*4 PAGES
      INTEGER*4 DSL_ID

      RECORD /WORF_CONFIG/ CONFIG

C     Get maximum size of Software Bus message required
      retstat = worf_swb_size (pages, dsl_id)
      if (.not.retstat) call exit (retstat)

      if (pages .gt. config.init.totalsize - config.init.staticsize) then
         print *, 'DSL ID number', dsl_id, ' requires', pages, ' pages'
```

Example (cont)

```
C        Add required size to user's static size
         config.init.totalsize = pages + config.init.staticsize

C        Disconnect, then re-connect to Software Bus
         retstat = worf_swb_reconnect (%val (0), %val (0))
         if (.not.retstat) call exit (retstat)
      end if

      return
      end
```

Notes:

WORF_TIMEOUT_RECOVERY

Description

Attempts to recover from node timeouts.

Format

WORF_TIMEOUT_RECOVERY ()

Operation

Goes through list of timed-out nodes and attempts to re-establish communications with each one.

A message is sent to each timed-out node, and this function returns without waiting for a response. Status of nodes may be checked with WORF_NL_GET_NODE_STATS_LIST.

⚠ CAUTION

Excessive use of this function may cause high levels of multicast traffic on an IEEE 802.3 network. One multicast message is sent for each node to be located.

WORF can have automatic timeout recovery (refer to the Timeout Recovery section). However, some users may want to disable the automatic cycle and cause timeout recoveries upon command.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_NOINIT	WORF layer was not initialized.

WORF_TIMEOUT_RECOVERY_CHECK

Description

Checks to determine if timeout recovery is in progress

Format

WORF_TIMEOUT_RECOVERY_CHECK ()

Operation

Timeout recovery can occur automatically or when the user calls the function WORF_TIMEOUT_RECOVERY. In either case, the recovery of timed-out nodes proceeds asynchronously.

This function indicates whether or not timeout recovery is being performed.

Automatic timeout recovery can be disabled by setting configuration value `dynamic.recovery_time` to zero. If this is done after an automatic timeout recovery cycle begins, the cycle runs to completion.

AST re-entrant: This routine can be called from either AST level or normal user mode.

Returns

Usage: Status value.
Type: Longword.
Access: Write only.
Mechanism: By value.

Status Values Returned

0 = Timeout recovery is not in progress

1 = Timeout recovery is in progress

Example

```
#include worf_def_user_c

main ()
{
    long dsl_id;
    long symbol_handle;
    long systat;


    SYMBOL_RECORD record;

    NODE_STATISTICS node;

    DATABASE_STATISTICS db;


    /* set up situation */
    systat = worf_init (0);
    systat = worf_dsl_create (CRISP_REAL_TIME, &dsl_id);
    systat = worf_dsl_add_symbol (dsl_id, &record, &symbol_handle, 0, NULL);
    systat = worf_dsl_resolve (dsl_id);

    /* get statistics */
    systat = worf_dsl_get_node_and_db (dsl_id, symbol_handle,
        &node, &db);

    /* recover from timeout */
    if ((node.status & WORF_NODE_M_TMO_CASRV) ||
        ((node.status & WORF_NODE_M_TMO_DBASRV))
    {
        worf_timeout_recovery ();

        while (worf_timeout_recovery_check ())
        {
            lib$wait (&1.0); /* wait slowly for timeout recovery */
        }
    }
}
```

WORF_TIME_HS_TO_VMS

Description

Converts a HISTORIAN time to a VMS quadword time.

Format

WORF_TIME_HS_TO_VMS (*hist, vms*)

Arguments

hist

Usage: HISTORIAN time.
Type: Longword.
Access: Read only.
Mechanism: By reference.

The address of the HISTORIAN time to be converted.

vms

Usage: VMS time.
Type: Quadword.
Access: Write only.
Mechanism: By reference.

The address of a quadword to receive the time.

Operation

Converts a HISTORIAN time into a VMS-usable quadword time.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Notes:

WORF_TIME_VMS_TO_HS

Description

Converts a VMS quadword time to a HISTORIAN time.

Format

WORF_TIME_VMS_TO_HS (*vms, hist*)

Arguments

vms

Usage: VMS time.
Type: Quadword.
Access: Read only.
Mechanism: By reference.

The address of a quadword holding the time to be converted.

hist

Usage: HISTORIAN time.
Type: Longword.
Access: Write only.
Mechanism: By reference.

The address of the longword to receive the HISTORIAN time.

Operation

Converts a VMS quadword time into a HISTORIAN time.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Notes:

WORF_TRANSLATE_ALIAS

Description

Translates an "alias" node name into an "actual" node name as configured from the WORF_ALIAS file.

Format

WORF_TRANSLATE_ALIAS (*alias_node_dx_ptr*, *actual_node_dx_ptr*)

Arguments

alias_node_dx_ptr

Usage: Alias node name
Type: Character string
Access: Read only
Mechanism: By descriptor -- fixed-length string descriptor

The alias node name to be translated.

actual_node_dx_ptr

Usage: Actual node name
Type: Character string
Access: Read/Write
Mechanism: By descriptor -- fixed-length string descriptor

The actual node name of the alias node name. This descriptor must be initialized and sized to point to a user's string buffer prior to entering this call. This call loads the user's string buffer with the translated node name and updates the descriptor to reflect the length of the actual node name.

Operation

This function searches through the alias list, as configured from the WORF_ALIAS file, for a matching ALIAS node name entry. Upon finding a match, the translated node name is copied into the ACTUAL node name string pointed to by the ACTUAL descriptor. The ACTUAL descriptor is updated to reflect the length of the actual node name.

If there is no match, the ALIAS node name is copied to the ACTUAL node name using the respective descriptors.

An alias node name of "0" translates to the name of the node on which the caller is running.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_NOINIT	WORF layer was not initialized.
WORF_TRUNC	Translated alias node name does not fit into the actual node name output string.

WORF_TRIM_BUFFERS

Description

Trims all WORF internal buffers.

Format

WORF_TRIM_BUFFERS ()

Operation

WORF allocates memory for various internal buffers. WORF never decreases the size of these buffers during normal operation. Some buffers may be used only a few times and then never used again.

This function frees ALL the internal buffers that can be freed. WORF then allocates new buffers as required.

NOTE

While this function reduces the memory requirements for WORF, frequent use may result in performance degradation. This function could be called after all Data Source Lists are resolved. However, calling this function before each call to WORF_DSL_READ, for example, would cause buffers to be continually freed and then allocated.

This function MUST NOT be called from an AST routine, and should not be called while I/O is in progress.

Returns

Usage: Condition Code.
Type: Unsigned Longword.
Access: Write only.
Mechanism: By value.

Condition Values Returned

Condition values returned are as follows.

Success:

SS$_NORMAL	Normal successful completion.

Error:

WORF_CORRUPT	Internal data structures were corrupted
WORF_NOINIT	WORF layer was not initialized
WORF_INPROGRESS	WORF I/O in progress.

Notes:

WORF_VERSION_NUMBER

Description

Returns the version number of WORF.

Format

WORF_VERSION_NUMBER ()

Operation

Returns the WORF version number.

When a significant change is made to WORF, the version number will be increased.

For a list of version numbers that can be returned, refer to the Compatibility and Capabilities Matrix section of this manual.

Negative values indicate that WORF has been compiled with DEBUG. Production users of WORF should avoid DEBUG versions of WORF.

Returns

Usage: Version number.
Type: Longword.
Access: Write only.
Mechanism: By value.

Example

```
#include stdio
#include stdlib
#include worf_def_user_c


long version;

version = worf_version_number ();

if (version < 0)
    printf ("**** Warning:  WORF debug version, number %d\n", -version);
else
    printf ("WORF production version, number %d\n", version);
```

Notes:

Appendix A - Glossary

AST	Asynchronous System Trap. The VMS method of implementing software interrupts.
BIOLM quota	The number of buffered I/O requests that a process can have outstanding.
BYTLM quota	The amount of non-paged pool that a process can use for I/O buffers. Each read-ahead buffer used by WORF consumes 1648 bytes of BYTLM.
CASRV	CRISP Access Server. Each CRISP system has a CASRV. Non-CRISP systems may have a CASRV to service Historian and SPC file requests.
DBASRV	Database Access Server. Each CRISP system has a DBASRV.
DBL	Database List. Associated with a DSL.
DSL	Data Source List. A list of data items.
Ethernet	Refer to IEEE 802.3.
Handle	A unique identification of an item, supplied by WORF.
HISTORIAN data	Historical data is seconds to years old. Stored in files known as point files. Accessed through a CASRV.
IEEE 802.3	A communications medium enabling processes on different systems to exchange data. Available on systems running DECnet.
Node Name	The name of a system. System-table logical name SYS$NODE is translated to determine the name of the node.
NULL	A pointer to nowhere. This is typically used in C to indicate that an optional argument is not being used. In Fortran, this is a zero address.
Real-time data	Data from a CRISP database. Accessed through a DBASRV.
Software Bus	A communications medium available on a CRISP system enabling processes on a system to communicate with each other. Refer to the *Software Bus/DBA Reference Manual*.
SPC data	Data related to Statistical Process Control. Looks like Historian data, and is accessed through a CASRV.
Trend data	Recent history data from a real-time data point. Accessed through a DBASRV. A CRISP system may or may not support trends.
WORF	Functions to access CRISP data.

Notes:

Appendix B - Example C

Example

The following sample C program provides an example of the use of WORF communications. In this example, the descriptive notes are on the facing page of this section.

The sample program prompts the user for a node name. The program then retrieves the CRISP time string from that node every 5 seconds.

This sample program is installed as file SYS$EXAMPLES:WORF_EXAMPLE_C.C (non-CRISP system) or file CRISP$HLP:WORF_EXAMPLE_C.C (CRISP systems).

(Continued on next page.)

Example (code)

```
#include stdio                                /* Standard C I/O */

#include descripdef_user_c                    /* descriptors */
#include macrodef_user                        /* FAILURE macro */

#include worf_def_user_c                      /* WORF constants and structures */

globalvalue const int SS$_NORMAL;

/* prototypes for functions declared in this module */
static long check_list (long *dsl_id_ptr, long *symbol_handle_ptr,
    unsigned long *argument_ptr, SYMBOL_RECORD *record_ptr);


main ()

{
    long retstat;
    long time_interval[2];
    long dsl_handle;
    long symbol_handle;

    struct {
        unsigned short  length;
        char            buffer[100];
    } buff1;

    char symbol_name[512];
    char node_name[256];

    SYMBOL_RECORD rec;

    DSC$DESCRIPTOR symbol_dx;

    static const $DESCRIPTOR (time_dx, "0 0:0:05");


    printf ("WORF_EXAMPLE_C -- Get CRISP/32 time from a single node\n\n");

    /* schedule a recurring wakeup */
    sys$bintim (&time_dx, &time_interval[0]);                              (1)
    sys$schdwk (0, 0, &time_interval[0], &time_interval[0]);


    retstat = worf_init (0);
    if (FAILURE (retstat))                                                 (2)
      exit (retstat);

    retstat = worf_dsl_create (CRISP_REAL_TIME, &dsl_handle);              (3)
    if (FAILURE (retstat))
      exit (retstat);
```

Example (description)

① Request that the operating system send a wake up signal to the process every 5 seconds.

② Initialize WORF. All configuration information values are default values.

③ Create a Data Source List.

Example (code) (cont)

```
    printf ("Node name (no colons, please):  ");                                    (4)
    gets (&node_name[0]);

    if (strlen (node_name) > 0)
    {
      strcpy (symbol_name, node_name);
      strcat (symbol_name, "::crisp:csp_s_time");                                   (5)
    }
    else
      strcpy (symbol_name, "crisp:csp_s_time");

    INIT_DX_PTR (&symbol_dx, &symbol_name[0], strlen (symbol_name));                (6)
    retstat = worf_dsl_parse (dsl_handle, &symbol_dx, &rec);
    if (FAILURE (retstat))
      exit (retstat);

    rec.rt.record_number = 0;
    rec.rt.transfer_count = 1;
    INIT_DX_PTR (&rec.rt.buffer_dx, (char *)&buff1, sizeof (buff1));                (7)
    retstat = worf_dsl_add_symbol (dsl_handle, &rec, &symbol_handle, 0, NULL);      (8)
    if (FAILURE (retstat))
      exit (retstat);

    retstat = worf_dsl_resolve (dsl_handle);                                        (9)
    if (FAILURE (retstat))
    {
      worf_dsl_traverse (dsl_handle, 0, &check_list);                               (10)
      exit (retstat);
    }

    while (1)                                                                       (11)
    {
      /* check for exit message */
      retstat = worf_check_for_exit ();                                             (11)
      if (retstat == WORF_EXITMSG)
          exit (SS$_NORMAL);
      else if (FAILURE (retstat))
          exit (retstat);

      retstat = worf_dsl_read (dsl_handle);                                         (12)
        if (FAILURE (retstat))
      {
          worf_dsl_traverse (dsl_handle, 1, &check_list);                           (10)
          exit (retstat);
      }

      /* null-terminate the string */
      buff1.buffer[buff1.length] = '\0';

      printf ("%s\n", &buff1.buffer[0]);                                            (13)

      /* hibernate */
      sys$hiber ();                                                                 (14)
    }
}
```

Example (description) (cont)

④ Ask the user for a CRISP node name.

⑤ Build the name of the symbol to be accessed.

⑥ Parse the symbol name into node name, database name, etc.

⑦ Point to a buffer for the data.
⑧ Add the symbol to the DSL.

⑨ Resolve the Data Source List.

⑩ On error, display status information for each Data Source List Element.

⑪ Loop until an exit message has been received.

⑪ Loop until an exit message has been received.

⑫ Read the current values of the symbol in the Data Source List.

⑩ On error, display status information for each Data Source List Element.

⑬ Display data.

⑭ Wait until awakened by the operating system or by an exit message.

Example (code) (cont)

```
static long check_list (long *dsl_id_ptr, long *symbol_handle_ptr,
    unsigned long *argument_ptr, SYMBOL_RECORD *record_ptr)                    (15)

{
    long retstat;
    long length;

    char string[512];
    char status[512];

    DSC$DESCRIPTOR string_dx;


    if (*argument_ptr == 0)
      printf ("Error ocurred during resolving\n");

    /** build name of symbol **/

    /* node name */
    strncpy (&string[0], record_ptr->node_name_dx.dsc$a_pointer,               (16)
      record_ptr->node_name_dx.dsc$w_length);
    length = record_ptr->node_name_dx.dsc$w_length;

    /* add in the double-colons */
    strncpy (&string[length], "::", 2);
    length += 2;

    /* database name */
    strncpy (&string[length], record_ptr->rt.database_name_dx.dsc$a_pointer,
      record_ptr->rt.database_name_dx.dsc$w_length);
    length += record_ptr->rt.database_name_dx.dsc$w_length;

    /* add in the colon */
    strncpy (&string[length], ":", 1);
    length += 1;

    /* symbol name */
    strncpy (&string[length], record_ptr->rt.symbol_name_dx.dsc$a_pointer,
      record_ptr->rt.symbol_name_dx.dsc$w_length);
    length += record_ptr->rt.symbol_name_dx.dsc$w_length;

    /* null-terminate */
    string[length] = '\0';

    /* print symbol name */
    printf ("%s\n", string);
```

Example (description) (cont)

⑮ Function to display status information for a symbol.

⑯ Build the complete name of the symbol.

Example (code) (cont)

```
    /* dump status bits */
    do                                                                    (17)
    {

      INIT_DX_PTR (&string_dx, &status[0], sizeof (status));
      retstat = worf_dsl_get_status_string_num (&record_ptr->status,
          &record_ptr->error_code, &string_dx, &string_dx.dsc$w_length);
      if (retstat != WORF_CHECK)
          break;
      status[string_dx.dsc$w_length] = '\0';

      printf ("  %s\n", status);
    }
    while (record_ptr->status != 0 || record_ptr->error_code != 0);

    /* zero return status keeps DSL_TRAVERSE going */
    return 0;
}
```

Example (description) (cont)

⑰ Display all status strings.

Notes:

Appendix C - Example Fortran

Example

The following sample Fortran program provides an example of the use of WORF communications. In this example, the descriptive notes are on the facing page of this section.

The sample program prompts the user for a node name. The program then retrieves the CRISP time string from that node every 5 seconds.

This sample program is installed as file SYS$EXAMPLES:WORF_EXAMPLE_FOR.FOR (non-CRISP system) or file CRISP$HLP:WORF_EXAMPLE_FOR.FOR (CRISP systems).

(Continued on next page.)

Example (code)

```
      PROGRAM WORF_EXAMPLE_FOR

      INCLUDE '(WORF_DEF_USER_FOR)'

      INTEGER*4 RETSTAT
      INTEGER*4 TIME_INTERVAL(2)
      INTEGER*4 POS
      INTEGER*4 DSL_HANDLE
      INTEGER*4 SYMBOL_HANDLE

      STRUCTURE /STRING/
        INTEGER*2     LENGTH
        CHARACTER*100 BUFFER
      END STRUCTURE

      RECORD /STRING/ BUFF1

      CHARACTER*512 SYMBOL_NAME
      CHARACTER*256 NODE_NAME

      RECORD /SYMBOL_RECORD/ REC

      EXTERNAL CHECK_LIST


      print *, 'WORF_EXAMPLE_FOR -- Get CRISP/32 time from a single node'
      print *, ' '
c     Schedule a recurring wakeup
      call sys$bintim ('0 0:0:05', time_interval)
      call sys$schdwk (%val (0), %val (0), time_interval, time_interval)      (1)


      retstat = worf_init (%val (0))
      if (.not.retstat) call exit (retstat)                                   (2)

      retstat = worf_dsl_create (%val (CRISP_REAL_TIME), dsl_handle)          (3)
      if (.not.retstat) call exit (retstat)

      print 5
5     format ('$Node name (no colons, please):  ')                            (4)

      read 10, node_name
10    format (a)

      pos = index (node_name, ' ')
      if (pos .gt. 1) then
          symbol_name = node_name (1:pos-1) // '::crisp:csp_s_time'           (5)
      else
          symbol_name = 'crisp:csp_s_time'
      end if

      retstat = worf_dsl_parse (%val (dsl_handle), symbol_name, rec)          (6)
      if (.not.retstat) call exit (retstat)
```

Example (description)

① Request that the operating system send a wake up signal to the process every 5 seconds.

② Initialize WORF. All configuration information values are default values.

③ Create a Data Source List.

④ Ask the user for a CRISP node name.

⑤ Build the name of the symbol to be accessed.

⑥ Parse the symbol name into node name, database name, etc.

Example (code) (cont)

```
      rec.rt.record_number = 0
      rec.rt.transfer_count = 1
      rec.rt.buffer_dx.dsc$w_length = sizeof (buff1)
      rec.rt.buffer_dx.dsc$a_pointer = %loc (buff1)                    (7)
      retstat = worf_dsl_add_symbol (%val (dsl_handle), rec,           (8)
     1     symbol_handle, %val (0), %val (0))
      if (.not.retstat) call exit (retstat)

      retstat = worf_dsl_resolve (%val (dsl_handle))                   (9)
      if (.not.retstat) then
          call worf_dsl_traverse (%val (dsl_handle), %val (0), check_list)   (10)
          call exit (retstat)
      end if

      do while (.true.)
C         check for exit message                                       (11)
          retstat = worf_check_for_exit ()
          if (retstat .eq. %loc (WORF_EXITMSG)) stop
          if (.not.retstat) call exit (retstat)

          retstat = worf_dsl_read (%val (dsl_handle))                  (12)
          if (.not.retstat) then
              call worf_dsl_traverse (%val (dsl_handle), %val (0),
     1            check_list)
              call exit (retstat)
          end if

          print *, buff1.buffer (1:buff1.length)                       (13)

c         Hibernate
          call sys$hiber                                               (14)

      end do                                                           (11)

      end

      FUNCTION check_list (dsl_id, symbol_handle, argument, record)    (15)

      INCLUDE '(WORF_DEF_USER_FOR)'

c     ARGUMENTS TO FUNCTION
      INTEGER*4 DSL_ID
      INTEGER*4 SYMBOL_HANDLE
      INTEGER*4 ARGUMENT
      RECORD /SYMBOL_RECORD/ RECORD

c     LOCAL VARIABLES
      INTEGER*4 RETSTAT
      INTEGER*2 LENGTH
      CHARACTER*100 STRING

      if (argument .eq. 0) print *, 'Error ocurred during resolving'

c     Print name of symbol
      call sys$fao ('!AS::!AS:!AS', length, string,                    (16)
     1    record.node_name_dx, record.rt.database_name_dx,
     2    record.rt.symbol_name_dx)
      print *, string (1:length)                                       (17)
```

Example (description) (cont)

⑦ Point to a buffer for the data.
⑧ Add the symbol to the DSL.

⑨ Resolve the Data Source List.

⑩ On error, display status information for each Data Source List Element.

⑪ Loop until an exit message has been received.

⑫ Read the current values of the symbol in the Data Source List.

⑬ Display data.

⑭ Wait until awakened by the operating system or by an exit message.
⑪ Loop until an exit message has been received.

⑮ Function to display status information for a symbol.

⑯ Build the complete name of the symbol.

⑰ Display all status strings.

Example (code) (cont)

```
c     Dump status bits
      do while (.true.)
          retstat = worf_dsl_get_status_string_num (record.status,
      1       record.error_code, string, length)
          if (retstat .ne. %loc (WORF_CHECK)) then
c             Zero return status keeps DSL_TRAVERSE going
              check_list = 0
              return
          end if

          print *, '   ' // string (1:length)
          if (record.status .eg. 0 .AND. record.error_code .eg. 0) then
c             Zero return status keeps DSL_TRAVERSE going
              check_list = 0
              return
          end if
      end do

      END
```

Appendix D - Status Values

Message Format

There are several values unique to WORF that can be returned by WORF functions. Other values are defined at the system level, and in the VMS System Messages and Recovery Procedures manual.

The format of the messages is as follows.

%WORF-sevr-error description

%WORF The facility name for WORF. This name begins all messages and is useful in identifying WORF messages interspersed with others.

sevr This one-letter code refers to the severity of the message. The levels of severity are as follows.

I-(Information) WORF continues executing normally. This condition might require user intervention.

W-(Warning) WORF continues to execute, but a condition was encountered that may not be properly handled. User intervention is needed.

E-(Error) WORF encountered a condition that it cannot correct. WORF continues to execute, but probably will not execute properly. User intervention is necessary.

F-(Fatal) WORF encountered a condition from which no recovery is possible. System management assistance will be needed before WORF can operate.

Return Status Values

The following lists the values that can be returned by WORF. It explains the cause of the message, and defines the user action that should be taken to correct the problem.

ABORTED I/O operation aborted at user's request.

Message Type: Warning.

Explanation: The user called WORF_DSL_ABORT_IO.

User Action: The need to call this function should be examined carefully; aborting WORF I/O is not a normal use of WORF.

(Continued on next page.)

Return Status Values (cont)

ALRINIT WORF Communications layer already initialized.

Message Type: Error.

Explanation: WORF_INIT was called twice.

User Action: Either call WORF_STOP before calling WORF_INIT again, or eliminate the redundant call.

ASTDIS ASTs are disabled.

Message Type: Error.

Explanation: ASTs were disabled, but IEEE 802.3 network channels were open.

User Action: Do not call WORF routines with ASTs disabled.

BADALIAS Format of alias table was not valid.

Message Type: Fatal.

Explanation: The format of the alias table was not valid.

User Action: Fix the table. Refer to the Miscellaneous section for more information.

BADHANDLE Symbol handle was not valid.

Message Type: Error.

Explanation: A WORF function was passed a symbol handle that was not assigned by WORF, is a handle from a DSL other than the DSL identification passed to the function, or has been deleted.

User Action: Correct the call to the function.

BADID Data Source List identification was not valid.

Message Type: Error.

Explanation: A WORF function was passed a DSL identification that was not assigned by WORF, or the DSL has been deleted.

User Action: Correct the call to the function.

(Continued on next page.)

Return Status Values (cont)

BADOPS Invalid mixture of operations on Historian/SPC Data Source List

Message Type: Error.

Explanation: A call was made to WORF_DSL_READ with a symbol whose operation type is HS_UPDATE_AT_TIMES, or a call was made to WORF_DSL_WRITE with a symbol whose operation type was not HS_UPDATE_AT_TIMES.

User Action: Correct the call to the function.

BOUNDS Data transfer attempted outside bounds of array or database.

Message Type: Error.

Explanation: An attempt was made to modify a real-time symbol so that its transfer count or subscript is not valid.

User Action: Fix the call to WORF_DSL_MODIFY_SYMBOL.

CANTPROC Server reports that it cannot process request.

Message Type: Error.

Explanation: The CRISP Access Server cannot process the request. This can be because the request causes the server subprocess to fail, or because of a fault in how the request is specified.

User Action: Check the request for validity. The CRISP$TT console on the CRISP/32 system will have a message giving more detail on this problem.

CHANGED Change(s) accepted.

Message Type: Information.

Explanation: The change to the WORF_DSL_M_INVALID bit was accepted.

User Action: None.

(Continued on next page.)

Return Status Values (cont)

CHECK Check Data Source List or Node List for entries with problems.

Message Type: Warning.

Explanation: One or more symbols in the Data Source List are marked as invalid, overrun their user buffer, or are marked as having an invalid subscript. When calling WORF_DSL_RESOLVE, this status indicates that one or more nodes were not located.

User Action: Examine Data Source List. Take appropriate action depending on application requirements.

CORRUPT Internal data structures are corrupt.

Message Type: Error.

Explanation: An internal consistency check failed.

User Action: Exit WORF soon. Check program to determine if the internal structures of WORF are being modified by user code.

CRSTOP Cannot connect to Software Bus because CRISP is stopped.

Message Type: Warning.

Explanation: CRISP is not running so WORF cannot connect to the Software Bus.

User Action: Attempt to connect later, if desired, by calling WORF_SWB_RECONNECT.

DBLEXISTS Database List entry already exists.

Message Type: Information.

Explanation: The database name specified was already in the Database List. The signature value specified by the user is used.

User Action: None.

(Continued on next page.)

Return Status Values (cont)

DSLEMPTY Data Source List has no entries.

Message Type: Information.

Explanation: An attempt was made to read, write, or resolve a Data Source List with no entries.

User Action: Check creation of Data Source List elements.

EXITMSG Exit message received.

Message Type: Information.

Explanation: An exit message was received from the local CRISP/32 system.

User Action: As a minimum, stop WORF. It is recommended that the user's process exit after saving status information.

FILNOTFOUND File Not Found.

Message Type: Warning.

Explanation: The file(s) matching the requested file specification can not be found.

Signaled By: WORF_HS_GET_FILE_LIST.

IGNORED Change(s) ignored.

Message Type: Information.

Explanation: The change to the WORF_DSL_M_INVALID bit was ignored.

User Action: If the purpose was to clear the bit, check the symbol's status for other reasons that the symbol must remain invalid. If the purpose was to set the bit, the symbol is already marked as invalid.

(Continued on next page.)

Return Status Values (cont)

INPROGRESS WORF I/O in progress.

Message Type: Error.

Explanation: An attempt was made to perform an I/O operation, but an I/O operation was already in progress.

User Action: Correct the call to the WORF function. This could indicate an attempt to perform I/O at AST level, which is not supported.

INVALID Data Source List entry has invalid information.

Message Type: Error.

Explanation: An attempt was made to add or modify a symbol that does not have valid information.

User Action: Correct the call to the function.

INVTYPE Type number of Data Source List was not valid.

Message Type: Error.

Explanation: An attempt was made to create a Data Source List for a DSL type that is not supported.

User Action: Pass the correct type value to the function.

LIMITS Attempt to ramp beyond limits, clamped at limit.

Message Type: Warning.

Explanation: The server ramped the symbol as far as possible in the desired direction, but the specified limits were reached.

User Action: None.

LIST2BIG Data Source List too large.

Message Type: Error.

Explanation: A message to be sent to a server was larger than the communications medium can handle. There is one message per real-time node/database pair.

User Action: Reduce size of Data Source List.

Return Status Values (cont)

MAXDSLS Maximum number of Data Source Lists on one node exceeded.

Message Type: Error.

Explanation: Each server supports a maximum number of 'client contexts'. One context is used for each combination of Data Source List and tracking option. The limit is 65535 contexts.

User Action: Eliminate unneeded Data Source Lists, or combine DSLs.

MISSING Missing information passed to WORF layer.

Message Type: Error.

Explanation: An essential piece of information was not passed to WORF.

User Action: Correct the call to the function.

NOCOMM No communications paths available for communications layer.

Message Type: Fatal.

Explanation: WORF cannot use either the Software Bus or an IEEE 802.3 network.

User Action: Make at least one communications path available. Assistance may be required from the system manager.

NOCONFIG Unable to configure WORF.

Message Type: Fatal.

Explanation: Memory for internal WORF structures could not be allocated.

User Action: Check the resources available to the process.

(Continued on next page.)

Return Status Values (cont)

NOINIT WORF layer not initialized.

Message Type: Error.

Explanation: A call was made to a function, but WORF_INIT had not been called.

User Action: Correct the call to the function to ensure that WORF is initialized first.

NONETWORK Unable to create any IEEE 802.3 network channels.

Message Type: Warning.

Explanation: WORF was unable to create any channels to any IEEE 802.3 devices. The Software Bus, however, will be used.

User Action: The process may have insufficient BYTLM quota. In other cases, assistance of the system manager will be required.

NOTQUEUED Request not queued to server yet.

Message Type: Error.

Explanation: The CRISP Access Server has a finite input queue of requests. This status indicates that the queue is full at the moment.

User Action: Re-try the operation at a later time.

NULLPTR Null pointer passed for required argument.

Message Type: Warning.

Explanation: A NULL pointer was passed for a required parameter. No operation was performed.

User Action: Correct the call to the function.

(Continued on next page.)

Return Status Values (cont)

NULLSTR Null string passed to WORF_DSL_PARSE.

Message Type: Error.

Explanation: The pointer to the descriptor passed to function WORF_DSL_PARSE was NULL, had a zero length, or pointed to a NULL location.

User Action: Correct the call to the function.

OVERRUN Overrun of array or buffer.

Message Type: Warning.

Explanation: The size of the data buffer passed to WORF_DSL_SYMBOL_RAMP or WORF_DSL_SYMBOL_WRITE was not the correct size for the symbol being written.

User Action: Correct the call to the function.

PARSERR Error parsing string.

Message Type: Error.

Explanation: The string passed to WORF_DSL_PARSE was not valid. Refer to the Symbol Name Syntax section for more information.

User Action: Correct the call to the function.

QUEUED Request queued at server.

Message Type: Information.

Explanation: The CRISP Access Server can perform a limited number of operations at one time. This status indicates that the request has been added to the input queue of the server, or that the server has begun processing the request.

User Action: Wait for operation to be completed.

(Continued on next page.)

Return Status Values (cont)

RESOLVE Resolving of Data Source List is necessary.

Message Type: Error.

Explanation: The Data Source List must be resolved before reading or writing can occur.

User Action: Call function WORF_DSL_RESOLVE.

SERVER Error code returned from server.

Message Type: Error.

Explanation: The server returned an error status code that was not expected. This can indicate an incompatibility between the versions of WORF and the server.

User Action: If debugging output is enabled, the WORF_SERVERNUM message will give the error code number as returned by the server.

STEALING WORF Communications layer using already open network channels.

Message Type: Warning

Explanation: IEEE 802.3 network channels were already opened before calling WORF_INIT. WORF will use the channels.

User Action: Sharing channels in this manner is not a normal use of WORF, since WORF is unable to specify the number of read-ahead buffers. Also, the user might be using the network channels in synchronous mode, which is incompatible with WORF.

STRING Attempt to ramp a character string.

Message Type: Error.

Explanation: WORF_DSL_SYMBOL_RAMP was called to ramp a string. This operation is not possible.

User Action: Correct the call to the function.

(Continued on next page.)

Return Status Values (cont)

SWBCONFIG Configuration information prevents connecting to the Software Bus.

Message Type: Warning.

Explanation: The configuration information specified values that were incompatible with connecting to the Software Bus.

User Action: If connecting to the Software Bus was intended, change the configuration as appropriate and then call WORF_SWB_RECONNECT.

SWBNOTAVAIL Software Bus not available.

Message Type: Error.

Explanation: WORF could not connect to the Software Bus because the system is not a CRISP system, or because CRISP has not been started at least once.

User Action: Attempt to connect later, if desired, by calling WORF_SWB_RECONNECT.

TRUNC String descriptor was too long.

Message Type: Error.

Explanation: A string was copied into another string. The source string was longer than the space available in the destination string.

User Action: Make the destination string descriptor longer. In the case of functions WORF_DBL_ADD_DB, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, and WORF_RT_GET_SYMBOL_LIST, the user-specified node name or database name had a zero length or was longer than the maximum.

UNIMPL Unimplemented code path.

Message Type: Fatal.

Explanation: An impossible error occurred. This is an internal consistency check, and should not occur.

User Action: Report problem to Square D.

(Continued on next page.)

Return Status Values (cont)

WRONGTYPE Data Source List is not correct type.

Message Type: Error.

Explanation: The function called does not operate on the type of DSL that was specified.

User Action: Correct the call to the function.

Appendix E - Status Messages

Message Format

WORF displays status messages when debugging output is enabled. Refer to the Configuration section for information on enabling and disabling debugging output. The format of the messages is as follows.

%WORF-sevr-error description

%WORF The facility name for WORF. This name begins all messages and is useful in identifying WORF messages interspersed with others.

sevr This one-letter code refers to the severity of the message. The levels of severity are as follows.

I-(Information) WORF executing normally.

W-(Warning) WORF continues to execute, but a condition was encountered that may not be properly handled.

E-(Error) WORF encountered a condition that it cannot correct. WORF continues to execute, but probably will not execute properly.

Signaled Messages

The following lists messages that WORF signals when debugging output is enabled.

ABORT Aborting all I/O.

Message Type: Information.

Explanation: All pending I/O operations are being aborted.

Signaled By: WORF_DSL_ABORT_IO and WORF_STOP.

ALIASOPEN Unable to open alias file.

Message Type: Error.

Explanation: The alias file could not be opened. This message implies that an alias file exists. The second line of the message shows the system condition code.

Signaled By: WORF_INIT.

Signaled Messages (cont)

BADALIASENT Bad alias table entry: "<entry>".

Message Type: Error.

Explanation: The alias table was not valid.

Signaled By: WORF_INIT.

BADNODENAME Node name "<name>" is not valid.

Message Type: Error.

Explanation: The node name was of zero length or longer than six characters.

Signaled By: WORF_INIT.

CANEXH Canceling exit handler.

Message Type: Information.

Explanation: The exit handler established by WORF_INIT is being canceled.

Signaled By: WORF_STOP.

CASRV Error status from the server of the node <node-name> at <time>.

Message Type: Warning.

Explanation: The CRISP Access Server returned an error status value.

Signaled By: WORF_DSL_READ and WORF_DSL_WRITE.

CRSTOP Cannot connect to Software Bus because CRISP is stopped.

Message Type: Warning.

Explanation: WORF is running on a CRISP/32 system, and would have connected to the Software Bus, but CRISP was not running.

Signaled By: WORF_INIT, WORF_SWB_RECONNECT.

(Continued on next page.)

Signaled Messages (cont)

DBASRV Error status from the server of the node <node-name> at <time>.

Message Type: Warning.

Explanation: The Database Access Server returned an error status value.

Signaled By: WORF_DSL_READ, WORF_DSL_RESOLVE, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, and WORF_RT_NEXT_SYMBOL.

DBLOCATED Database <database-name> located on node <node-name> at <time>.

Message Type: Information.

Explanation: The database was located on the node.

Signaled By: WORF_DSL_RESOLVE.

DBL_BUFFERS Freeing Database List buffers, DSL <number>, DBL <number>, Track <number>, node <node-name>.

Message Type: Information.

Explanation: The internal WORF buffers for the specified Data Source List are being deleted.

Signaled By: WORF_TRIM_BUFFERS.

DELALLDSL Deleting all Data Source Lists.

Message Type: Information.

Explanation: All Data Source Lists are being deleted.

Signaled By: WORF_STOP.

(Continued on next page.)

Signaled Messages (cont)

DELCONFIG Deleting all configuration information.

Message Type: Information.

Explanation: The configuration information allocated by WORF_CONFIG is being deleted.

Signaled By: WORF_STOP.

DISCONNECT Disconnecting from Software Bus.

Message Type: Information.

Explanation: WORF is disconnecting from the Software Bus.

Signaled By: WORF_STOP.

DSL_HEADER Deleting Data Source List Header Array.

Message Type: Information.

Explanation: The master index to the Data Source Lists is being deleted.

Signaled By: WORF_STOP.

DUMPING Dumping internal structures.

Message Type: Information.

Explanation: Dumping of the internal structures of WORF was enabled, and the dump is being written.

Signaled By: WORF_STOP and WORF_DUMP.

FREE_EF Freeing event flags.

Message Type: Information.

Explanation: The event flags allocated by WORF_INIT are being freed.

Signaled By: WORF_STOP.

Signaled Messages (cont)

IEEE802ERR Unable to transact with remote server at <time>.

Message Type: Warning.

Explanation: WORF was unable to send a message using the IEEE 802.3 medium.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

LOCATEDNET Located the server <server-type> of the node <node-name> on network <number> at <time>.

Message Type: Information.

Explanation: The server on the specified node was found on the network.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

LOCATEDSWB Located the server <server-type> of the node <node-name> on Software Bus at <time>.

Message Type: Information.

Explanation: The server on the specified node was found on the Software Bus.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

(Continued on next page.)

Signaled Messages (cont)

LOCATNET Locating the server <server-type> of the node <node-name> on network <number> at <time>.

Message Type: Information.

Explanation: The specified server was not located on the Software Bus. The available IEEE 802.3 networks are being searched for the server.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

MSGFROM Message from <name> via <network_type>

Message Type: Information.

Explanation: A message was received by WORF.

Signaled by: WORF_CHECK_FOR_EXIT.

NETAST Error condition in IEEE 802.3 network AST routine.

Message Type: Error.

Explanation: An error occurred during processing of a network message. No mechanism exists for returning the error to the caller, because it occurred in an AST routine.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

(Continued on next page.)

Signaled Messages (cont)

NETAVAIL Network <number> is available, address <address> (<sap>).

Message Type: Information.

Explanation: A channel is now open on the specified IEEE 802.3 network. The address and IEEE 802.3 Service Access Point (SAP) are signaled.

Signaled By: WORF_INIT and WORF_NETWORK_RECONNECT.

NETWORK Closing all network channels.

Message Type: Information.

Explanation: All IEEE 802.3 network channels, except those opened by the user, are being closed.

Signaled By: WORF_STOP and WORF_NETWORK_RECONNECT.

NL_ARRAY Deleting Node List Array.

Message Type: Information.

Explanation: The Node List information is being deleted.

Signaled By: WORF_STOP.

NL_BUFFERS Freeing Node List buffers, entry <number>, node <node-name>.

Message Type: Information.

Explanation: The Node List buffers are being deleted.

Signaled By: WORF_STOP and WORF_TRIM_BUFFERS.

Signaled Messages (cont)

NODENAME System node name <node-name>.

Message Type: Information.

Explanation: The node name of the local system is as signaled.

Signaled By: WORF_INIT.

NOLOCATSWB Unable to locate the server <server-type> of the node <node-name> on Software Bus at <time>.

Message Type: Information.

Explanation: The server on the local machine is not connected to the Software Bus.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

NONODE System logical name SYS$NODE not defined.

Message Type: Warning.

Explanation: SYS$NODE must be defined in the system logical name table. If DECnet is installed on the system, this name will already be defined.

Signaled By: WORF_INIT.

NOSWBRTL No CRISP Software Bus run-time library available.

Message Type: Warning.

Explanation: Before WORF can connect to the Software Bus, the Software Bus RTL must be located. If the system is a CRISP/32 system, contact the system manager -- CRISP may be installed improperly. This message is normal on a non-CRISP system.

Signaled By: WORF_INIT, WORF_SWB_RECONNECT.

Signaled Messages (cont)

RECOVERED

Node <node-name> server <server-type> has recovered at <time>.

Message Type: Information.

Explanation: The server on the specified node has responded to communications after having been marked as timed-out.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

RECOVERING

Recovering from server timeout(s) at <time>.

Message Type: Information.

Explanation: Recovery from node timeouts has begun.

Signaled By: WORF_TIMEOUT_RECOVERY, or can occur spontaneously when automatic timeout recovery is enabled.

SERVERNUM

Error code <number> (<hex-number>) returned from server <server-type> at node <node-name>.

Message Type: Information.

Explanation: The specified server returned a status code that was not recognized. This can indicate an incompatibility between the versions of WORF and the server.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

Signaled Messages (cont)

STRAY Stray AST received.

Message Type: Warning.

Explanation: An IEEE 802.3 message was received that was not expected. Signaled as a second line of a WORF_NETAST message.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

SWBAVAIL Software Bus available, connect name <name>.

Message Type: Information.

Explanation: The Software Bus RTL has been found, CRISP is running, and the initialization parameters have been properly set. WORF will attempt to connect to the Software Bus.

Signaled By: WORF_INIT.

SWBERR Unable to transact with local <server-type> server at <time>.

Message Type: Warning.

Explanation: WORF was unable to send a message to the server using the Software Bus.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

Signaled Messages (cont)

SYMBOL

Error in adding or modifying a symbol in a Data Source List.

Message Type: Warning.

Explanation: An error occurred during adding or modifying a symbol.

Signaled By: WORF_DSL_ADD_SYMBOL and WORF_DSL_MODIFY_SYMBOL.

TIMEOUT

The server <server-type> of the node <node-name> timed out at <time>.

Message Type: Information.

Explanation: The specified server failed to respond within the timeout period, or sent back an error code. The server will be marked as timed out.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

TIME_AST

Disabling timeout recovery AST.

Message Type: Information.

Explanation: Automatic recovery from node timeouts was previously enabled, and has now been disabled.

Signaled By: WORF_STOP.

Signaled Messages (cont)

WRONGNODE <server-type> server sent node name <node-name>, expecting <node-name>.

Message Type: Error.

Explanation: A response of a server to an attempt by WORF to locate a node includes the name of the node. This should always match exactly. WORF checks the response anyway.

Signaled By: WORF_DSL_RESOLVE, WORF_DSL_READ, WORF_DSL_SYMBOL_RAMP, WORF_DSL_SYMBOL_WRITE, WORF_DSL_WRITE, WORF_RT_GET_DB_LIST, WORF_RT_GET_NODE_LIST, WORF_RT_GET_SYMBOL_LIST, WORF_RT_NEXT_SYMBOL, and WORF_TIMEOUT_RECOVERY.

Appendix F - WORF Internal Structure

Internal Structure

To be able to read the dump produced, enable configuration option `dynamic.dump_output`. The user must understand the internal structure of WORF. This information is provided to assist the user and is not to be used to 'reverse engineer' the product.

Figure F-1 shows the primary WORF internal structure.

The key to all the other structures is the 'DSL Header Array Root'. The root points to the Node List and the DSL Header Array.

Each node that WORF communicates with is represented in the Node List. The information in the Node List is shared among all DSLs and is updated by each I/O operation.

The DSL Header Array is an array of pointers that point to individual DSLs.

In Figure F-1, element 2 points to a DSL Header. Each DSL Header points to a linked list of DSL Elements and the Database List.

Each DSL Element points to the associated Database List entry. In the figure, DSL element 0 points to the first DBL entry. DSL elements 1 and 2 both point to the DBL entry 1.

Each Database List entry points to the Node List. In the figure, DBL entry 0 points to Node List entry 0. DBL entries 1 and 2 point to Node List entry 2.

Each Database List entry also points up to three tracking entries. Each DSL entry contains a number that indicates if the symbol is to be always updated, tracked active, or tracked standby. In the figure, DSL element 0 is always updated. DSL elements 1 and 2 are tracked active. Not shown are two other DSL elements that would point to a different Database List entry (because of the active tracking).

(Continued on next page.)

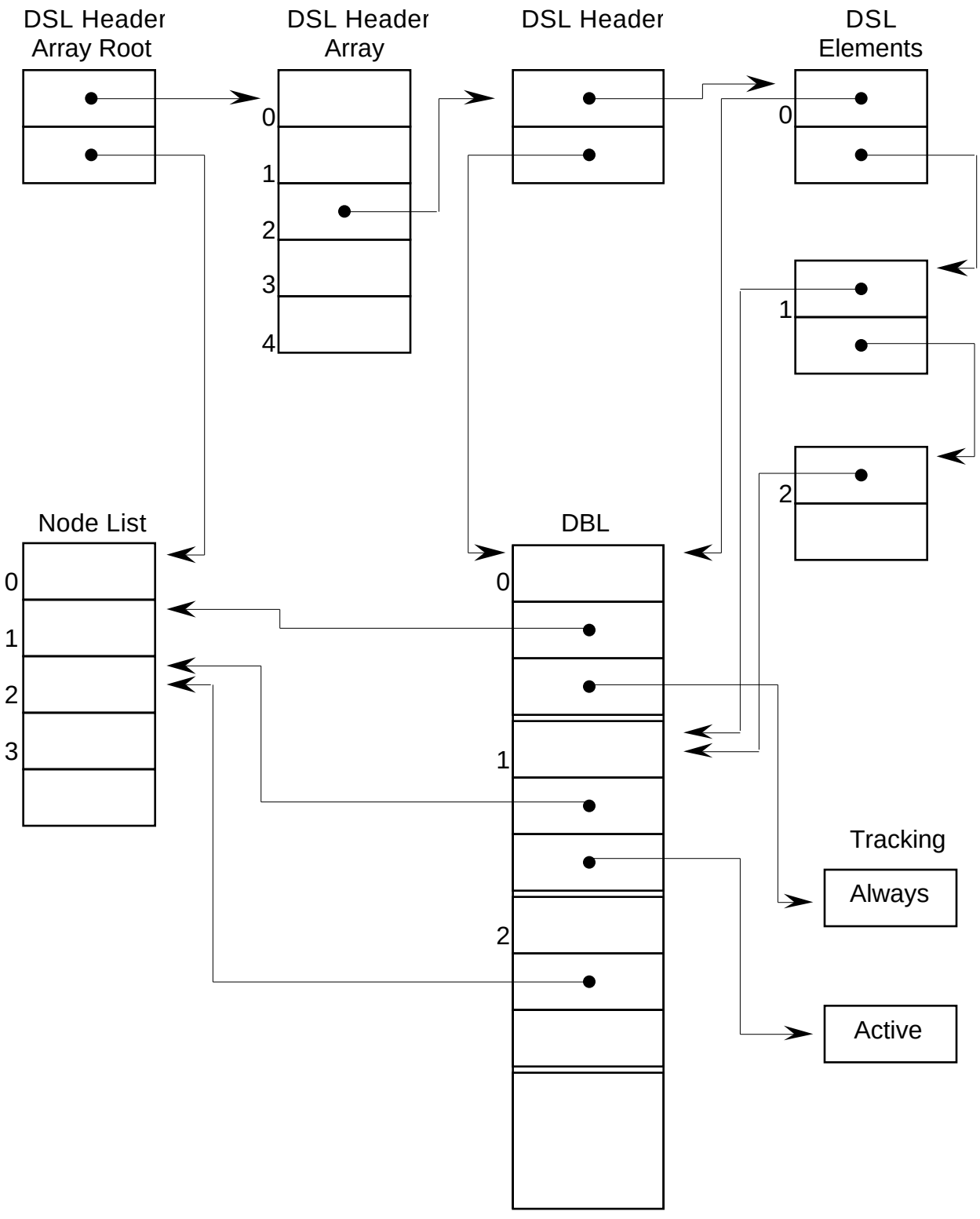

Figure F-1. WORF Internal Structure

Appendix G - Reading a WORF Dump

General

The WORF dump is a snapshot of the status of WORF. Major sections are divided with equal signs (=). Within a major section, sections are divided by minus signs (-) and further divided by dashed minus signs (- -).

Addresses and sizes of all buffers are listed. When a user's buffer is dumped, it is first checked to determine if it can be read. Communications buffer are dumped as the CRISP/32 network header and server-specific header.

WORF uses 'self-pointers' extensively. Each structure contains a pointer to itself, so WORF can detect inadvertent overwriting of memory. These pointers are checked when WORF is running and during the dumping process.

If WORF detects an inconsistency in any information, it prints a message flagged with an asterisk (*).

Dump Structure

A WORF dump is organized as follows.

Static data
Configuration data
Alias table
Node List
  Node list entry 0
  Node list entry 1
  Node list entry 2, etc.
Data Source List Header 0
  DSL element 0
  DSL element 1
  DSL element 2, etc.
  Database List entry 0
    Tracking entry for elements always updated
    Tracking entry for elements tracked in active databases
    Tracking entry for elements tracked in standby databases
  Database List entry 1
    Tracking entry for elements always updated
    Tracking entry for elements tracked in active databases
    Tracking entry for elements tracked in standby databases
  Database List entry 2, etc.
    Tracking entry for elements always updated
    Tracking entry for elements tracked in active databases
    Tracking entry for elements tracked in standby databases
Data Source List Header 1
  DSL elements
  Database List entries
Data Source List Header 2, etc.

Sample Dump

The following dump is an example of WORF reading the value of CRISP:CSP_S_TIME (the CRISP/32 time string) from node ELAB. In this example, the descriptive notes are on the facing page of this section.

WORF is running on node EVAX, using network communications only. WORF did not connect to the Software Bus.

```
=======================================================
DUMP OF WORF INTERNAL STRUCTURES

Dumping at  6-DEC-1990 14:59:44.40

WORF compiled at Dec  5 1990 15:09:20

WORF version = 1

=======================================================

Statically-Allocated Data

WORF_ABORT_IO = 0
WORF_IO_IN_PROGRESS = 0
WORF_MUST_EXIT = 0
WORF_EVENT_FLAG_IEEE802 number = 62, state = 0
WORF_EVENT_FLAG_PAUSE number = 61, state = 0
WORF_NETWORK_COUNT = 1
WORF_PENDING_COUNT = 0
WORF_SEQUENCE_NUMBER = 8                                              (1)
AST's are enabled
WORF_NODE_NAME = "EVAX" (length = 4, address = 0003E968)              (2)
Name of Database Access Server = "DBASRV" (length = 6, address = 0003B502)
Name of CRISP Access Server = "CASRV" (length = 5, address = 0003B50F)

Network    Status
   0       Started    Not Borrowed                                    (3)
   1       Not Started
Timeout recovery timer is not running

=======================================================
```

Sample Dump (cont)

① The total number of messages exchanged with servers.

② The name of the system on which this program is running.

③ IEEE 802.3 network status.

Sample Dump (cont)

```
Configuration Data

Options = 00000000
Communications timeout 1000 milliseconds
Communications retries = 3
Server timeout automatic recovery is disabled
Debugging output is off
SWB connect name will be Process ID
SWB total size = 20  (512-byte pages)
SWB static size = 0  (512-byte pages)
Number of SWB queues = 2
Not connected to the Software Bus
Address of SWB$CONNECT = 00000000
Address of SWB$DIALUP = 00000000
Address of SWB$DISCONNECT = 00000000
Address of SWB$RECEIVEMSG = 00000000
Address of SWB$SENDMSG = 00000000
Address of SWB$SIZEOF_NEXTMSG = 00000000
Configuration self pointer = 0009F650
Configuration self-pointer is okay

=====================================================

Alias Table

No alias table--file not found, or was empty

=====================================================
WORF Root self pointer = 0003ECF8
WORF Root self-pointer is okay

=====================================================

Node List Data

List size = 5 entries
Node List starting address = 000A0D98

Node List entry 0 is used
Node List self pointer = 000A0E9C
Node List self-pointer is okay
Node name = "elab" (length = 4, address = 000A0DA0)                    (4)
Node located
Communications path is IEEE 802
Node address = AA-00-04-00-06-04                                       (5)
CASRV is not timed out on network 0
DBASRV is not timed out on network 0
CASRV is reachable
CASRV timeout count = 0
CASRV capability = 2, CRISP/32 version = "X2.7-0"                      (6)
```

Sample Dump (cont)

```
DBASRV is reachable
DBASRV timeout count = 0
DBASRV capability = 0, CRISP/32 version = ""
```

④ Node Name.

⑤ IEEE 802.3 network address of node.

⑥ CRISP/32 version number. Not retrieved for software bus nodes.

Sample Dump (cont)

```
Last server return status number was 1 (00000001)
Transmit buffer address = 0009F6A0, length = 40
Last transmit message size = 39
Transmit Network header dump
     Network version number 8202
     Message sequence number = 1
     Network Request Options 0000
     Network Response Status 0010
     Message Length = 0
     Continuation sequence number = 0
     Client ID 0000
     Alternate response address 00-00-00-00-00-00
     Alternate response SAP 00
     Application type = 8
     Function code = 65535 (FFFF)
     Status code = 543649385 (20676E69)
Receive buffer address = 000A2088, length = 336
Last receive message size = 58
Receive Network header dump
     Network version number 8202
     Message sequence number = 1
     Network Request Options 0000
     Network Response Status 0020
     Message Length = 30
     Continuation sequence number = 0
     Client ID 0000
     Alternate response address 00-00-00-00-00-00
     Alternate response SAP 00
     Application type = 8
     Function code = 65535 (FFFF)
     Status code = 1 (00000001)
IEEE 802 communications status = 1 (00000001)
IEEE 802 communications handle = 661848 (000A1958)
IEEE 802 channel status = 1 (00000001)
IEEE 802 IOSB iostat = 1, message length = 58
IEEE 802 IOSB timeouts = 0, total timeouts = 0

There are a total of 1 context blocks, address 0009F678
    0 00000003                                                    (7)

Node List entry 1 unused

Node List entry 2 unused

Node List entry 3 unused

Node List entry 4 unused

=======================================================

Data Source List Header Data

List size = 5 entries
Data Source List Header Array address = 0009F528
```

Sample Dump (cont)

⑦ Hexadecimal printout of a series of bit masks. Each '1' bit represents a client context on the server. There can be a maximum of 65536 contexts per server.

Sample Dump (cont)

```
-------------------------------------------------------
Data Source List Header entry 0 is in use                                    (8)
DSL Header starting address = 0009F560
Data Source List Header self pointer = 0009F588
Data Source List Header self-pointer is okay
DSL type is real-time
DSL is resolved
All transmit lists sent
All DSL entries valid, and no overrun buffers

-------------------------------------------------------
There are 1 DSL entries

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DSL Entry number 0          (9)
DSL Entry self pointer = 000A1DD8
DSL Entry self-pointer is okay
DSL entry starting address = 000A1A00
Corresponding Data Base List entry number = 0                                (10)
 Data Base List entry number okay
Attempted to resolve symbol
Resolve process finished for symbol
Node name = "elab" (length = 4, address = 000A1AB0)
Status 00000004, Error code 00000000
Status description = "Symbol resolved successfully" (length = 28, address = 7FF30E05)
Entry is not a dual entry
User data = 512 (00000200)
Alternate node name = "" (length = 0)
Database name = "crisp" (length = 5, address = 000A1AF0)
Symbol name = "csp_s_time" (length = 10, address = 000A1AF8)
Subscript value = "" (length = 0)
Tracking option value = 0
Transfer count = 1
User's buffer address 00000B08, length 1024
Subscript value = 0
Symbol type code = 200  (String)
Array dimension = 0
Minimum data length = 25
Record number = 23
Update count = 1
Data value = " 6-DEC-1990 15:00:13.23" (string length = 23)
Back-pointer to Data Source List Header is okay

-------------------------------------------------------
Data Base List descriptor self pointer = 0009F580
Data Base List descriptor self-pointer is okay
There are 5 DBL entries, beginning at location 000A0998
```

Sample Dump (cont)

⑧ This is the first Data Source List.

⑨ This is the first DSL element.

⑩ This element uses the first entry in the Database List.

Sample Dump (cont)

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DBL Entry number 0 is in use                                          (11)
DBL Entry self pointer = 000A09F4
DBL Entry self-pointer is okay
Node name = "elab" (length = 4, address = 000A09A0)                   (12)
Database has been located
Server reports that database was active
Back-pointer to Data Source List Header is okay
Corresponding Node List entry number = 0
 Node List entry number okay
Database name = "crisp" (length = 5, address = 000A09E0)              (13)
Database number at server = 0
Database status at server = 001C
Database signature at server = 0093D01400000000 (20-SEP-1990 15:16:29.43)

Tracking entry:  ALWAYS                                               (14)
DBL Tracking Entry self pointer = 000A0C1C
DBL Tracking Entry self-pointer is okay
Back-pointer to Data Base List entry is okay
     Resolve is not in progress
     Transmit list has been sent
     Client ID number in use = 0001
     Number of symbols pointing to this entry = 1
     DSL entry pointer of resolve list = 000A1A00
     Count offset into resolve message = 34
     Data offset into resolve message = 38
     Master resolve message buffer size = 0
     Address of master resolve message = 00000000
     Last server return status number was 1 (00000001)
     Transmit buffer address = 000A22C0, length = 103
     Last transmit message size = 38
Transmit Network header dump                                          (15)
     Network version number 8202
     Message sequence number = 8
     Network Request Options 0000
     Network Response Status 0010
     Message Length = 0
     Continuation sequence number = 0
     Client ID 0001
     Alternate response address 00-00-00-00-00-00
     Alternate response SAP 00
     Application type = 6
     Function code = 19 (0013)
     Status code = 0 (00000000)
     Receive buffer address = 000A2348, length = 108
     Last receive message size = 71
```

Sample Dump (cont)

⑪ This is the first Database List entry for DSL #0.

⑫ Node name.

⑬ Database name.

⑭ Since none of the DSL elements specifies 'active' or 'standby' tracking, this entry is used.

⑮ The actual data buffers used for communicating with the server are part of the tracking entry.

Sample Dump (cont)

```
Receive Network header dump                                          (15)
     Network version number 8202
     Message sequence number = 8
     Network Request Options 0000
     Network Response Status 0020
     Message Length = 43
     Continuation sequence number = 0
     Client ID 0001
     Alternate response address 00-00-00-00-00-00
     Alternate response SAP 00
     Application type = 6
     Function code = 19 (0013)
     Status code = 1 (00000001)
     IEEE 802 communications status = 1 (00000001)
     IEEE 802 communications handle = 661848 (000A1958)
     IEEE 802 channel status = 1 (00000001)
     IEEE 802 IOSB iostat = 1, message length = 71
     IEEE 802 IOSB timeouts = 0, total timeouts = 0

Tracking entry:  ACTIVE
    Entry is unused

Tracking entry:  STANDBY
    Entry is unused

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DBL Entry number 1 unused

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DBL Entry number 2 unused

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DBL Entry number 3 unused

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DBL Entry number 4 unused

-------------------------------------------------------
DSL Header entry 1 is unused

-------------------------------------------------------
DSL Header entry 2 is unused

-------------------------------------------------------
DSL Header entry 3 is unused

-------------------------------------------------------
DSL Header entry 4 is unused

=======================================================

Normal status from dump operation
END OF DUMP OF WORF INTERNAL STRUCTURES

=======================================================
```

Sample Dump (cont)

⑮ The actual data buffers used for communicating with the server are part of the tracking entry.

Notes:

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X