Rocket

Rocket Model 204 Janus Open Client

Reference Manual

Version 7.4

July 2013
JOC-0704-RM-01

Notices

Edition

Publication date: July 2013

Book number: JOC-0704-RM-01

Product version: Rocket Model 204 Janus Open Client Version 7.4

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

Note

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100

Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Global Technical Support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone:

North America	+1 800 755 4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

Proprietary Notices . . . . . **ii**

Contents . . . . . **v**

Summary of Changes . . . . . **xi**
Sirius Mods Version 6.0 xi
Sirius Mods Version 4.6 xi

Chapter 1: Janus Overview **1**

Janus, the Sirius Mods, and UL/SPF 3
Versions and compatibility 4
Related manuals 5
Related products 6
System requirements 6

Chapter 2: Janus / Connectivity Concepts **9**

Server Ports 9
Environment Definition 10

Chapter 3: Janus Commands **11**

JANUS command overview 12
JANUS DEFINE 14
ALLOCC 16
BINDADDR xxx 16
BSIZE xxx 16
CHARSET xxx 17
IBSIZE xxx 17
LANGUAGE xxx 17
MASTER 18
NOUPCASE 18
OBSIZE xxx 18
PRELOGINUSER userid 19
RAWINPUT 19
RAWINPUTONLY 20
SSL 20
SSLBSIZE xxxx 21
SSLCACHE xxxx 22
SSLCIPH xxx 23
SSLCLCERT and SSLCLCERTR 23

SSLIBSIZE xxxx . . . 24
SSLMAXAGE xxx . . . 25
SSLMAXCERTL xxx . . . 25
SSLOBSIZE xxxx . . . 26
SSLPROT xxx . . . 26
SSLUNENC . . . 27
TCPKEEPALIVE . . . 28
TIMEOUT xxxx . . . 28
TRACE xxx . . . 29
UPCASE . . . 29
XTAB table . . . 30

Chapter 4: Janus Open Client $Functions . . . 31

Open Client User Language coding considerations . . . 32
$DB_ALTBIND . . . 37
$DB_ALTCOLID . . . 38
$DB_ALTCOLLEN . . . 39
$DB_ALTCOLNAME . . . 40
$DB_ALTCOLOP . . . 40
$DB_ALTCOLTYPE . . . 41
$DB_BIND . . . 42
$DB_CLOSE . . . 44
$DB_COLLEN . . . 44
$DB_COLNAME . . . 45
$DB_COLTYPE . . . 46
$DB_LANGPUT . . . 47
$DB_MSGHANDLE . . . 47
$DB_NEXTROW . . . 49
$DB_NUMALT . . . 50
$DB_NUMALTCOL . . . 50
$DB_NUMCOL . . . 51
$DB_NUMRET . . . 52
$DB_ONCLOSE . . . 52
$DB_OPEN . . . 54
$DB_RESULTS . . . 55
$DB_RETDATA . . . 55
$DB_RETLEN . . . 56
$DB_RETNAME . . . 57
$DB_RETSTATUS . . . 58
$DB_RETTYPE . . . 59
$DB_RPCINIT . . . 60
$DB_RPCPARAM . . . 61
$DB_SEND . . . 63
$DB_SQL . . . 63
$DB_TEST . . . 64
$DB_WAIT . . . 64

Appendix A: Sample Open Client programs **67**

Retrieve from SQL Server via Language Request 67

Appendix B: Datetime Processing Considerations **73**

Datetime Formats 74
Valid Datetimes 78
Processing Dates With Two-Digit Year Values 78
CENTSPAN 78
SPANSIZE 79
Datetime and format examples 80
$DB_ Functions CENTSPAN Argument 84

Index **85**

Figures

Figure 1: Model 204 TCP/IP Connectivity using Janus **2**

Figure 2: JANUS DEFINE command syntax **14**

Figure 3: Prepare & Send Request from *Janus Open Client* **35**

Figure 4: Retrieving Results of *Janus Open Client* Request **35**

Figure 5: $DB_ALTBIND function **37**

Figure 6: $DB_ALTBIND return codes. **37**

Figure 7: $DB_ALTCOLID function **38**

Figure 8: $DB_ALTCOLID return codes **38**

Figure 9: $DB_ALTCOLLEN function **39**

Figure 10: $DB_ALTCOLLEN return codes **39**

Figure 11: $DB_ALTCOLNAME function **40**

Figure 12: $DB_ALTCOLNAME return codes **40**

Figure 13: $DB_ALTCOLOP function **40**

Figure 14: $DB_ALTCOLOP return codes **41**

Figure 15: $DB_ALTCOLTYPE function 41
Figure 16: $DB_ALTCOLTYPE return codes 42
Figure 17: $DB_BIND function 42
Figure 18: $DB_BIND return codes 43
Figure 19: $DB_CLOSE function 44
Figure 20: $DB_CLOSE return codes 44
Figure 21: $DB_COLLEN function 44
Figure 22: $DB_COLLEN return codes 44
Figure 23: $DB_COLNAME function 45
Figure 24: $DB_COLNAME return codes 45
Figure 25: $DB_COLTYPE function 46
Figure 26: $DB_COLTYPE return codes 46
Figure 27: $DB_LANGPUT function 47
Figure 28: $DB_LANGPUT return codes 47
Figure 29: $DB_MSGHANDLE function 47
Figure 30: $DB_MSGHANDLE return codes 47
Figure 31: $DB_NEXTROW function 49
Figure 32: $DB_NEXTROW return codes 49
Figure 33: $DB_NUMALT function 50
Figure 34: $DB_NUMALT return codes 50
Figure 35: $DB_NUMALTCOL function 50
Figure 36: $DB_NUMALTCOL return codes 51
Figure 37: $DB_NUMCOL function 51
Figure 38: $DB_NUMCOL return codes 51

Figure 39: $DB_NUMRET function 52
Figure 40: $DB_NUMRET return codes 52
Figure 41: $DB_ONCLOSE function 52
Figure 42: $DB_ONCLOSE return codes 53
Figure 43: $DB_OPEN function 54
Figure 44: $DB_OPEN return codes 54
Figure 45: $DB_RETDATA function 55
Figure 46: $DB_RETDATA return codes 56
Figure 47: $DB_RETLEN function 56
Figure 48: $DB_RETLEN return codes 57
Figure 49: $DB_RETNAME function 57
Figure 50: $DB_RETNAME return codes 57
Figure 51: $DB_RETSTATUS function 58
Figure 52: $DB_RETSTATUS return codes 58
Figure 53: $DB_RETTYPE function 59
Figure 54: $DB_RETTYPE return codes 59
Figure 55: $DB_RPCINIT function 60
Figure 56: $DB_RPCINIT return codes 60
Figure 57: $DB_RPCPARAM function 61
Figure 58: $DB_RPCPARAM return codes 62
Figure 59: $DB_SEND function 63
Figure 60: $DB_SEND return codes 63
Figure 61: $DB_TEST function 64
Figure 62: $DB_TEST return codes 64

Figure 63: $DB_WAIT function 64

Figure 64: $DB_WAIT return codes 64

Summary of Changes

This section describes significant changes to the documentation. Usually, these changes correspond to enhancements made to the underlying product, although they might be simple documentation improvements.

Sirius Mods Version 6.0

The following changes correspond to changes in *Janus Open Client* in version 6.0 of the *Sirius Mods*.

- DEBUG keyword replaced by TRACE
- AUDTERM replaced by NOAUDTERM as port default (JANUS DEFINE)

Sirius Mods Version 4.6

The following changes correspond to changes in *Janus Open Client* in version 4.6 of the *Sirius Mods*.

- Manual converted for layout changes

CHAPTER 1 Janus Overview

Janus is a family of products that provides direct bi-directional access, via a TCP/IP network, between *Model 204* and programs running on other platforms. One of the Janus products, *Janus TCP/IP Base*, can be used by itself, and is also required by the other Janus products, each of which is optional. The Janus product family consists of the following:

Janus TCP/IP Base

Janus TCP/IP Base provides TCP/IP connectivity to *Model 204*.

It also includes a *Janus IFDIAL Library* for access to *Model 204* similar to IFDIAL, enabling TCP/IP access to *Model 204* from Unix workstations, DOS and Windows-based PCs, and other machines that support the C language and the TCP/IP protocol layers. The library has C routines for communication with a *Model 204* Online and C programs that can be used to communicate with a *Model 204* Online without additional programming, similar to the BATCH2 utility.

Janus Open Server

Janus Open Server provides a set of $functions that allow *Model 204* to be a server in response to Sybase CT and DB-Library Open Client calls and SQL EXECUTE statements.

A server application consists of a set of User Language procedures which are invoked by a client application's request. Client applications request the execution of a procedure via Sybase remote procedure calls (RPCs), which are implemented as part of Sybase's DB-Library Open Client code. The Sybase client sends the name of a stored procedure and an arbitrary number of parameters, and the Janus server executes the corresponding User Language procedure and returns the requested data in *Model 204* images (which appear as “rows” to the client) or as RPC return parameters.

Janus Open Client

Janus Open Client provides a set of $functions that allow *Model 204* applications to be client applications to Sybase SQL Servers or Open Servers. A *Model 204* client application sends RPCs or language requests (for example, SQL) to a Sybase open server or Sybase SQL server, and retrieves the results for further processing. It is possible for a User Language application to act as a client to several different servers simultaneously, and it is possible for a server application to simultaneously act as a client application.

Janus Specialty Data Store

Janus Specialty Data Store provides access to *Model 204* from Sybase Adaptive Server OmniConnect or from the older Sybase Omni SQL Server or Gateway. This includes optimized translation of SQL into User Language and a cataloging facility for mapping *Model 204* files to SQL structures. A user application issues SQL requests, which are routed to Sybase Omni and then routed to *Janus Specialty Data Store*, which provides the SQL response using data from one or more *Model 204* files.

The cataloging facility, JANCAT, maps *Model 204* files and fields onto a table/column structure, which Sybase Omni can then re-map onto its own table/column definitions. *Janus Specialty Data Store* does not require UNIQUE attributes nor any other alteration of your *Model 204* files.

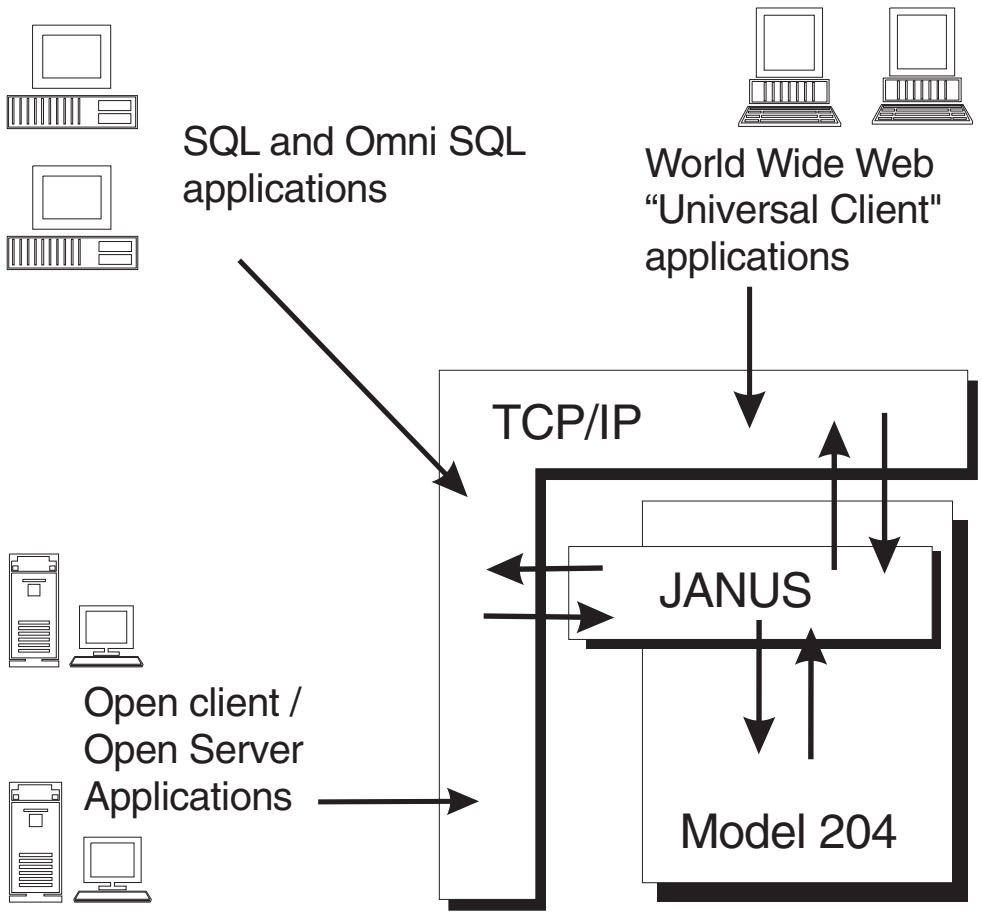
Janus Web Server

Janus Web Server is a full-featured Hyper-Text Transmission Protocol (http) server for the World Wide Web (WWW). It provides an interface to all WWW data types stored in *Model 204* procedures or database files. It can pass plain text, Hyper-Text Markup Language (HTML), and binary data types to web browser applications, and it can control access to secured, high-performance applications. *Janus Web Server* also provides a full API for building web applications in User Language.

Janus Network Security

Janus Network Security supports the SSL (Secure Sockets Layer) protocol, which provides secure communications for users of Janus products. It can be used with any Janus product, but it is most commonly used to secure communications of Janus Web Server applications. Users of web browsers communicating with such applications can be confident that their communications will be encrypted, and the identity of the server with which they are communicating is authenticated.

Using Janus products involves the **JANUS** *Model 204* command, which is documented in the *Model 204* documentation wiki (see **http://m204wiki.rocketsoftware.com/index.php/JANUS_command**). This command provides Janus port definition, starting, stopping, and monitoring of ports; for Web implementations it also defines the Web Server rules.



Model 204 TCP/IP Connectivity using Janus

The above figure shows that with TCP/IP as the communications protocol, Janus provides high-speed bi-directional access to *Model 204* from any client.

1.1 Janus, the Sirius Mods, and UL/SPF

Janus Open Client is part of the Janus family of products that provides connectivity to the *Model 204* database. A site that has *Janus Open Client* must have the *Janus TCP/IP Base* because without it, it is impossible to use *Janus Open Client*. A *Janus Open Client* site might also have one or more of the other products in the Janus family, though no others are required. Note that if *Limited Janus Web Server* is available, then *Janus TCP/IP Base* is automatically authorized. *Limited Janus Web Server* is a free, restricted version of *Janus Web Server*; they are both documented in the *Model 204* documentation wiki (see **http://m204wiki.rocketsoftware.com/index.php/Janus_Web_Server**).

The Janus family of products is itself made up of two distinct components:

- A collection of object code enhancements to the *Model 204* database-engine nucleus.

  Prior to version 7.5 of *Model 204*, these enhancements were distributed as components of the *Sirius Mods* and made up a collection of products including those in the Janus family. The *Sirius Mods* included many non-connectivity related products (such as *Fast/Backup*, *Fast/Reload*, and the *Fast/Unload User Language Interface*) not part of the Janus family. No *Sirius Mods* products are required to run *Janus Open Client* other than itself and *Janus TCP/IP Base*.

- A collection of *Model 204* procedures that contain User Language, documentation, and assorted other data.

  These *Model 204* procedures install and implement the components of the User Language Structured Programming Facility, also known as **UL/SPF**. All the *UL/SPF* files reside in the SIRIUS procedure file, which also contains code and data useful to Janus users including *Janus Open Client* users.

  UL/SPF also includes files that are components of non-connectivity related products such as *SirPro*, *SirScan*, and *SirMon*. No other *UL/SPF* products are required to run *Janus Open Client*, or any other Janus product, for that matter.

Thus, to install *Janus Open Client*, both the *Sirius Mods* and *UL/SPF* must be installed, following the instructions in the *Sirius Mods Installation Guide* and the UL/SPF installation guide (**http://m204wiki.rocketsoftware.com/index.php/UL/SPF_installation_guide**), respectively. If the *Sirius Mods* are installed, all other products owned by the installing site that are part of the *Sirius Mods* will also be installed. Similarly, when *UL/SPF* is installed, all other products owned by the installing site that are part of *UL/SPF* will be installed.

1.2 Versions and compatibility

Because the *Sirius Mods* and *UL/SPF* have somewhat different release cycles, the version numbers for these two components will often differ in a distribution. For example, version 7.6 of the *Sirius Mods* might be shipped with version 7.3 of *UL/SPF*. All the products in *UL/SPF* depend on certain features being present in the version of the *Sirius Mods* that is installed in the *Model 204* load module under which *UL/SPF* is running. This implies, obviously, that the *Sirius Mods* must be installed for any *UL/SPF* component to operate correctly. The *Sirius Mods* version must match or be higher than the *UL/SPF* version number.

The *Sirius Mods* however, do not depend on any particular features of the *UL/SPF* product, merely the presence of the *UL/SPF* SIRIUS file. The SIRIUS file contains the code for the sample Janus Web Server, and Janus port definitions have default rules that call to this file.

Any User Language application (including *UL/SPF*) that uses the *Sirius Mods* runs correctly on subsequent versions of the *Sirius Mods*. It is, thus, always possible to upgrade the *Sirius Mods* without having to worry about upgrading *UL/SPF*. This is not to say that this is always a good idea, only that it is possible and that the installed version of a *UL/SPF* product will continue to run as it had before the *Sirius Mods* upgrade.

While the Janus family of products has a *UL/SPF* component, most of the critical code is actually in the *Sirius Mods* — object code enhancements to the *Model 204* nucleus. The *UL/SPF* component of the Janus family consists mostly of utilities, examples, and documentation. Because of this, the version number of a Janus product is generally considered to be the version of the *Sirius Mods* in which it is contained.

Any documentation that requires at least a particular version of the *Sirius Mods* or *UL/SPF* will be clearly marked to indicate this. For example, a JANUS DEFINE parameter that is only available in versions 7.7 and later will have a sentence such as "This parameter is only available in version 7.7 and later of *Sirius Mods*" in its documentation. If a feature, $function, command, or parameter is not indicated as requiring any specific version of the *Sirius Mods*, it can be assumed that it is available, as documented, in all versions of *Janus Open Client*.

1.3 Related manuals

As mentioned in "Janus, the Sirius Mods, and UL/SPF" on page 3, *Janus Open Client* requires the installation of both the *Sirius Mods* (prior to version 7.5 of *Model 204*) and *UL/SPF*. As such, the person responsible for the installation of *Janus Open Client* should refer to the *Sirius Mods Installation Guide* and the UL/SPF installation guide (**http://m204wiki.rocketsoftware.com/index.php/UL/SPF_installation_guide**). In addition, documentation of *Sirius Mods* error messages (**http://m204wiki.rocketsoftware.com/index.php/Category:Sirius_Mods_messages**) might be useful to application programmers.

As stated in on page 3, *Janus Open Client* depends on the *Janus TCP/IP Base* product, and there is much useful information starting at **http://m204wiki.rocketsoftware.com/index.php/Janus_TCP/IP_Base**.

You are authorized to use a number of Sirius $functions along with *Janus Open Client*, including procedure processing $functions and $list processing $functions. Unless otherwise stated, these functions, and any $functions mentioned in this manual that are not documented here and that are not standard *Model 204* $functions, are documented in the documentation wiki (see **http://m204wiki.rocketsoftware.com/index.php/List_of_$functions**).

1.4 Related products

The *Janus TCP/IP Base* must be installed to use *Janus Open Client*. This is the only other product that must be installed in a *Model 204* region to use *Janus Open Client*.

If security is a concern, whether it be internet or intranet security, SSL (Secure Socket Layer) is the de-facto standard for providing encryption and validation security for web-based applications. The *Janus Network Security* product provides SSL support for *Janus Open Client* (as well as other products in the Janus family).

One of the convenient debugging features available with *Janus Open Client* is a TRACE facility which logs Janus request/response information to the *Model 204* journal. If you don't have good tools to view the journal, using it for debugging is a tedious process. AUDIT204 and ISPF provide some capabilities for viewing the journal, but they have many inherent shortcomings and inefficiencies. Because of this, it is **strongly** recommended that any site that installs *Janus Open Client* also install *SirScan*.

SirScan is a product in the *UL/SPF* family that facilitates the interactive extraction of journal information within the *Model 204* region. It does so via a user-friendly web browser or full-screen 3270 interface and low-level routines to provide efficient access to in-memory and on-disk journal buffers. *SirScan* can provide an order of magnitude improvement in debugging efficiency for non-terminal-related *Model 204* processes such as Horizon, BATCH2 and Janus server applications.

1.5 System requirements

The current release of *Janus Open Client* requires the following components to run:

- Mainframe operating systems:
  - Any supported version of z/OS
  - z/VSE Version 4 or later or
  - CMS (releases currently supported by IBM) running under any supported version of z/VM
- *Model 204* Version 6 Release 1 or later
- One of the following mainframe TCP/IP implementations:
  - IBM TCP/IP for z/VM or z/OS
  - InterLink TCP/IP for MVS - Version 1.1 or later
  - TCP/IP for VSE (Connectivity Systems, Inc., Columbus, OH) - Version 1 Release 4.0 or later

One of the following is required for a server communicating with *Janus Open Client*:

Sybase DB-Library Release 4 or later

any client software that uses TDS release 4 or later

CHAPTER 2 Janus / Connectivity Concepts

2.1 Server Ports

In order for a client application to communicate with a server application it must have a way to identify the server application on the network. Under the TCP/IP protocols the identity of a server has two parts. The first part identifies the machine on which the server runs. This part is called the machine's (or host's) IP address. The second part distinguishes the server application from other applications on the host. This part is called the port number.

A host's IP address is a 32 bit unsigned binary number that is displayed in “dotted” format, i.e. 198.242.244.33. To avoid having to refer to these types of addresses most networks have nameservers or names files that map names to IP addresses. That way a client application can connect to a host by a name (such as IBM3090) rather than an address.

A port number is a number from 1 to 65535 that is assigned to every server application that is available on a host. In the case of a Janus IFDIAL Server or a Janus Open Server, this port number is specified by the second parameter on the JANUS DEFINE command. Since this port number must be unique for the host, it is impossible to start (JANUS START) a port with a port number that matches a port number for any other application running on the same host. This includes any Janus port on the same or different Online. This also includes any other non-Janus server application. For example, port number 23 is almost always used by the telnet server. An attempt to start a Janus server for port number 23 will undoubtedly encounter a port in use situation and be unable to start. On a system with several local server applications, or more than one Online (maybe test and production) with several Janus ports, a simple strategy to keep port numbers from conflicting is to simply assign a range of ports to each Online. For example, port numbers 300-399 might be reserved for the test Online and port numbers 400-499 might be reserved for the production Online.

Sybase provides a way of mapping an application name to a host name (or address) and port number. This makes it possible to access a specific application by specifying only a single application name. This mapping is done through a mapping file called the interfaces file on Unix workstations, and through entries in WIN.INI under Microsoft Windows. For more information on this mapping refer to the Sybase DB/Library manuals.

2.2 Environment Definition

Once the Janus object modules are linked into *Model 204*, the system manager must modify the CCAIN stream for Janus operation.

The following parameters should be set by all Janus sites:

TCPSERV name of the TCP/IP server address space (MVS) or virtual machine (CMS). If not specified, this parameter defaults to `TCPIP`.

TCPTYPE specifies the type of TCP/IP network to which *Model 204* is connected. The valid values are

IBM	to specify IBM TCP/IP (the default).
INTERLNK	to specify InterLink TCP/IP.
KNET	to specify K-Net TCP/IP.

Any Janus session which treats the *Model 204* address space as a server — that is, IFDIAL or Open Server sessions — requires the same resources as any user session, including a *Model 204* server and a thread on which to run. A special facility called an SDAEMON (pronounced ess-demon), makes these resources available.

SDAEMONs are special users that are activated when *Model 204* establishes a Janus Open Server or IFDIAL connection. SDAEMONs are also used by other $functions, methods, and *Model 204* products. Janus IFDIAL and Server support require an SDAEMON for each active connection. This means that at least as many SDAEMONs must be defined in the Online as the maximum number of concurrent IFDIAL and Janus Open Server connections that must be supported. SDAEMON definition is explained in the *Model 204* documentation wiki (see **http://m204wiki.rocketsoftware.com/index.php/Sdaemons**).

Most of the communication with the TCP/IP address space is accomplished via a PST. Because of this, NSUBTKS may need to be increased by 1 before using Janus.

CHAPTER 3 Janus Commands

The Janus command set (simply referred to as "Janus commands") consists of commands and subcommands that begin with the string JAN. The two Janus commands currently supported are **JANUS** and **JANUSDEBUG**. For a full description of the Janus commands, see **http://m204wiki.rocketsoftware.com/index.php/JANUS_command** and **http://m204wiki.rocketsoftware.com/index.php/JANUSDEBUG_command**. The manual you are reading describes the commands and $functions that are specific to *Janus Open Client*.

You use Janus commands to:

- Define *Model 204* as a server on the TCP/IP network. Janus commands set port numbers for your Janus server applications and start, stop, and monitor Janus activity in the *Model 204* address space.
- Define remote servers to the *Model 204* client for access by *Janus Open Client* applications and *Janus Sockets* client applications, and define which remote hosts can establish connections with *Janus Specialty Data Store*, *Janus Open Server*, and *Janus Sockets*.
- Add security to Janus ports using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to provide encrypted communications.

Janus commands require the executing user to have System Manager privileges. The JANUSDEBUG command, which can be issued by any logged-in user, is an exception. Janus commands can also be issued as operator commands (on the Online virtual console under VM) or as replies to the HALT message under OS/390.

Janus commands make use of the following wildcard characters:

*	An asterisk represents any string of characters.
?	A question mark represents any character.
"	A double quote escapes wildcard translation of the special character that follows it.

For example, the following command starts all Janus ports whose names begin with the string BA (like BART, BARNEY, BALES, BAY):

```
JANUS START BA*
```

The following command drains all Janus ports whose names are three characters long, beginning with BA (BAY, BAD, BAX, etc.):

```
JANUS DRAIN BA?
```

The following command starts all Janus ports whose names end in ? (WHODONEIT?, WHERES_THE_BEEF?, WHAT?_ME_WORRY?, WHO_YA_GONNA_CALL?, etc.):

```
JANUS START *"?
```

3.1 JANUS command overview

The principal command of the Janus command set is the JANUS command, which consists of a set of mutually exclusive subcommands. To execute a subcommand, you specify it with the prefix JANUS: for example, JANUS DEFINE ..., JANUS STATUS ..., etc.

The following list shows the JANUS subcommands with a brief description of what they do. For more information about the subcommands not described in this manual, see **http://m204wiki.rocketsoftware.com/index.php/Janus_commands** in the *Model 204* documentation wiki.

Subcommand execution requires System Manager privileges.

ADDCA	Adds a trusted certifying authority's certificate to a port.
CHARSET	Specifies the default character set.
CLSOCK	Specifies rules to allow a User Language program to access a CLSOCK port.
CONFIGURATION	Displays global configuration values.
DEFINE	Defines a Janus port.
DEFINEIPGROUP	Defines a grouping of IP addresses for web access control.
DEFINEREMOTE	Defines a remote server for *Janus Open Client*, and associates it with a Janus OPENSERV or SDS port.
DEFINEUSGROUP	Defines a grouping of user IDs for web access control.
DELCA	Deletes a trusted certifying authority's certificate from a port.
DELETE	Deletes a port definition.
DELETEIPGROUP	Deletes a grouping of IP addresses.

DELETEREMOTE	Deletes an association between a remote server and a Janus OPENSERV or SDS port.
DELETEUSGROUP	Deletes a grouping of user IDs.
DISPLAY	Displays Janus port definitions.
DISPLAYCA	Displays the contents of a trusted certifying authority's certificate.
DISPLAYREMOTE	Displays remoter server definitions.
DISPLAYSOCK	Displays CLSOCK and SRVSOCK port rules.
DISPLAYWEB	Displays WEBSERV port rules.
DISPLAYXT	Displays translate table definitions.
DOMAIN	Specifies the domain; used with IBM TCP/IP to resolve unqualified host names.
DRAIN	Prevents new connections to port and stops port when last connection is closed.
FORCE	Breaks all connections to port and stops port when last connection is closed.
FTP	Specifies Janus FTP Server processing rules.
LANGUAGE	Specifies default *Janus Open Server* language.
LIMITS	Displays the Janus connection limits for an Online.
LOADXT	Loads or reloads a translate table and, optionally, an entity translate table.
NAMESERVER	Specifies IP address and port number of the domain name server used with *Janus Sockets* CLSOCK applications and *Janus Open Client* applications; only used with the IBM TCP/IP interfaces.
RELOAD	Reloads the *Model 204*-to-SQL mappings from the JANCAT file for a *Janus Specialty Data Store* port.
SRVSOCK	Specifies rules that determine which SRVSOCK connections to allow.
SSLSTATUS	Displays SSL (Secure Sockets Layer) statistics for SSL ports.
START	Makes a port available for connections.

STATUS	Displays port status.
STATUSCA	Displays the status of a trusted certifying authority's certificate.
STATUSREMOTE	Displays status of remote servers.
TCPLOG	Stores all input and output streams to and from a port.
TRACE	Changes trace settings for a port or for specific IP addresses connected to a port.
TSTATUS	Displays thread utilization statistics.
WEB	Specifies *Janus Web Server* processing rules.

3.2 JANUS DEFINE

The JANUS DEFINE command is used to specify the characteristics of a *Janus Open Client* port as well as any other Janus port. It defines the usage of the named port as one of the following:

- Access by IFDIAL clients
- Open Server or Open Client connections
- Specialty Data Store access
- Web access
- FTP server connections
- Telnet server connections
- Generic Sockets usage — with the *Model 204* online either requesting (CLSOCK) or accepting (SRVSOCK) the connection
- Connection between the *Janus Debugger* or *TN/3270 Debugger* workstation GUI and programs being debugged in *Model 204*

For any except a CLSOCK or DEBUGGERCLIENT port, this subcommand associates a service with a TCP/IP port number.

Among the characteristics specified by JANUS DEFINE is whether the port will use Secure Sockets Layer (SSL) for encrypted communications.

```
JANUS DEFINE portname portnum type maxcon other_parms...
```

JANUS DEFINE command syntax

Where each of the first four parameters is positional and required:

portname A 1- to 30-character name by which the port is identified. It is used on other JANUS subcommands, such as JANUS START and JANUS DISPLAY.

portnum The TCP/IP port number at which the service is available. *portnum* is the server port number, and it must be between 1 and 65535, inclusive. This number is used by client applications on the network when they require access to the *Model 204* server. The server port number must be unique on the host. Several "well-known" port numbers for various TCP/IP services (for example, 53 for nameserver) should be avoided. For a discussion of server ports, see **http://m204wiki.rocketsoftware.com/index.php/Defining_server_ports**.

type Port type. For *Janus Open Client* usage, you must define one or more ports of type SDS or OPENSERV, and you may need to use the MASTER parameter on a port definition.

maxcon Maximum number of simultaneous active connections to be allowed on the port. This number must be less than or equal to the number of TCP/IP connections for which the site is licensed.

If you are defining multiple ports for your site, the sum of the *maxcon* connections you define is allowed to be greater than the number for which the site is licensed. In this case, *Janus Web Server* will automatically prevent any connection that would exceed the site license limit.

For *Janus Open Client*, note that a server-to-server connection requires an extra connection for the **site handler**. Thus, a single connection to a remote server would use two connections, while 10 connections to a remote server would use 11.

There are no restrictions on the allowed values for maxcon, but licensed thread limits are still enforced at the time a connection is made.

You can use the JANUS TSTATUS command to view the thread usage and connection limits for your port, and you can use the JANUS LIMITS command to view similar information for your Online.

other_parms A set of blank-delimited parameters that describe the characteristics of and processing to be performed on the port. These parameters are keywords, sometimes followed by values. They are all optional, except:

- for OPENSERV ports, CMD is required

All the parameters allowed on the JANUS DEFINE command are documented in the documentation wiki at **http://m204wiki.rocketsoftware.com/index.php/JANUS_DEFINE#JANUS_DEFINE_parameters**

For your convenience, only those applicable to port definitions used with *Janus Open Client* are described in the following subsections.

3.2.1 ALLOCC

This parameter indicates that input, output and request buffers are to be allocated when a connection is established and are to be freed when the connection is closed. If ALLOCC is not specified, all necessary buffers are allocated when the JANUS START command is executed and are kept until the port is stopped, after a JANUS DRAIN or JANUS FORCE command. All buffers are allocated above the line using space reserved by SPCORE.

3.2.2 BINDADDR xxx

This parameter specifies the IP address to which the port will be bound, if the host (machine) on which *Model 204* is running supports multiple IP addresses. The IP address must be an IP address of the host. If BINDADDR is not specified, the port binds the port number for all IP addresses associated with the host; that is, it can be accessed via any IP address associated with the host.

This parameter only really makes sense on a host with more than one IP address. For example, if a host on which an Online is running has IP addresses 198.242.244.47 and 198.242.244.130, a `BINDADDR 198.242.244.47` specification indicates that the port can only be reached through IP address 198.242.244.47.

This parameter is especially useful for allowing a single mainframe host or even an Online to act as more than one web server without the inconvenience of having port numbers on URLs. This can be done because there can be multiple port 80's (the default web port number) on the host, each accessed by its indicated BINDADDR. The separate IP addresses could, in turn, be associated with different DNS host names even though these separate names refer to the same underlying machine.

Note that there is not likely to be much, if any, performance benefit to having multiple Janus ports with the same port number but different BINDADDRs in the same Online. There might certainly be, however, some organizational advantages to running such a configuration.

3.2.3 BSIZE xxx

This parameter specifies the size of the TCP/IP input and output buffers. The default is 4096 for IBSIZE and 8192 for OBSIZE. BSIZE is a shorthand way of specifying both IBSIZE and OBSIZE when their sizes are the same.

3.2.4 CHARSET xxx

This parameter indicates, to the remote host, the character set being used by Janus. This allows a port-specific override of either the default character set or the character set specified on the JANUS CHARSET subcommand. The default character set is iso_1. CHARSET has no effect on the operation of any application in *Model 204*. The name of the specified character set is simply forwarded to the remote host.

For SDS ports, the commands are executed before the port begins acting as a Specialty Data Store. It is strongly recommended that this command be used mainly to set user table sizes and user parameters for SDS ports. This might be necessary because *Janus Specialty Data Store* might have very different table size requirements than other applications running on an sdaemon.

For WEB ports the commands specified by CMD are executed after all rules are executed except the ON rules. The specified commands can be used to invoke an APSY subsystem when using the Janus Web UL API or to reset UTABLEs and other parameters.

Examples of some valid CMD clauses:

```
JANUS DEFINE MYWEB 80 WEBSERV 10 CMD WEBAPSY
JANUS DEFINE SDS204 1777 SDS 20 -
      CMD 'R MCPU 5000' AND 'UTABLE LQTBL 1000' -
      AND 'R PROMPT 16'
JANUS DEFINE OPENXXX 1234 OPENSERV 15 -
      OPEN FILE OPENPROC CMD 'R PROMPT 16' AND -
      OPENAPSY
```

This parameter is required for OPENSERV port types.

3.2.5 IBSIZE xxx

This parameter specifies the size of the TCP/IP input buffer. The default is 4096, the minimum is 512, and the maximum is 65534 (prior to version 5.2 the maximum was 32767). There is one input buffer used for each connection.

A larger input buffer size provides better CPU performance in both *Model 204* and the TCP/IP address space at the expense of more virtual (and real) storage. Generally, the size of the input buffer has an impact only on a port being used for *Janus Open Client* or *Janus Sockets* connections or on a *Janus Web Server* port used for file uploads.

3.2.6 LANGUAGE xxx

This parameter indicates, to the remote host, the language being used by Janus. This allows a port-specific override of either the default language or the language specified on

the JANUS LANGUAGE subcommand. The default language is us_english. LANGUAGE has no effect on the operation of any application in *Model 204*. The name of the specified language is simply forwarded to the remote host.

3.2.7 MASTER

This parameter specifies that this is the default port for *outgoing* connections to remote servers. A single MASTER port will serve as the access route to multiple external server address spaces. The servers may be other *Model 204* servers or may be Sybase/Microsoft servers that are to receive *Janus Open Client* function calls.

Users accessing the *Model 204* address space over an OPENSERVER port will use the same port they came in on for any outgoing *Janus Open Client* connections. Users accessing the *Model 204* address space with other threads (for example, 3270 or web based applications) must have a MASTER port defined in order for the *Janus Open Client* functions to establish connections to other address spaces.

Note that the port number is irrelevant for outgoing purposes, though it must still be specified.

Multiple ports may be DEFINEd and STARTed with the MASTER parameter specified, but the one used in any particular instance will not be predictable.

3.2.8 NOUPCASE

This parameter indicates that no client data is to be converted to upper case. By setting NOUPCASE the userid and password must be specified by the client in the correct case (probably upper case). Note that it is possible to have lower case userids and passwords in *Model 204*. For example, the userids HOMER, homer and Homer would be treated as three separate userids by *Model 204*. The NOUPCASE parameter simplifies the interaction between clients where names tend to be in lower case or case-insensitive and *Model 204* where they tend to be in upper case.

The NOUPCASE parameter is the opposite of UPCASE. The default is for all ports to have UPCASE set.

3.2.9 OBSIZE xxx

This parameter specifies the size of the TCP/IP output buffer. The default is 8192, the minimum is 512, and the maximum is 65534 (prior to version 5.2 the maxiumum was 32767). There is one output buffer used for each connection.

A larger output buffer size provides better CPU performance in both *Model 204* and the TCP/IP address space at the expense of more virtual (and real) storage.

3.2.10 PRELOGINUSER userid

This parameter indicates the userid under which pre-login processing runs. Pre-login processing is that which occurs before a user login.

Pre-login processing runs under the default userid of “NO USERID” or under the userid specified by the PRELOGINUSER parameter; it is visible to *SirMon*, the MONITOR command, and the LOGWHO command; and it is BUMP'able.

On many port types, much processing can take place before a thread is actually logged on to a user. The PRELOGINUSER parameter can be useful in helping distinguish users in pre-login processing on different ports.

3.2.11 RAWINPUT

This parameter tells *Janus Web Server* to save the raw input stream for an HTTP POST, regardless of the mime type set by the client in the `content-type` header. This has two basic advantages:

1. The raw input content for an HTTP POST is always available to *Janus Web Server* applications (via $web_input_content) regardless of the content-type. This could be useful for debugging, or perhaps for logging, input content.

2. It is possible for *Janus Web Server* to interact correctly with clients that don't set the mime type, regardless of what content they send. Prior to the availability of RAWINPUT, if a client sent, say, XML data, but it did not set the content-type, *Janus Web Server* would assume that the content was `application/x-www-form-urlencoded` (form POST) encoded. If after it read some of the content, *Janus Web Server* discovered that it was not HTML form data, it was too late: the request had to be rejected for having an invalid format.

   With the RAWINPUT parameter set, however, *Janus Web Server* proceeds as follows:

   a. It loads the input content into CCATEMP.

   b. If the mime type is set to `application/x-www-form-urlencoded`, or if it is not set at all, *Janus Web Server* determines if the input has the `application/x-www-form-urlencoded` format.

   c. If the format is *not* `application/x-www-form-urlencoded`, the request is *not* rejected, and the *Janus Web Server* application can still access the data.

3.2.12 RAWINPUTONLY

RAWINPUTONLY indicates that, regardless of the POST data content-type set by the client, *Janus Web Server* should do both of the following:

- Save the raw input stream of an HTTP POST.
- Refrain from parsing the input content into form fields.

RAWINPUTONLY is very similar to the RAWINPUT port definition parameter ("RAWINPUT" on page 19), except that:

- RAWINPUTONLY can be an ON rule parameter, so it can be set for specific URLs.

- RAWINPUT does not prevent *Janus Web Server* from trying to parse the form parameters, if the `content-type` for the POST is set to `application/x-www-form-urlencoded` or `multipart/form-data`. RAWINPUTONLY prevents this parsing, so it protects *Janus Web Server* applications from errors in this parsing. These errors include invalid-form-data errors and request-buffer-full errors.

For more information about RAWINPUTONLY processing, see the "RAWINPUTONLY | NOTRAWINPUTONLY" section in **http://m204wiki.rocketsoftware.com/index.php/JANUS_WEB_ON#Named_parameters**.

3.2.13 SSL

The SSL parameter indicates that communications on this port should be encrypted using *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) support. The parameter has the following mutually exclusive options:

SSL procfile procname
: Identifies the file (typically JANSSL) and procedure that contain the certificate to be presented to clients on server ports and to the server on CLSOCK ports.

SSL *
: Presents to the client or server the "self-signed certificate" provided for your site by *Janus Network Security*.

SSL 0
: Indicates for CLSOCK ports that, although the connection is encrypted, the client is not to provide a certificate to the server if requested.

  Server certificates are required to establish an encrypted connection, but client certificates are optional and are not used at all by many secured servers.

Certificates and authentication are described further in the *Janus Network Security Reference Manual*.

Other optional DEFINE command parameters used in conjunction with the SSL parameter include:

- For server sockets:
  SSLBSIZE, SSLCIPH, SSLCLCERT/SSLCLCERTR,
  SSLIBSIZE, SSLOBSIZE, SSLPROT, SSLSES

- For client sockets:
  SSLOPT

- For both types of sockets:
  SSLCACHE, SSLMAXAGE, SSLMAXCERTL, SSLUNENC

Other JANUS commands useful for SSL ports include:

- For ports that authenticate incoming certificates:
  ADDCA, DELCA, DISPLAYCA, STATCA

- For monitoring a port's SSL activity:
  SSLSTAT

Janus Web Server $functions useful for SSL applications and described in the documentation wiki at **http://m204wiki.rocketsoftware.com/index.php/List_of_Janus_Web_Server_$functions** include:

$WEB_CERT_INFO, $WEB_CERT_LEVELS, $WEB_CIPHER,
$WEB_PROTOCOL, $WEB_SECURE

3.2.14 SSLBSIZE xxxx

This tuning parameter specifies the size of the input buffer used for reading encrypted data for an SSL port. An SSL port is a Janus port whose definition includes an SSL parameter (“SSL” on page 20) setting, which indicates that communications on this port may be encrypted using *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) support.

The SSLBSIZE parameter also specifies the size of the SSL output buffer. To set the input and output buffer sizes independently, you use the SSLIBSIZE and SSLOBSIZE parameters.

The default for SSLBSIZE is 4096 bytes; the minimum and maximum values are 1024 and 32767, respectively.

If you set SSLBSIZE greater than the SSL specification maximum buffer size of 16000, the port's input buffer size is set to the SSLBSIZE value, but the output buffer size is set to 16000 bytes. Setting the input buffer greater than 16000 bytes might be necessary if

the port will have connections with SSL implementations that don't fully conform to the SSL specification. For more information about buffer sizing and about Janus handling of oversized packets, see ("SSLIBSIZE xxxx" on page 24) and ("SSLOBSIZE xxxx" on page 26).

3.2.15 SSLCACHE xxxx

This parameter specifies the number of entries in virtual storage to be allocated for caching information related to this port's SSL sessions. A Janus port whose definition includes an SSL parameter ("SSL" on page 20) setting supports *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) encrypted sessions.

The SSL cache helps limit the CPU overhead of establishing an SSL session. It does not reduce the effectiveness of security, but it does reduce the overhead at the cost of a relatively small amount of virtual storage.

SSL sessions can persist for a length of time determined by either the client or server. *Janus Network Security* limits the life-span of SSL V2 connection sessions to the lesser of 2 minutes or the value of SSLMAXAGE ("SSLMAXAGE xxx" on page 25), and it limits SSL V3 and TLS connections to 1440 minutes (24 hours). For most sites, the default SSLCACHE should be sufficient.

Each session requires approximately 512 bytes per entry to cache session related information. A further SSLMAXCERTL ("SSLMAXCERTL xxx" on page 25) bytes are required to hold server certificates for CLSOCK ports, or to hold client certificates for Janus server ports that request them by including SSLCLCERT or SSLCLCERTR ("SSLCLCERT and SSLCLCERTR" on page 23).

If the SSLCACHE value is too small, and a larger than anticipated number of users attempt to access an SSL-secured port, entries in the cache are removed on a least-recently-used basis. This may lead to greater overhead for re-execution of the CPU intensive initial public-key/private-key encryption/decryption operations. The indicator that the SSLCACHE value is not large enough to hold all the contemporaneous SSL sessions is a non-zero value in the "SesNF" column of the JANUS SSLSTAT command result. This is not necessarily problematic as long as the SesNF value is relatively small, because it is not unreasonable to suffer an occasional lost session in order to reduce virtual storage.

Note: SSLCACHE is specified in *entries*, and the default SSLCACHE allocation is the number of storage entries required for 16 times the number of threads defined on the port. So by default, 10 threads would result in 160 entries; at 512 bytes per entry, this would require 81,920 bytes of virtual storage. 100 threads would require 819,200 bytes.

The default SSLCACHE value is likely to be excessively large for CLSOCK ports that only connect to a single or to a few servers. All CLSOCK connections to a particular server use the same SSL session regardless of how many different threads initiate connections.

3.2.16 SSLCIPH xxx

This parameter lets you limit the stream ciphers (encryption algorithms) that this port offers for SSL connections. A Janus port whose definition includes an SSL parameter ("SSL" on page 20) setting supports *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) encrypted connections.

Typically, SSLCIPH is allowed to default to 0: all the Janus-supported ciphers are available, and the cipher that is ultimately used depends on the outcome of the handshake negotiation with the client that seeks the service at this port. The negotiation selects the strongest available cipher that the client can support.

However, to make only a subset of the server ciphers available, you can specify SSLCIPH followed by the (bitmask) value that selects the subset. For example, SSLCIPH 2 indicates that only strong RC4 encryption is available.

Currently, these ciphers are supported:

1 RC4 bulk cipher with MD5 digest algorithm with 40 bits of the 128 bit RC4 key transmitted encrypted, the rest transmitted "in the clear" (unencrypted). This is considered a moderately strong encryption algorithm and is available on virtually every client implementation of SSL.

2 RC4 bulk cipher with MD5 digest algorithm with all 128 bits of the RC4 key transmitted encrypted. This is considered a very strong encryption algorithm but is only available on clients that have been specially configured to support this cipher. This encryption level is not available for export from the United States.

3.2.17 SSLCLCERT and SSLCLCERTR

These parameters specify that an SSL server port will request an SSL certificate from the client. An SSL port is a Janus port whose definition includes an SSL parameter ("SSL" on page 20) setting, which indicates that communications on this port may be encrypted using *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) support.

If the client does *not* present a certificate when requested:

- SSLCLCERT specifies that normal processing should continue.
- SSLCLCERTR specifies either of the following:
  - The connection should be closed with no further processing (and "MSIR.0646: Error requesting client certificate - client did not have required certificate" is journaled).

- Processing continues to run the SSLNOCERTERR exception handler, if this is a WEBSERV port and an ON SSLNOCERTERR clause is part of the port definition. For information about this exception handler, see **http://m204wiki.rocketsoftware.com/index.php/JANUS_WEB_exception_rules**, and also see the example below.

To verify a certificate that is passed by a client, you must first have added to the port one or more CA-signed certificates by using the JANUS ADDCA command (**http://m204wiki.rocketsoftware.com/index.php/JANUS_ADDCA**).

When a client presents a certificate, that certificate is available to User Language code via $WEB_CERT_LEVELS and $WEB_CERT_INFO on WEBSERV ports, and it is available via $SOCK_CERT_LEVELS and $SOCK_CERT_INFO on SRVSOCK ports.

The following example shows a web server SSL port definition that specifies the SSLCLCERTR parameter, JANUS ADDCA commands that are needed to store CA-signed certificates to authenticate the client certificate, and a rule that specifies the ONSSLCERTERR exception handler for cases where the client does not present a certificate:

```
JANUS DEFINE CLCERTWEB 9733 WEBSERV 1Ø HTTPVERSION 1.1  -
       SSL JANSSL TM2ØØ8.PKEY SSLCLCERTR

JANUS ADDCA CLCERTWEB MYPROC SECURESE.CERT
JANUS ADDCA CLCERTWEB MYPROC THAWTE.CERT
JANUS ADDCA CLCERTWEB MYPROC VERIJUNK.CERT

JANUS WEB CLCERTWEB ON SSLNOCERTERR OPEN FILE MYPROC    -
       CMD 'INCLUDE MISSING_CERTIFICATE_ERROR'
```

3.2.18 SSLIBSIZE xxxx

This parameter specifies the size of the SSL input buffer to be used on SSL ports. An SSL port is a Janus port whose definition includes an SSL parameter (“SSL” on page 20) setting, which indicates that communications on this port may be encrypted using *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) support.

The size of the SSL input buffer was specified with the SSLBSIZE parameter (“SSLBSIZE xxxx” on page 21) before the SSLIBSIZE parameter became available.

Technically, the maximum “legal” SSL buffer size is 16000, but it may be necessary to use a larger input buffer if there will be connections with SSL implementations that don't fully conform to the SSL specification. If an application tries to send an SSL packet larger than SSLIBSIZE to a Janus SSL port, the connection will be broken and an error written to the audit trail (MSIR.0386 SSL INPUT MESSAGE TOO LONG - INCREASE SSLBSIZE). The other side of the SSL connection will not receive this error message or any other indication of why the connection was broken. There will be no effect on other users on the same port.

The default for SSLIBSIZE is 4096, and the minimum and maximum allowable values are 1024 and 32767, respectively.

For WEBSERV ports that are used for file uploads (HTTP PUT or form-based uploads), it will probably be necessary to set SSLIBSIZE to at least 16000, because most browsers will send SSL packets that are as large as possible. For most other applications, the SSLIBSIZE default is probably sufficient, though web applications that POST very large forms might require a slight increase of SSLIBSIZE.

3.2.19 SSLMAXAGE xxx

This parameter specifies the maximum number of minutes that an SSL session is to be maintained. A Janus port whose definition includes an SSL parameter (“SSL” on page 20) setting supports SSL (Secure Sockets Layer) or TLS (Transport Layer Security) encrypted sessions. The discussion of this SSLMAXAGE parameter uses "SSL" to refer to SSL or TLS.

An SSL session is a series of SSL connections that are made using a single “master secret” shared by the SSL client and server. To set up an SSL session, the master secret must be exchanged using computationally expensive public-key/private-key encryption/decryption. SSL sessions are a way of reducing the overhead of SSL by reducing the number of public-key/private-key encryption/decryption operations.

The SSLMAXAGE default is 1440 (24 hours), which is the specified maximum life-span of an SSL V3 or a TLS session. The maximum life-span of an SSL V2 session is 2 minutes, so larger values of SSLMAXAGE are ignored for SSL V2 sessions.

The 24-hour life-span of SSL V3 and TLS sessions is generally considered "safe," but if even greater security is required, a smaller SSLMAXAGE can be specified. Setting SSLMAXAGE to 0 forces a new session for every request, which forces a public-key/private-key encryption/decryption operation for every connection. This might be useful for benchmarking the overhead associated with the public-key/private-key operations. The JANUS SSLSTAT command can provide useful information in monitoring the efficacy of SSL session caching.

3.2.20 SSLMAXCERTL xxx

For a Janus port defined (by the `SSL` parameter) to support encrypted connections, this parameter indicates the number of bytes of virtual storage to be allocated to hold incoming certificates presented for authentication. Authentication verifies (or not) the certifying authority signature on the incoming certificate. Such a certificate may be:

- A server certificate sent in reply to a CLSOCK port.
- A client certificate sent in reply to a WEBSERV, SRVSOCK, OPENSERV, or SDS port that has the SSLCLCERT or SSLCLCERTR parameter in its definition.

Since incoming certificates are cached, SSLMAXCERTL bytes are allocated for each SSL session in the cache, the size of which is determined by the explicit or implicit setting of the SSLCACHE parameter ("SSLCACHE xxxx" on page 22).

The default SSLMAXCERTL size is 1024, which should be large enough to hold most certificates received from clients or servers. The minimum and maximum SSLMAXCERTL values are 256 and 32767, respectively. It is unlikely that any incoming certificate will be smaller than 512 bytes, and it is extremely unlikely that an incoming certificate will be larger than 2048 bytes. If an incoming certificate is larger than SSLMAXCERTL, an error message is logged to the audit trail and the connection is closed.

3.2.21 SSLOBSIZE xxxx

This parameter specifies the size of the SSL output buffer to be used on SSL ports. An SSL port is a Janus port whose definition includes an SSL parameter ("SSL" on page 20) setting, which indicates that communications on this port may be encrypted using *Janus Network Security* SSL (Secure Sockets Layer) or TLS (Transport Layer Security) support.

There is little or no performance benefit to using large SSL output buffers, because the amount of work associated with creating an SSL output packet is almost directly proportional to the size of the packet. Typically, it is sensible to use the default SSLOBSIZE of 4096, or even to make it smaller to save on memory.

The default for SSLOBSIZE is 4096, and the minimum and maximum allowable values are 1024 and 16000, respectively.

For *Model 204* to *Model 204* applications, the SSLOBSIZE on each side must be less than or equal to the SSLIBSIZE ("SSLIBSIZE xxxx" on page 24) on the other side.

3.2.22 SSLPROT xxx

This parameter lets you specify the degree of SSL-like encryption available at this port. *Janus Network Security* currently supports two Secure Socket Layer (SSL) protocols (SSL V2 and SSL V3) and the Transport Layer Security (TLS) protocol, an extension to SSL V3 but developed by the IETF Internet standards group.

During the negotiation for a connection to or from this port, Janus will offer the most secure protocol available, then, if necessary, will fall back to the next lower one available, and so on. The SSLPROT parameter lets you explicitly disallow one or more protocols from the negotiation.

SSLPROT is a bitmask parameter whose main values are:

X'01' SSL, V2 support. This is less secure than SSL V3 or TLS.

X'02' SSL, V3 support. This is less secure than TLS.

X'04' TLS, V1 support.

X'07' The default. SSL V2, SSL V3, and TLS are available. Janus will try for them in the order: TLS, SSL V3, SSL V2.

A typical reason for explicitly specifying an SSLPROT value is to require a more secure connection for a port. If a client attempts to connect to a Janus server port using a protocol explicitly disallowed by SSLPROT, the connection is immediately broken, except for WEBSERV ports where the SSLPROTOCOLERR exception handler will be run if available.

Janus CLSOCK ports will attempt to connect under the most secure protocol available, and will fall back to the next-most secure protocol available; if less-secure protocols are disallowed by SSLPROT, the connection attempt will fail.

3.2.23 SSLUNENC

This parameter indicates that an unencrypted private key is being used in the certificate specified by the SSL parameter ("SSL" on page 20) on this Janus server port definition.

This parameter is obsolete because *Janus Network Security* now automatically determines whether or not the private key is encrypted, and if not, prompts for a password. A corrupted private key procedure could lead *Janus Network Security* to believe that the private key must be encrypted, and so result in a password prompt.

Using unencrypted private keys is discouraged.

Formerly, SSLUNENC was required on a port definition if an unencrypted private key was used. Otherwise, the JANUS START command for an SSL-secured port would prompt for a password (technically, a seed for the encryption algorithm) to use to decrypt the private key. Any data, or even a null value, entered for the password will incorrectly be used in an attempt to decrypt the private key (rendering the key unusable), and the START will fail.

Similarly, if an encrypted private key **were** used in the certificate specified on the SSL parameter, the SSLUNENC parameter was **not** to be specified. Specifying SSLUNENC would prevent password prompting for that key, thus bypassing decryption of the private key (rendering it unusable), and causing the START to fail.

The certificate and private key generation process is described further in the *Janus Network Security Reference Manual*.

3.2.24 TCPKEEPALIVE

This parameter specifies that connections on the port should use TCP keepalives. TCP keepalives request that the TCP stack send periodic “keepalive” packets to the communications partner to see if it is still there. The time interval between these packets, which cannot be set by Janus, is set in the TCP/IP stack configuration. For example, with the IBM stacks, the keepalive interval is set in the TCPCONFIG INTERVAL parameter for BPX (IBM Communications Server) and in the KEEPALIVEOPTIONS INTERVAL parameter for VM TCP/IP.

In some sense, the term “keepalive” is a misnomer — keepalive packets that are not responded to cause a connection to be closed, so keepalives actually cause connections to be closed faster than they might be otherwise.

TCPKEEPALIVE probably only makes sense for ports where connections are held open for long periods of time. TNSERV ports are the most likely candidate. For these ports, TCPKEEPALIVE might be useful for two reasons:

1. It can detect connections lost due to a client failure (say, a turned-off workstation), reducing threads wasted for connections to lost clients.

2. It can reassure certain routers, especially those doing network address translation (NAT) that the connection is still active. Some routers will stop routing packets for connections on which no activity is seen for some period of time. Keepalives ensure that there is periodic activity on a connection, even if there is no user interaction. Of course, for this to be successful, the TCP/IP stack's keepalive interval must be less than any applicable router's inactivity timeout. For this particular application, keepalives live up to their name.

Since the TCP/IP stack does the keepalives, the overhead in *Model 204* for setting this parameter is virtually zero.

3.2.25 TIMEOUT xxxx

This parameter specifies the number of seconds of inactivity after which clients connected to this port will be disconnected. The default for TIMEOUT is 0, which means that connections never time out.

For WEBSERV ports Browser requests never involve waits on user input so the TIMEOUT parameter for WEBSERV ports involves terminating connections when network response is **extremely** slow or cases where the client workstation has been turned off before a response is received from *Janus Web Server*. Because of this, TIMEOUT can be set fairly aggressively for WEBSERV ports. A value of 60 (seconds) would be reasonable.

For all other port types The TIMEOUT value should reflect the fact that a connection might require user input waits.

3.2.26 TRACE xxx

This parameter specifies the initial TRACE setting for the port. The TRACE setting controls what Janus-related trace information is logged to the audit trail. The port TRACE setting can be overridden by the JANUS TRACE command.

Like the JANUS TRACE command, the TRACE parameter value is a bit mask integer that sums the values of the options that will be logged. The default value is 3 for SDS and OPENSERV ports, and it is 0 for WEBSERV and all other ports. For a description of the individual bit options and for more information about the TRACE setting, see **http://m204wiki.rocketsoftware.com/index.php/JANUS_TRACE**.

3.2.27 UPCASE

This parameter indicates that all client “names” are to be converted to upper case. “Names” includes userids and passwords, variable names for OPENSERV ports, column names for SDS ports and header parameters, header values, cookie names, and form field names for WEBSERV ports. By setting UPCASE as a port parameter, the userid and password can be specified by the client in case insensitive form, that is, it can be specified in lower case.

Note that it is possible to have lower case userids and passwords in *Model 204*. For example, the userids HOMER, homer, and Homer would be treated as three separate userids by *Model 204*. The UPCASE parameter simplifies the interaction between clients (where names tend to be in lower case) and *Model 204* (where they tend to be in upper case).

Note: The UPCASE parameter never results in data being converted to upper case. That is, if a client sends variable “@customer” with a value of “Dolly Dinkle”, and UPCASE is active for the connection, the User Language application would see a variable called “@CUSTOMER” with a value of “Dolly Dinkle”.

For SDS ports, the UPCASE parameter means that all table and column names passed from the Adaptive Server will be converted to upper case. This means that when defining the columns and tables (using JANCAT), the names must all be upper case. It also means that if an SDS port has the UPCASE parameter set but has mixed case table and column names, those tables and columns will be inaccessible.

The UPCASE parameter is the opposite of NOUPCASE. The default is for all ports to have UPCASE set.

3.2.28 XTAB table

This parameter indicates the EBCDIC-to-ASCII, ASCII-to-EBCDIC, and character entity translation tables to be used for the port.

You can specify a translation table that has not yet been loaded with the JANUS LOADXT command, but the table must be loaded before the port can be started.

The default translation table is `STANDARD`, which is a fairly generic pair of EBCDIC-to-ASCII and ASCII-to-EBCDIC translate tables that formerly was the only available option.

You can replace a translate table with the JANUS LOADXT command at any time, even if the port has active connections.

The XTAB parameter is Valid for all port types.

CHAPTER 4 Janus Open Client $Functions

Janus Client Functions allow client applications, written in User Language, to place calls to a Sybase SQL Server, a Sybase Open Server or a Janus *Model 204* Open Server. Client Functions are prefixed with ***$DB***_, and provide the facility by which User Language applications can communicate with remote servers.

Although *Sirius Mods* version 6.5 made mixed-case User Language available for use with Sirius Janus products, the *Janus Open Client Reference Manual* retains the all-uppercase presentation for $function names and User Language entities. For more information about mixed-case User Language, see **http://m204wiki.rocketsoftware.com/index.php/Mixed-case_User_Language**.

The *Model 204* $functions that communicate with and respond to remote servers over Janus TCP/IP ports are listed below and described thereafter in individual sections.

$DB_ALTBIND Bind a *Model 204* variable to an alternate column.

$DB_ALTCOLID Get the base column number that was used to create an alternate column.

$DB_ALTCOLLEN Get the length of an alternate column.

$DB_ALTCOLNAME Get the name of an alternate column.

$DB_ALTCOLOP Get the operator that was used to create an alternate column.

$DB_ALTCOLTYPE Get the type of an alternate column.

$DB_BIND Bind a *Model 204* variable to a column.

$DB_CLOSE Close a connection.

$DB_COLLEN Get the length of a column.

$DB_COLNAME Get the name of a column.

$DB_COLTYPE Get the type of a column.

$DB_LANGPUT Send language data (probably SQL) to server.

$DB_MSGHANDLE Specify the message handler.

$DB_NEXTROW	Get the next row from the remote server.
$DB_NUMALT	Get number of alternate rows.
$DB_NUMALTCOL	Get number of alternate columns.
$DB_NUMCOL	Get number of columns.
$DB_NUMRET	Get number of returned parameters.
$DB_ONCLOSE	Specify action to take on closed connection.
$DB_OPEN	Open a connection to a remote server.
$DB_RESULTS	Wait for reply from server.
$DB_RETDATA	Get the value of a return parameter.
$DB_RETNAME	Get the name of a return parameter.
$DB_RETLEN	Get the length of a return parameter.
$DB_RETSTATUS	Get the return status from the last RPC.
$DB_RETTYPE	Get the type of a return parameter.
$DB_RPCINIT	Set up to perform a remote procedure call.
$DB_RPCPARAM	Add an RPC parameter.
$DB_SEND	Send request to server.
$DB_SQL	Send language data (probably SQL) to server.
$DB_TEST	Test connection to server.
$DB_WAIT	Wait for reply from server.

4.1 Open Client User Language coding considerations

Before the Open Client functions can be used, a JANUS port must be defined and have a remote server associated with it via the JANUS DEFINEREMOTE command.

Janus Open Server applications--applications that are front-ended on workstations and access the *Model 204* online via Janus Open Server ports--will automatically use the same port they came in on for outgoing access to remote servers. For this sort of

application, you must execute the JANUS DEFINEREMOTE command to associate a remote server with the port used by the incoming client application, as shown in the example below:

```
* Specify the domain:
JANUS DOMAIN sirius-software.com
* Specify the location of the name server:
JANUS NAMESERVER 198.242.244.131
* Define the port the client application
* will come in on and the action it will take:
JANUS DEFINE REGION1 1001 OPENSERV 20 OPEN FILE -
    PRODPROC CMD 'I REGIONAL.QUERIES,1,HIGH'
* Associate a remote server with REGION1:
JANUS DEFINEREMOTE REGION1 REGION1_SQL_SERVER -
    SPARC2 3001 TIMEOUT 120 -
    SITEUSER BART SITEACCT SIMPSONS
* Make the port available for client access:
JANUS START REGION1
```

Janus applications that originate *inside* the *Model 204* online (3270-based applications for example) require an OPENSERV port to be defined as MASTER, and a remote server associated with it. The other JANUS commands are similar to those above, so:

```
* Specify the domain:
JANUS DOMAIN sirius-software.com
* Specify the location of the name server:
JANUS NAMESERVER 198.242.244.131
* Define master port for outgoing calls -
*  disallow incoming use of the port,
*  let submitting user be the site handler:
JANUS DEFINE REGION1 1001 OPENSERV 20 CMD '*' -
    OUTONLY MASTER
* Define a remote server to REGION1:
JANUS DEFINEREMOTE REGION1 REGION1_SQL_SERVER -
    SPARC2 3001
* Make the port available for client access:
JANUS START REGION1
```

Once the ports and remote servers are defined, User Language programs using the Open Client functions can be run against them.

The Janus Open Client functions can send either language requests (usually SQL) or RPC's (Remote Procedure Calls) to a remote server. The remote server may reply by sending back rows of data, “alternate rows” (rows generated by SQL COMPUTE statements), return parameters, or nothing. The *Janus Open Client* functions include functions that let you define characteristics of the server connection, how you want to handle server messages and error conditions, and functions to manipulate the rows, alternate rows or parameters returned by the server.

A Janus Open Client application can establish one or more logical connections with one or more remote servers. Each logical connection is identified by a *connection identifier* that is returned by $DB_OPEN at the time it is established. Almost all *Janus Open Client* functions (except $DB_MSGHANDLE and $DB_ONCLOSE) operate on a specific connection and hence require the *connection identifier* as their first argument. In addition, any connection might be lost at any time (because of a JANUS FORCE, a BUMP of the sitehandler, a termination of the remote server, a network error, etc..) so any Janus Open Client function that operates on a connection might encounter a connection lost error. A connection lost error always results in the *Janus Open Client* $function returning a -1. The ERRCLOSE option on the $DB_OPEN command eliminates the need to check for this condition with every function call, and the $DB_ONCLOSE function allows the programmer to specify a label where processing is directed when the connection to the remote server is unexpectedly lost.

A Janus Open Client User Language program is always in a particular state in relation to its remote server. That state is based on which $DB_ functions have been executed and on the response of the remote server. The connection to the remote server is *half-duplex*, with the User Language client opening the connection, building a request and sending it, and waiting for a response from the remote server before any more action can be directed at the connection. Here, briefly, is the order in which processing proceeds in a client program:

A program has no connection to a remote server until it performs a $DB_OPEN. The $DB_OPEN returns a *connection identifier*--usually held in a %variable--and places the connection into an uninitialized send state. The program then initializes the connection for *remote procedure calls* (RPCs) using $DB_RPCINIT, or initializes it for *language requests*, using $DB_LANGPUT.

If a connection is initialized for RPCs, $DB_RPCPARAM is used to set parameter names and values for the RPC. If the connection is initialized for language requests, subsequent calls to $DB_LANGPUT may be executed to further define the language request. When the request is completely specified it is sent to the remote server with a $DB_SEND, which places the program into an uninitialized receive state; that is, the client program can continue executing, but cannot refer to the remote server connection in any way until it executes a $DB_WAIT. (If $DB_WAIT is called without a previous call to $DB_SEND, the request is sent to the remote server by $DB_WAIT). $DB_WAIT initializes the receive state so that processing pauses, and waits for a response fron the server.

The return code from $DB_WAIT determines how the receive state proceeds. Aside from error conditions, $DB_WAIT will indicate that the remote server sent a row, an alternate row or return data. *Rows* are the returned database information from SQL SELECT statements. *Alternate rows* are the return data from SQL COMPUTE statements. Alternate rows are sometimes called *compute rows*. *Return data* can be parameter names and values, or status codes for the RPC.

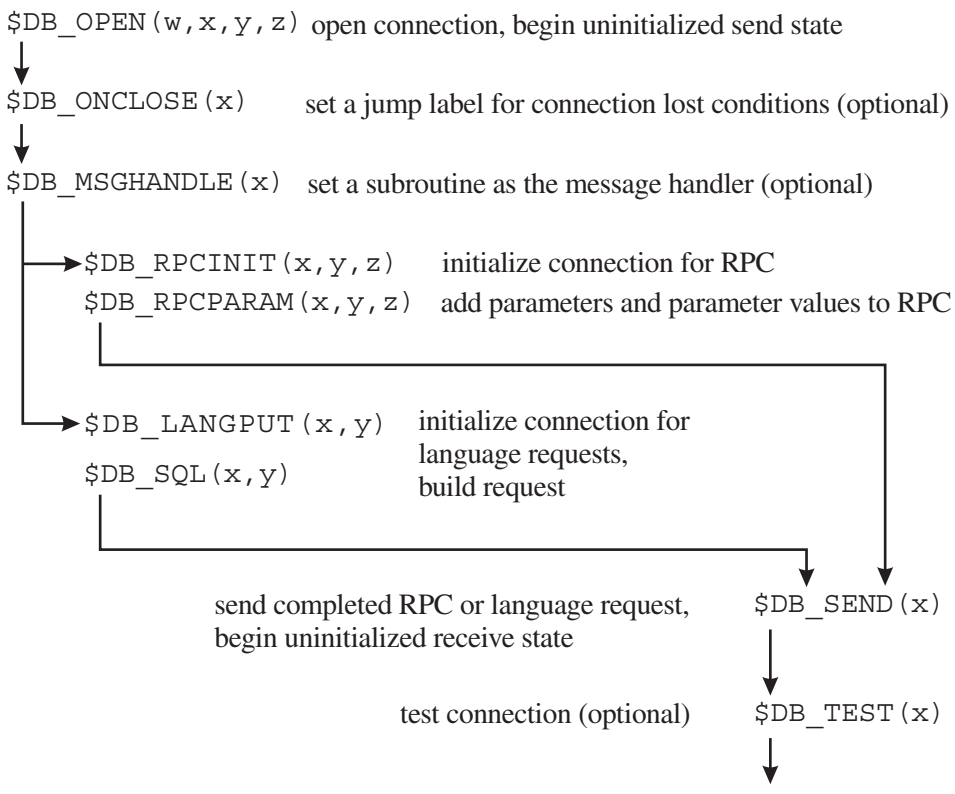
Most of the remaining $DB_ functions are specific to handling one of the three types of information returned by a server. When receiving row or alternate rows, returned

columns must be *bound* to User Language %variables. A program must determine the column structure of the return rows, and then bind the columns of interest to %variables using $DB_BIND or $DB_ALTBIND (for rows and alternate rows respectively). The %variables become populated with the values in the columns to which they are bound at the next invocation of *$DB_NEXTROW*.

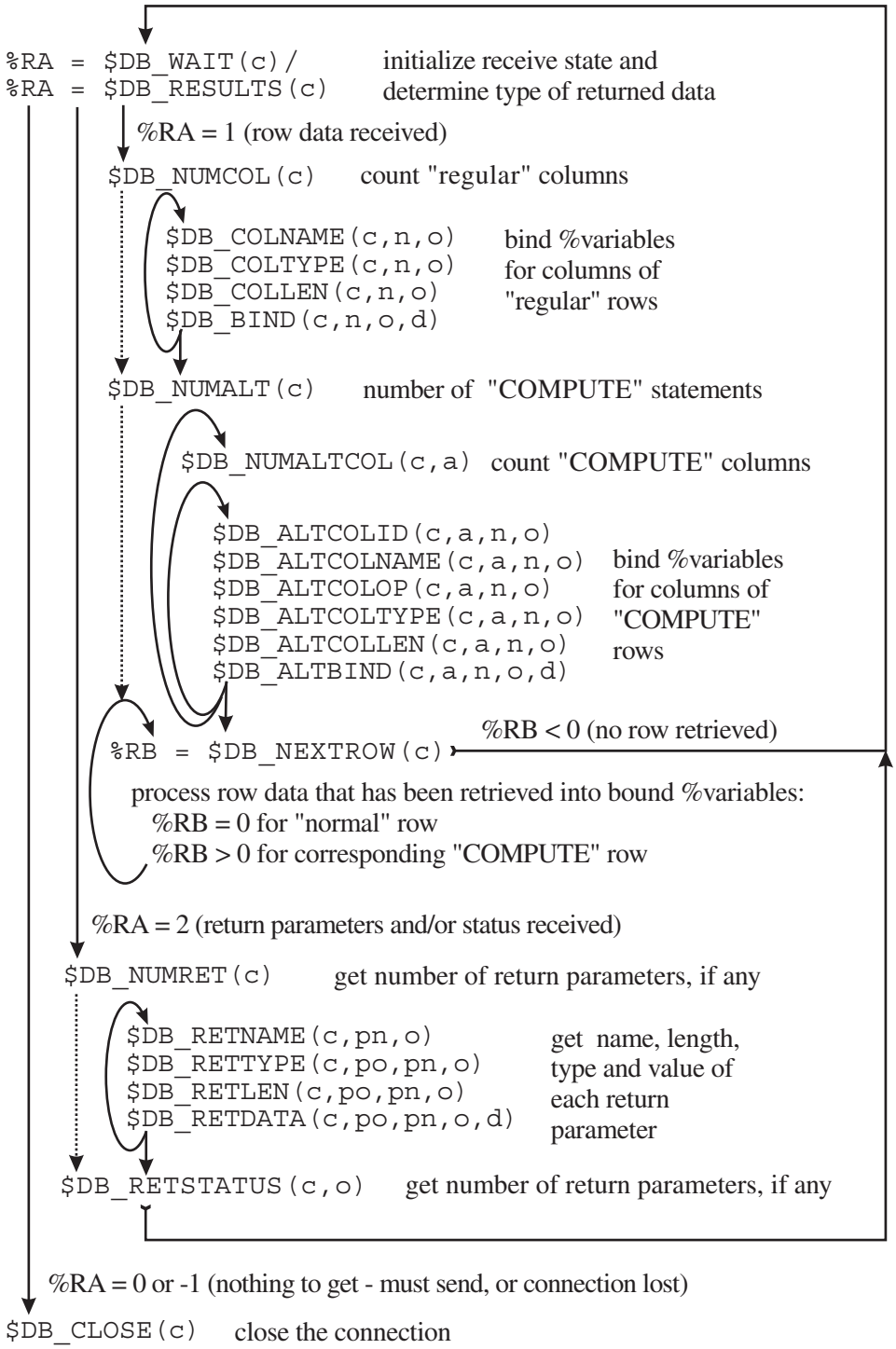
A few other functions handle processing cleanup. $DB_CLOSE closes a specific connection or all connections. $DB_MSGHANDLE and $DB_ONCLOSE provide a way of handling messages and broken connections. Client applications can be written so they open a connection and hold it open while any number of requests are defined and sent. They may also be written so the connection is closed after each client request, and reopened when another interation is required with the server.

Sample programs in the appendicies show how to use the functions in the context of particular applications. These and other sample programs are also in the *Model 204* procedure file JANUS, installed with the Janus products.

All input arguments to $DB_ functions are required arguments unless otherwise noted.



Prepare & Send Request from *Janus Open Client*



Retrieving Results of *Janus Open Client* Request

4.2 $DB_ALTBIND

$DB_ALTBIND binds the contents of a specified column in an alternate row to a %variable.

Binding associates a return column with a %variable. The %variable is populated with the value when a row is retrieved using $DB_NEXTROW.

$DB_ALTBIND accepts 5 arguments and returns a 0 for success or a negative error return code.

```
%RC = $DB_ALTBIND( connection_id, compute_id, -
column_no, %variable, date_time_format )
```

$DB_ALTBIND function

The first argument is the connection_id returned by $DB_OPEN when the connection was established with the remote server.

The second argument identifies the COMPUTE row of interest. The first COMPUTE clause creates alternate_row 1, the second COMPUTE clause creates alternate_row 2, and so on. The COMPUTE id is the same as the altername row number. The number of alternate rows a request has generated can be determined using $DB_NUMALT.

The third argument is the column number in the alternate row that you wish to bind to the %variable.

The fourth argument is the %variable to which the column will be bound.

The fifth argument is a datetime format for columns that are Sybase DATETIME or SMALLDATETIME data types; it is ignored for other column types. See “Datetime Formats” on page 74 for an explanation of datetime formats.

```
 0 - Bind successful
-1 - Connection lost
-2 - Not receiving rows
-3 - No such alternate row
-4 - No such column
-5 - Invalid target variable
-6 - Invalid datetime format
```

$DB_ALTBIND return codes.

For example, given the following block of SQL:

```
select * from customer
  order by store
  compute sum(gross), sum(net), sum(cost), avg(net) by store
```

The following fragment of User Language binds the 4 columns of the computed alternate row to the variables %SUM_GROSS, %SUM_NET, %SUM_COST and %SUM_AVG_NET:

```
%RC = $DB_ALTBIND(%CONN1,1,1,%SUM_GROSS)
%RC = $DB_ALTBIND(%CONN1,1,2,%SUM_NET)
%RC = $DB_ALTBIND(%CONN1,1,3,%SUM_COST)
%RC = $DB_ALTBIND(%CONN1,1,4,%SUM_AVG_NET)
```

4.3 $DB_ALTCOLID

$DB_ALTCOLID gets the identifier (the number) for a column from which a specified alternate column was created.

$DB_ALTCOLID accepts 4 arguments and returns a numeric result code. The column identifier is placed into a percent variable specified as the fourth argument.

```
%RC = $DB_ALTCOLID( connection_id, -
          compute_id, col_no, %variable )
```

$DB_ALTCOLID function

The first argument is the connection_id returned by $DB_OPEN when the connection was established with the remote server.

The second argument is the number of the alternate row.

The third argument is the number of the column in the alternate row for which the source column id is requested.

The fourth argument is a %variable to be assigned the alternate column id.

```
 0 - Target set
-1 - Connection lost
-5 - Invalid target variable
```

$DB_ALTCOLID return codes

Given the following SQL statements:

select store, invoice from customer order by store compute count(invoice) by store

The following call:

```
%RESULT = $DB_ALTCOLID( %CONNECTION, 1, 1, -
            %COLUMN_SOURCE )
```

returns a 0 in %RESULT, and sets %COLUMN_SOURCE to 2, indicating that the second regular column is the source column for the first computed value in the first alternate row (on connection %CONNECTION).

4.4 $DB_ALTCOLLEN

$DB_ALTCOLLEN gets the maximum length for the returned data in a column for an alternate row.

$DB_ALTCOLLEN accepts 4 arguments and returns a numeric result code.

```
%RC = $DB_ALTCOLLEN( connection_id, -
        compute_id, col_no, %variable )
```

$DB_ALTCOLLEN function

The first argument is the connection identifier.

The second argument is the id of the alternate row containing the column for which the length is requested.

The third argument is the number of the column in the alternate row for which the length is requested.

The fourth argument is the %variable to be assigned the length of the column identified in the third argument.

```
 0 - Target set
-1 - Connection lost
-2 - Not receiving rows
-3 - No such alternate row
-4 - No such column
-5 - Invalid target variable
```

$DB_ALTCOLLEN return codes

For example:

```
%RESULT = $DB_ALTCOLLEN( %CONNECTION, 1, 1, -
    %COMPUTE_LEN(1) )
```

returns a 0 in %RESULT, and sets %COMPUTE_LEN(1) to the maximum length of the data returned by the server for alternate row 1, column 1.

4.5 $DB_ALTCOLNAME

$DB_ALTCOLNAME gets the name of an alternate column.

$DB_ALTCOLNAME accepts 4 arguments and returns a numeric result code.

```
%RC = $DB_ALTCOLNAME( connection_id, -
          compute_id, col_no, %variable )
```

$DB_ALTCOLNAME function

The first argument is the connection identifier.

The second argument is the id of the alternate row containing the column for which the name is requested.

The third argument is the number of the column in the alternate row for which the name is requested.

The fourth argument is the %variable to be assigned the column name.

```
 0 - Target set
-1 - Connection lost
-2 - Not receiving rows
-3 - No such alternate row
-4 - No such column
-5 - Invalid target variable
```

$DB_ALTCOLNAME return codes

For example:

```
%RESULT = $DB_ALTCOLNAME( %CONNECTION, %Y, -
             %X, %COMPUTE_NAME(%X) )
```

returns a 0 in %RESULT, and sets %COMPUTE_NAME(%X) to the name of the column %X in alternate row %Y.

4.6 $DB_ALTCOLOP

$DB_ALTCOLOP gets the operator that was used to aggregate a specified column in an alternate row.

$DB_ALTCOLOP accepts 4 arguments and returns a result code. The operator is placed into a percent variable specified as the fourth argument.

```
%RESULT = $DB_ALTCOLOP
```

$DB_ALTCOLOP function

The first argument is the connection identifier. This is a required argument.

The second argument is the id of the alternate row containing the column for which the operator is requested.

The third argument is the number of the column in the alternate row for which the operator is requested.

The fourth argument is the %variable to be assigned the operator for the specified column. Returned operators will either be 'COUNT', 'SUM' or 'AVG'.

```
 0 - Target set
-1 - Connection lost
-5 - Invalid target variable
```

$DB_ALTCOLOP return codes

Given the following SQL statements:

```
select store, invoice from customer
  order by store, invoice
  compute count(invoice) by store
```

The following call:

```
%RESULT = $DB_ALTCOLOP( %CONNECTION, 1, 1, -
   %COMPUTE_OPERATOR )
```

returns a 0 in %RESULT, and sets %COLUMN_OPERATOR to 'COUNT', indicating that the first computed value in the first alternate row was aggregated with a count statement.

4.7 $DB_ALTCOLTYPE

$DB_ALTCOLTYPE gets the type of data in a column of an alternate row.

$DB_ALTCOLTYPE accepts 4 arguments, returns a numeric result code, and places the data type in the %variable specified as the fourth argument.

```
%RC = $DB_ALTCOLTYPE( connection_id, -
      compute_id, col_number, %var )
```

$DB_ALTCOLTYPE function

The first argument is the connection identifier. This is a required argument.

The second argument is the id of the alternate row containing the column for which the type is requested.

The third argument is the number of the column in the alternate row for which the type is requested.

The fourth argument is the %variable to be assigned the data type of the column.

```
 0 - Target set
-1 - Connection lost
-2 - Not receiving rows
-3 - No such alternate row
-4 - No such column
-5 - Invalid target variable
```

$DB_ALTCOLTYPE return codes

The following call:

%RC = $DB_ALTCOLTYPE(%CONNECTION, 4, 3, %TYPE)

returns a 0 in %RC, and sets %TYPE to the data type of the third COMPUTE in the fourth alternate row.

Data type returned by $DB_ALTCOLTYPE will be one of the following:

```
INT          DATETIME
SMALLINT     SMALLDATETIME
TINYINT      CHAR
FLOAT        VARCHAR
REAL         BINARY
MONEY        VARBINARY
```

4.8 $DB_BIND

$DB_BIND binds the value of a specified column to a %variable.

Binding associates a return column with a %variable. The %variable is populated with the value when a row is retrieved via $DB_NEXTROW.

$DB_BIND accepts 4 arguments and returns a numeric result code.

```
%RC = $DB_BIND( connection_id, col_no, -
      %variable, date_time_format )
```

$DB_BIND function

The first argument is the connection identifier.

The second argument is the number of the column whose values will be assigned to the %variable specified in the third argument.

The third argument is the %variable to which the column will be bound.

The fourth argument is a datetime format for columns that are Sybase DATETIME or SMALLDATETIME data types; it is ignored for other column types. See "Datetime Formats" on page 74 for an explanation of datetime formats.

```
 Ø - Bind successful
-1 - Connection lost
-2 - Not receiving rows
-4 - No such column
-5 - Invalid target variable
-6 - Invalid datetime format
```

$DB_BIND return codes

For this select statement:

```
select name, address, phone, fax from vendors
```

The following User Language fragment binds name and fax number to %NAME and %FAXNO, and displays them after the variables are populated with the $DB_NEXTROW call:

```
%RESULT = $DB_BIND( %CONN1, 1, %NAME )
%RESULT = $DB_BIND( %CONN1, 4, %FAXNO )
REPEAT
   %RESULT = $DB_NEXTROW( %CONN1 )
   IF %RESULT LT Ø THEN
      LOOP END
   END IF
   PRINT %NAME AND %FAXNO
 ...
END REPEAT
```

4.9 $DB_CLOSE

$DB_CLOSE closes a connection to a remote server.

$DB_CLOSE accepts one argument and returns the number of connections closed.

```
%RC = $DB_CLOSE( connection_id )
```

$DB_CLOSE function

The single argument to $DB_CLOSE is the identifier for the connection to be closed. If the parameter is 0 or is not specified, all connections are closed.

%RESULT = $DB_CLOSE(%CONN)

In the above example the connection identified by the %variable %CONN is closed. If %CONN is not open %RESULT is set to 0.

```
Ø - connection is not open/no connections
    closed
>Ø - Number of connections closed
```

$DB_CLOSE return codes

4.10 $DB_COLLEN

$DB_COLLEN gets the maximum length for data in a column of a regular row.

$DB_COLLEN accepts 3 arguments and returns a numeric result code.

```
%RESULT = $DB_COLLEN( connection_id, -
             col_no, %variable )
```

$DB_COLLEN function

The first argument is the connection identifier. This is a required argument.

The second argument is the number of the column for which the length is requested.

The third argument is the %variable to be assigned the length for the column.

```
 0 - Target set
-1 - Connection lost
-2 - Not receiving rows
-4 - No such column
-5 - Invalid target variable
```

$DB_COLLEN return codes

%RESULT = $DB_COLLEN(%CONN1, 1, %LEN)

In the above example, %RESULT is set to 0, and %LEN is set to the maximum length of returned data for the first regular row column.

4.11 $DB_COLNAME

$DB_COLNAME gets the name of a column.

$DB_COLNAME accepts 3 arguments and returns a numeric code.

```
%RC = $DB_COLNAME( connection_id, -
          col_no, %variable )
```

$DB_COLNAME function

The first argument is the connection identifier.

The second argument is the number of the column for which the name is requested.

The third argument is the %variable which will contain the name of the column.

```
 0 - Target set
-1 - Connection lost
-2 - Not receiving rows
-4 - No such column
-5 - Invalid target variable
```

$DB_COLNAME return codes

%RC = $DB_COLNAME(%RMT_SERVER, %Y, %TOKEN)

In the above example, %RC is set to 0 and %TOKEN is set to the name of column %Y on connection %RMT_SERVER when $DB_COLNAME is called successfully.

4.12 $DB_COLTYPE

$DB_COLTYPE gets the type of a column.

$DB_COLTYPE accepts 3 arguments and returns a numeric result code, setting the variable specified in the third argument to the column type.

```
%RC = $DB_COLTYPE( connection_id, -
          col, %variable )
```

$DB_COLTYPE function

The first argument is the connection identifier.

The second argument is the number of the column for which the type is requested.

The third argument is the %variable to be assigned the column type.

```
     0 - Target set
    -1 - Connection lost
    -2 - Not receiving rows
    -4 - No such column
    -5 - Invalid target variable
```

$DB_COLTYPE return codes

The following call:

%DUMMY = $DB_COLTYPE(%MASTER, %COL, %TYPE)

returns a 0 in %DUMMY, and sets %TYPE to the data type of the column specified in %COL.

Data type returned by $DB_COLTYPE will be one of the following:

```
INT           DATETIME
SMALLINT      SMALLDATETIME
TINYINT       CHAR
FLOAT         VARCHAR
REAL          BINARY
MONEY         VARBINARY
TEXT          IMAGE
```

4.13 $DB_LANGPUT

$DB_LANGPUT sends a language request to the remote server.

$DB_LANGPUT accepts 2 arguments and returns a numeric condition code.

```
%RC = $DB_LANGPUT( connection_id, send_string )
```

$DB_LANGPUT function

The first argument is the connection identifier.

The second argument is the language data to be sent to the remote server.

```
>0 - Line number of sent line
-1 - Connection lost
-2 - Not in a send state
-3 - A non-language request was
     already started
```

$DB_LANGPUT return codes

The following code fragment initializes a connection to server MASTER and sends a language request to execute a stored procedure called IMPORTS.

```
%SYBASE = $DB_OPEN( 'MASTER',%LOGID,%PWD, -
   'UPCASE LOWCASE ERRCLOSE' )
%DUMMY  = $DB_LANGPUT( %SYBASE, 'EXEC IMPORTS' )
%RC     = $DB_WAIT( %SYBASE )
JUMP TO ( ROW_DATA, PARAMETERS ) %RC
 ...
```

4.14 $DB_MSGHANDLE

$DB_MSGHANDLE specifies the error handling subroutine.

$DB_MSGHANDLE accepts 1 argument and returns a numeric result code.

```
%RC = $DB_MSGHANDLE( subroutine_name )
```

$DB_MSGHANDLE function

The first and only argument is the name of a complex subroutine which will perform the handling of messages from remote server connections.

```
 Ø - Handler set
-1 - Complex subroutine not found
-2 - Subroutine does not have
     appropriate parameters
```

$DB_MSGHANDLE return codes

A call to $DB_MSGHANDLE clears any existing message handler definitions.

The following call:

%DUMMY = $DB_MSGHANDLE('MSGHANDLER')

returns a 0 in %DUMMY, and sets the client program to call the complex subroutine MSGHANDLER any time the remote server sends a message.

If no message handling subroutine is defined, messages from the remote server are directed to the user's screen.

The message handling subroutine must conform to the following format -- the %variable names can be changed, but the data type and definition is fixed by Janus:

```
SUBROUTINE MSGHANDLER (%CONNECT_IDENTIFIER    IS FIXED, -
   %SYBASE_MSG_NO         IS FIXED, -
   %SYBASE_MSG_STATE      IS FIXED, -
   %SEVERITY              IS FIXED, -
   %MSG_TEXT              IS STRING LEN 255, -
   %SERVER_NAME           IS STRING LEN  3Ø, -
   %RMT_PROCEDURE_NAME    IS STRING LEN  3Ø, -
   %ERROR_LINE_NO         IS FIXED          )
```

The %variables in the complex subroutine are automatically set when a message is received. Handling of the message situation is entirely up to the application coder. The variables have the following definitions (using the var names from the above sample):

```
%CONNECT_IDENTIFIER - connection identifier from $DB_OPEN
%SYBASE_MSG_NO      - Sybase message number
%SYBASE_MSG_STATE   - Sybase message state
%SEVERITY           - message severity
%MSG_TEXT           - message text
%SERVER_NAME        - name of server experiencing error
%RMT_PROCEDURE_NAME - name of procedure experiencing error
%ERROR_LINE_NO      - line number in proc causing error
```

The message handler need not have all the parameters specified. The string variables may be shorter than shown here, and values are truncated as necessary.

4.15 $DB_NEXTROW

$DB_NEXTROW retrieves the next row.

$DB_NEXTROW accepts 1 argument and returns a numeric result code.

```
%RC = $DB_NEXTROW( connection_id )
```

$DB_NEXTROW function

The first and only argument is the connection identifier. This is a required argument.

```
   >0 - Got an alternate row with indicated id
    0 - Got a regular row
   -1 - Connection lost
   -2 - Not receiving rows
   -3 - No more rows
```

$DB_NEXTROW return codes

Once a row is retrieved with $DB_NEXTROW, the column values are stored in the variables designated previously via $DB_BIND or $DB_ALTBIND.

The following code fragment shows how the return value from $DB_NEXTROW can be used to direct processing depending upon the data type received:

```
%ROW_TYPE = $DB_NEXTROW( %JANUS )
JUMP TO -
    ( ALT_ROW1, ALT_ROW2, ALT_ROW3, ALT_ROW4 ) -
       %ROW_TYPE
IF NOT %ROW_TYPE THEN
   JUMP TO REGULAR_ROW
ELSEIF %ROW_TYPE EQ -2 THEN
   JUMP TO PROGRAMMER_ERROR
ELSE
   JUMP TO COMPLETE
END IF
```

In this case, a calculated jump handles processing when the return code indicates an alternate row was received. Then a series of IF/ELSEIF's is used to direct processing for regular rows (return code 0) and error conditions. Note that if $DB_ONCLOSE was specified, the -1 condition need not be checked, because the ONCLOSE condition automatically handles it.

For -1 and -2 return codes, the connection is automatically closed if ERRCLOSE is specified on the $DB_OPEN.

4.16 $DB_NUMALT

$DB_NUMALT gets the number of alternate rows.

$DB_ accepts 1 argument and returns the number of alternate rows for the specified connection id.

```
%NUM = $DB_NUMALT( connection_id )
```

$DB_NUMALT function

The only input argument is the connection id for which the number of alternate rows is to be returned.

```
   >=0 - Got an alternate row with indicated id
    -1 - Connection lost
    -2 - Not receiving rows
```

$DB_NUMALT return codes

In the following example, %TOTAL_COMPUTED_ROWS is set to the number of alternate rows sent by the RPC “name_address”

```
%REMOTE = $DB_OPEN( 'VENDORS',,,'ERRCLOSE' )
%DUMMY  = $DB_RPCINIT( %REMOTE, 'name_address' )
%RC     = $DB_WAIT( %REMOTE )
 ...
%TOTAL_COMPUTED_ROWS = $DB_NUMALT( %REMOTE )
 ...
```

4.17 $DB_NUMALTCOL

$DB_NUMALTCOL gets the number of alternate columns.

$DB_ accepts 2 arguments and returns the number of columns for the specified alternate row on the specified connection.

```
%NAC = $DB_NUMALTCOL( connection_id, -
          alternate_row_id )
```

$DB_NUMALTCOL function

The first argument is the connection id for the specified alternate row.

The second argument is the id of the alternate row for which the number of columns is requested.

```
    >0 - Got an alternate row with indicated id
    -1 - Connection lost
    -2 - Not receiving rows
    -3 - No such alternate column
```

$DB_NUMALTCOL return codes

In the following example the values of the array %ALT_COLS are set to the number of columns in each alternate row returned by RPC “regional_eis”.

```
%ALT_COLS IS FLOAT ARRAY (10)
 ...
%REMOTE   = $DB_OPEN( 'VENDORS',,,'ERRCLOSE' )
%DUMMY    = $DB_RPCINIT( %REMOTE, 'regional_eis' )
%RC       = $DB_WAIT( %REMOTE )
 ...
FOR %X FROM 1 TO $DB_NUMALT(%REMOTE)
   %ALT_COLS(%X) = $DB_NUMALTCOL(%REMOTE, %X)
END FOR
```

4.18 $DB_NUMCOL

$DB_NUMCOL gets the number of columns for regular rows on the specified connection.

$DB_NUMCOL accepts 1 argument and returns the number of columns for a specified connection.

```
%NC = $DB_NUMCOL( connection_id )
```

$DB_NUMCOL function

The first and only argument is the connection id. This is a required argument.

```
    >0 - Number of columns
    -1 - Connection lost
    -2 - Not receiving rows
    -3 - No such row
```

$DB_NUMCOL return codes

In this example, %TOTAL_REGULAR_COLS is set to 5, because the select statement sent to connection %SYBASE generates 5 columns of information in a regular row.

```
%SYBASE   = $DB_OPEN( 'CUSTOMERS',,,'ERRCLOSE' )
%DUMMY    = $DB_LANGPUT( %SYBASE, -
'select name, address, city, state, -
zip from cust')
%RC       = $DB_WAIT( %SYBASE )
%TOTAL_REGULAR_COLS = $DB_NUMCOL( %SYBASE )
```

4.19 $DB_NUMRET

$DB_NUMRET gets the number of return parameters.

$DB_NUMRET accepts 1 argument and returns a numeric value.

```
%NR = $DB_NUMRET( connection_id )
```

$DB_NUMRET function

The only input argument is the connection id for which the number of return parameters is requested.

```
>=0 - Number of parameters returned by server
 -1 - Connection lost
 -2 - Don't have RPC return data
```

$DB_NUMRET return codes

In the following example, %RETURN_PARM_COUNT is set to the number of return parameters on connection %SYBASE.

%RETURN_PARM_COUNT = $DB_NUMRET(%SYBASE)

4.20 $DB_ONCLOSE

$DB_ONCLOSE specifies the action to take when the connection is lost.

$DB_ONCLOSE accepts 1 argument and returns a numeric code.

```
%RC = $DB_ONCLOSE( label )
```

$DB_ONCLOSE function

The only input argument is the label to which processing should jump when a connection to a remote server is lost.

When a jump destination label is specified it frees the programmer from having to check the “connection lost” return code on any $DB_ function, because processing will always shift to the ONCLOSE label.

The label specified in the $DB_ONCLOSE function is *scoped*, that is, if a single $DB_ONCLOSE is specified in the mainline of a program, all $DB_ function calls that result in -1 return codes will immediately transfer processing to that label, even if the $DB_ call is in a subroutine (technically, processing will return to back to the CALL statement, then transfer immediately to the label specified in the $DB_ONCLOSE function. If dropped connections detected by functions within subroutines are to be routed to a label within the subroutine, a $DB_ONCLOSE call should be made in the subroutine, specifying that label. The subroutine-specific label will not override the mainline label when for processing outside the subroutine.

A $DB_ONCLOSE call with no label or an invalid label will clear any existing ONCLOSE definition.

$DB_ONCLOSE cannot be issued from within an ON unit.

The label specified in $DB_ONCLOSE must be in the current scope and cannot be in a loop or an ON unit, nor can the label be subroutine.

```
 Ø - ONCLOSE handler set
-1 - Label not found in current scope
-2 - Target label is a subroutine
-3 - Can't issue from ON unit
-4 - Label is in a loop, ON unit or simple
     subroutine
```

$DB_ONCLOSE return codes

For example:

```
%RC = $DB_ONCLOSE( 'SERVER_OOPS' )
IF %RC LT Ø THEN
   JUMP TO PROGRAMMER_ERROR
END IF
 ...
SERVER_OOPS:
   PRINT 'Remote server dropped connection.'
   STOP
END
```

The ONCLOSE condition prints a message to the user and terminates the current procedure any time connection to the remote server is lost.

4.21 $DB_OPEN

$DB_OPEN establishes a connection with a server.

$DB_ accepts 4 arguments and returns a positive integer identifying the connection, or a negative number indicating an error condition.

```
%CONN = $DB_OPEN( server, userid, -
            password, options )
```

$DB_OPEN function

The first argument specifies the server to connect to. The target server must already have been defined to a port by a DEFINEREMOTE command.

The second argument specifies the userid to log onto the remote server. If no userid is specified, the current *Model 204* logon id is used.

The third argument specifies the password to use in logging on to the remote server.

The fourth argument is a list of options describing aspects of the connection. Valid options are as follows:

UPCASE Automatically converts row names, column names and parameter names from the remote server to upper case.

LOWCASE Converts outgoing identifiers to lowercase. This is useful when communicating with a SQL Server, where identifiers are typically in lower case and the 204 application is written using upper case only. Don't use this parameter when using mixed case names.

ERRCLOSE Specifying ERRCLOSE causes all errors to close the connection to the server. This spares you having to write error processing for each possible error condition a $DB_xxxxx function call may cause on a given connection.

NOCLOSE The default is for a connection to a remote server to be closed when a procedure completes. Specifying NOCLOSE on the $DB_OPEN causes that connection to remain open until it is closed with a $DB_CLOSE, or the user or site handler is bumped.

NOSITE If the port from which the Open Client application is initiating the connection is specified as an OPTSITE port, NOSITE indicates that a site handler is not to be used for the connection. That is, a basic client to server rather than a server to server connection should be established.

```
    >0 - Connection id of closed connection
    -1 - Server name missing
    -2 - Userid missing
    -3 - Invalid option
    -4 - No master port available
    -5 - Server not defined
    -6 - All connections on port in use
    -7 - Insufficient virtual storage
    -8 - Maximum connections exceeded
    -9 - Couldn't resolve target host
    -10 - Server port not responding
    -11 - Login failed
    -12 - Other error during connection
```

$DB_OPEN return codes

For example, when connecting to a server called “MASTER”:

```
%CONN = $DB_OPEN('MASTER','BART','SECRET' -
,'UPCASE LOWCASE ERRCLOSE')
```

User BART is logged on with password SECRET. Incoming identifiers are converted to upper case and outgoing identifiers are converted to lower case. ERRCLOSE is specified so that all $DB_xxxx function calls to this port that result in errors will close the connection.

4.22 $DB_RESULTS

$DB_RESULTS is a synonym for $DB_WAIT. This is supplied to provide compatibility with the Sybase DB-Library functions.

4.23 $DB_RETDATA

$DB_RETDATA gets the value of a return parameter.

$DB_RETDATA accepts 5 arguments and returns a numeric code, also setting its fourth argument to the value of the specified return parameter.

```
%RC = $DB_RETDATA( connection_id, parm_no, -
      parm_name, %variable, date_time_format )
```

$DB_RETDATA function

The first argument is the connection identifier.

The second argument is the number of the requested return parameter. If a parameter name is specified in the third argument, the second argument does not need to be set.

The third argument is the name of the requested return parameter. If the second argument (parm number) is specified, the third argument may be left blank.

The fourth argument is the %variable to which the incoming value is assigned.

The fifth argument is a datetime format for parameters that are Sybase DATETIME or SMALLDATETIME data types; it is ignored for other parameter types. See “Datetime Formats” on page 74 for an explanation of datetime formats.

If the parameter type is DATETIME or SMALLDATETIME, and no datetime format is specified, the default format of 'MON DD YYY HH:MI:SS' is assumed.

```
 0 - %variable set to specified parameter value
-1 - Connection lost
-2 - Don't have RPC return data
-3 - Return parameter not found
-4 - Invalid target variable
-5 - Conversion error
-6 - Invalid datetime format
```

$DB_RETDATA return codes

In the following example

%RC = $DB_RETDATA(%DBASE,,'INVOICE',%INVOICE)

%RC is set to 0 for successful calls, and %INVOICE is set to the value of the named parameter 'INVOICE'.

4.24 $DB_RETLEN

$DB_RETLEN gets the length of a return parameter.

$DB_RETLEN accepts 4 arguments and returns a numeric return code.

```
%RC = $DB_RETLEN( connection_id, parm_no, -
      parm_name, %variable )
```

$DB_RETLEN function

The first argument is the connection id.

The second argument is the number of the return parameter for which the length is requested. This argument is optional and does not need to be specified if a name is supplied as the third argument.

The third argument is the name of the return parameter for which the length is requested. This argument is optional and does not need to be specified if a number is supplied as the second argument.

The fourth argument is the %variable to which the return parameter length will be assigned.

```
 0 - Length set
-1 - Connection lost
-2 - Don't have RPC return data
-3 - Return parameter not found
-4 - Invalid target variable
```

$DB_RETLEN return codes

In the following example

%RC = $DB_RETLEN('MASTER',2,,%PARMLEN)

%RC is set to 0 for successful calls, and %PARMLEN is set to the length of the second argument on connection 'MASTER'.

4.25 $DB_RETNAME

$DB_RETNAME gets the name of a return parameter.

$DB_RETNAME accepts 3 arguments and returns a numeric code.

```
%RC = $DB_RETNAME( connection_id, -
          parm_no, %variable )
```

$DB_RETNAME function

The first argument is the connection identifier.

The second argument is the number of the parameter for which the name is requested.

The third argument is the %variable to which the parameter name will be assigned.

```
 Ø - Name set
-1 - Connection lost
-2 - Don't have RPC return data
-3 - Invalid parameter number
-4 - Invalid target variable
```

$DB_RETNAME return codes

In the following example

%RC = $DB_RETNAME('MASTER',4,%PARMNAME)

%RC is set to 0 for successful calls, and %PARMNAME is set to the name of the fourth argument on connection 'MASTER'.

4.26 $DB_RETSTATUS

$DB_RETSTATUS gets the status of latest remote procedure call.

$DB_RETSTATUS accepts 2 arguments and returns a numeric result code.

```
%RC = $DB_RETSTATUS( connection_id, %variable )
```

$DB_RETSTATUS function

The first argument is connection identifier.

The second argument is the %variable to which the RPC status will be assigned.

```
 Ø - Status set
-1 - Connection lost
-2 - Don't have RPC return data
-3 - Server didn't set return status
-4 - Invalid target variable
```

$DB_RETSTATUS return codes

In the following example

%RC = $DB_RETSTATUS(%CUSTOMERS, %STATUS)

%RC is set to 0 for successful calls, and %STATUS is set to the status of the last RPC requested on connection %CUSTOMERS.

4.27 $DB_RETTYPE

$DB_RETTYPE gets a return parameter type.

$DB_RETTYPE accepts 4 arguments and returns a numeric result code.

```
%RC = $DB_RETTYPE( connection_id, parm_no, -
      parm_name, %variable )
```

$DB_RETTYPE function

The first argument is the connection identifier.

The second argument is the number of the return parameter for which the type is requested. This argument is optional and does not need to be specified if a name is supplied as the third argument.

The third argument is the name of the return parameter for which the type is requested. This argument is optional and does not need to be specified if a number is supplied as the second argument.

The fourth argument is the %variable to which the return parameter type will be assigned.

```
 Ø - Type set
-1 - Connection lost
-2 - Don't have RPC return data
-3 - Return parameter not found
-4 - Invalid target variable
```

$DB_RETTYPE return codes

In the following example

%RC = $DB_RETTYPE(%CLIENTS, %NO, , %PARMTYPE)

%RC is set to 0 for successful calls. %PARMTYPE is set to the type of return data in parameter %NO on connection %CLIENTS. Return data types for parameters are

```
INT          DATETIME
SMALLINT     SMALLDATETIME
TINYINT      CHAR
FLOAT        VARCHAR
REAL         BINARY
MONEY        VARBINARY
```

4.28 $DB_RPCINIT

$DB_RPCINIT sets up to perform a remote procedure call.

$DB_RPCINIT accepts 3 arguments and returns a numeric result code.

```
%RC = $DB_RPCINIT( connection_id, -
          rpc_name, comp_status )
```

$DB_RPCINIT function

The first argument is the connection identifier.

The second argument is the name of the RPC being initialized. The RPC name may be up to 30 characters in length.

The third argument specifies whether the RPC should be recompiled at the remote server. A 0 (zero) or blank indicates that no recompilation should take place. Any other value specifies recompilation.

```
 Ø - RPC initialization successful
-1 - Connection lost
-2 - Not in a send state
-3 - A non-RPC request already started
-4 - An RPC is already started
-5 - RPC name missing or invalid
```

$DB_RPCINIT return codes

In the following example:

```
%RC = $DB_RPCINIT( %RS6000, 'ANNUALIZED_RETURN', Ø )
IF %RC LT Ø THEN
   JUMP TO ERRORS
END IF
%RC = $DB_SEND( %RS6000 )
%RC = $DB_WAIT( %RS6000 )
 ...
```

%RC is set to 0 for successful calls to $DB_RPCINIT. The RPC 'ANNUALIZED_RETURN' is sent to the remote server defined to connection %RS6000, and the program waits for the reply.

4.29 $DB_RPCPARAM

$DB_RPCPARAM adds a parameter to an RPC.

$DB_RPCPARAM accepts 6 arguments and returns a numeric status code. $DB_RPCINIT must be called before $DB_RPCPARAM can be called.

```
%RC = $DB_RPCPARAM( conn_id, parm_name, -
parm_value, in_out, type, len_or_CENTSPAN )
```

$DB_RPCPARAM function

The first argument is the connection identifier.

The second argument is the name of the RPC parameter. When communicating with a Sybase SQL Server, parameter names must be no greater than 30 characters and must begin with "@". If the name is left blank, the parameter is a *positional parameter*, that is, it can only be referenced by position number in both the client and the server programs. The position is determined by the number of times $DB_RPCPARAM has been successfully called against the RPC, so the first postional parameter is set by the first $DB_RPCPARAM call, the second set by the next call, etc.

The third argument is the value to which the parameter is set. Valid values for this argument are determined by the parameter type, set in the fifth argument.

If the fourth argument is blank or 0 (zero), the server may not return a value in this parameter (i.e. it is not an output parameter from the remote server's point of view). If the fourth argument is a value other than blank or 0, the remote server may return a value in the parameter; that is, it may be used as both an input and output parameter.

The fifth argument specifies the parameter type. Valid values are:

```
INT             DATETIME <format>
SMALLINT        SMALLDATETIME <format>
TINYINT         CHAR
FLOAT           VARCHAR
REAL            BINARY
MONEY           VARBINARY
```

If this parameter is left blank, Janus will automatically translate *Model 204* %variables as follows:

```
$DB_RPCPARAM variable    SQL Server receives as
STRING                   char
FIXED                    int
FLOAT                    float
```

The sixth argument specifies the maximum length for values of the RPC parameter, unless the type is DATETIME or SMALLDATETIME.

For date types:

- The fifth argument is either the keyword DATETIME or SMALLDATETIME, followed by the format of the datetime string value. See “Datetime Formats” on page 74 for an explanation of datetime formats.
- The sixth argument is a CENTSPAN value; the default is -50. See “CENTSPAN” on page 78 for an explanation of CENTSPAN processing.

```
 Ø - Parameter successfully set
-1 - Connection lost
-2 - RPC not initialized
-3 - Invalid parameter type
-4 - Invalid parameter max length
-5 - Conversion error
-6 - Invalid datetime format
-7 - Invalid CENTSPAN
```

$DB_RPCPARAM return codes

For example

```
%RC = $DB_RPCINIT(  %CUSTOMERS, 'REPORT_BY_STORE', )
%RC = $DB_RPCPARAM( %CUSTOMERS, 'STORE', %STORE, Ø, -
         'TEXT', 3Ø )
%RC = $DB_RPCPARAM( %CUSTOMERS, 'DATE_RANGE', %DATE, -
         Ø, 'TEXT', 16 )
%RC = $DB_WAIT( %RS6ØØØ )
 ...
```

The RPC 'REPORT_BY_STORE' is defined on a connection identified in %CUSTOMERS. Two parameters are defined for the RPC on this connection. The first, 'STORE', is assigned the value in variable %STORE. The fourth argument indicates the remote server may not return a value in the parameter, and the fifth and sixth arguments specify the type and length of the parameter.

The second argument is named 'DATE_RANGE', and is assigned the value in %DATE. Again, the remote server is not permitted to return a value in this parameter, and the type and length of the parameter are constrained by the fifth and sixth arguments. Then the RPC is sent ($DB_WAIT performs the send if a $DB_SEND has not yet been executed for the specified connection), and the client program waits for a response from the remote server.

4.30 $DB_SEND

$DB_SEND sends a request to the remote server.

$DB_SEND accepts 1 argument and returns a numeric condition code.

```
%RC = $DB_SEND( connection_id )
```

$DB_SEND function

The only argument accepted is the connection identifier. This is a required argument.

```
 0 - Request sent
-1 - Connection lost
-2 - Nothing to send
```

$DB_SEND return codes

For example:

```
%RC = $DB_LANGPUT(  %SHIPPING,  'EXECUTE EIS_SUMMARY' )
%RC = $DB_LANGPUT(  %INVENTORY, 'EXECUTE EIS_SUMMARY' )
%RC = $DB_SEND( %INVENTORY )
%RC = $DB_SEND( %SHIPPING  )
%RC = $DB_WAIT( %INVENTORY )
 ...
```

The same EXECUTE statement (EXECUTE EIS_SUMMARY) is sent to both a %SHIPPING and %INVENTORY connection, then the client program waits for a reply from the %INVENTORY connection.

If a $DB_WAIT is called before a $DB_SEND for the same port, the $DB_WAIT performs the $DB_SEND function. If the application only communicates with a single port there is no requirement to use $DB_SEND. $DB_SEND allows RPCs and language requests to be sent to a number of ports before the application waits for a reply from any of them.

4.31 $DB_SQL

$DB_SQL is a synonym for $DB_LANGPUT. It is supplied to maintain compatibility with the Sybase DB-library function calls.

4.32 $DB_TEST

$DB_TEST tests the connection to the remote server.

$DB_TEST accepts 1 argument and returns a numeric indicator of open status of the connection.

```
%RESULT = $DB_TEST( connection_id )
```

$DB_TEST function

The only input argument is the connection identifier. This is a required argument.

```
     0 - Connection exists
    -1 - Connection does not exist
```

$DB_TEST return codes

For example:

```
%STATUS = $DB_TEST( %MASTER )
IF %STATUS THEN
   PRINT 'NO CONNECTION'
END IF
 ...
```

The above code fragment checks to see if any connection exists for the identifier in %MASTER, and prints a warning if none is found.

The $DB_ONCLOSE jump is not taken for -1 return codes from $DB_TEST.

4.33 $DB_WAIT

$DB_WAIT sends any outgoing data, and waits for a reply from the remote server.

$DB_WAIT accepts 1 arguments and returns a numeric result code.

```
%RC = $DB_WAIT( connection_id )
```

$DB_WAIT function

The first and only argument is the connection identifier. This is a required argument.

```
     0 - Nothing to wait for, must send first
     1 - rows have been sent
     2 - return parameters/values have been sent
    -1 - Connection lost
```

$DB_WAIT return codes

If a $DB_RPCINIT or a $DB_LANGPUT is called, and no $DB_SEND has been called to send the pending RPC or language request to the server, the $DB_WAIT performs the send before placing the client program into a wait state. If you want to send RPCs or language requests to multiple connections before receiving the reply from any of them, $DB_SENDs must be coded for each connection before a $DB_WAIT is called. The $DB_WAIT is connection specific, so regardless of which remote server is ready to reply first, the client program will wait at the first $DB_WAIT it hits.

A $DB_WAIT must be executed before anything can be received from a remote server. The return code from the $DB_WAIT is used to determine whether the remote server returned parameters or rows.

```
%RC            = $DB_LANGPUT( %CARTOON, -
                     'EXECUTE ILLUSTRATOR_LIST' )
%RETURN_STATUS = $DB_WAIT( 'CARTOON' )
JUMP TO (PROCESS_ROWS, PROCESS_PARMS) %RETURN_STATUS
 ...
```

In the above code fragment the stored procedure ILLUSTRATOR_LIST is executed on connection CARTOON. Non-error return values (1 for rows being returned and 2 for parameters being returned) are directed to the appropriate label by a calculated JUMP statement.

APPENDIX A

Sample Open Client programs

A.1 Retrieve from SQL Server via Language Request

The following Janus/*Model 204* client application program sends a language request of a single EXECUTE statement to a SQL Server called RS_6000, and displays the return rows of informations with print statements.

```
*/ Janus Open Client sample program                    /*
*/ Sends a request to a SQL Server to run “sp_who”.    /*
*/ Remember to define our outgoing port to the remote  /*
*/ server so it knows where to send results back to.   /*
JANUS DELETE PORTO
JANUS DEFINE PORTO 3030 OPENSERV 5 MASTER CMD '*' -
UPCASE MSG204 9999
JANUS DEFREM PORTO RS_6000 RS6000 2040
JANUS START  PORTO
BEGIN
%ALT           IS FLOAT
%COLLEN        IS FLOAT
%I             IS FLOAT
%MAXALTCOL     IS FLOAT
%MAXCOL        IS FLOAT
%NUMCOL        IS FLOAT
%PRINT_ALTHDR  IS FLOAT
%PRINT_HDR     IS FLOAT
%RC            IS FLOAT
%SRV           IS FLOAT
%X             IS FLOAT
%Y             IS FLOAT
%ALTHEADER     IS STRING   LEN 132
%COLDATA       IS STRING   LEN  30
%COLID         IS STRING   LEN  30
%COLNAME       IS STRING   LEN  30
%COLOP         IS STRING   LEN  30
%COLTYPE       IS STRING   LEN  30
%ERROR         IS STRING   LEN   3
%FUNCTION      IS STRING   LEN  12
%HEADER        IS STRING   LEN 132
%IMAGE         IS STRING   LEN 132
%NAME          IS STRING   LEN  30
%PARSE         IS STRING   LEN 132
%RETSTATUS     IS STRING   LEN  10
%TYPE          IS STRING   LEN  10
%UID           IS STRING   LEN  10
%ALTCOL        IS STRING   LEN  40 ARRAY (40)
%COL           IS STRING   LEN  40 ARRAY (40)
%ALTWIDTH      IS FLOAT            ARRAY (40)
%WIDTH         IS FLOAT            ARRAY (40)
%PRINT_ALTHDR  = 1
%PRINT_HDR     = 1
* Define the subroutine to handle messages from
* the remote server:
%RC = $DB_MSGHANDLE( 'MSGHANDLE' )
IF %RC NE 0 THEN
CALL CLIENT_ERR( '$DB_MSGHANDLE', %RC )
END IF
* Define a label to jump to if the connection is
* unexpectedly lost:
%RC = $DB_ONCLOSE( 'ONCLOSE_ERROR' )
IF %RC NE 0 THEN
CALL CLIENT_ERR( '$DB_ONCLOSE', %RC )
END IF
* Open the connection:
```

```
%SRV = $DB_OPEN( 'RS_6000', 'homer', 'homer', 'ERRCLOSE' )
IF %SRV LE 0 THEN
CALL CLIENT_ERR ( '$DB_OPEN', %SRV )
END IF
* Initialize the connection as a language request by
* staging the name of the remote procedure to execute:
%RC = $DB_LANGPUT( %SRV, 'sp_who' )
WAITLOOP:
* Send the language request and wait for a reply:
%RC = $DB_WAIT( %SRV )
IF %RC LE 0 THEN
JUMP TO DONE
END IF
JUMP TO (ROWS, RETURN) %RC
IF %RC NE 0 THEN
CALL CLIENT_ERR( '$DB_WAIT', %RC )
END IF
ROWS:
* If row data was sent, collect the column header info:
%MAXCOL = $DB_NUMCOL( %SRV )
FOR %I FROM 1 TO %MAXCOL
%RC = $DB_COLNAME( %SRV, %I, %COLNAME )
%RC = $DB_COLTYPE( %SRV, %I, %COLTYPE )
%RC = $DB_COLLEN(  %SRV, %I, %COLLEN  )
%RC = $DB_BIND(    %SRV, %I, %COL(%I) )
IF %RC NE 0 THEN
CALL CLIENT_ERR( '$DB_BIND', %RC )
END IF
%WIDTH(%I) = $MAX( %COLLEN, $LEN(%COLNAME) ) + 1
%HEADER  = %HEADER WITH $PAD( %COLNAME, ' ', %WIDTH(%I) )
END FOR
* If alternate rows were send, collect their
* column header info:
%MAXALTCOL = $DB_NUMALT( %SRV )
FOR %ALT FROM 1 TO %MAXALTCOL
PRINT 'ALTERNATE ROW ' WITH %ALT
FOR %I FROM 1 TO $DB_NUMALTCOL(%SRV, %ALT)
%RC = $DB_ALTCOLNAME(%SRV, %ALT, %I, %COLNAME    )
%RC = $DB_ALTCOLTYPE(%SRV, %ALT, %I, %COLTYPE    )
%RC = $DB_ALTCOLLEN( %SRV, %ALT, %I, %COLLEN     )
%RC = $DB_ALTCOLOP(  %SRV, %ALT, %I, %COLOP      )
%RC = $DB_ALTCOLID(  %SRV, %ALT, %I, %COLID      )
%RC = $DB_ALTBIND(   %SRV, %ALT, %I, %ALTCOL(%I) )
IF %RC NE 0 THEN
CALL CLIENT_ERR( '$DB_ALTBIND', %RC )
END IF
%ALTWIDTH(%I)  = $MAX( %COLLEN, $LEN(%COLNAME) ) + 1
%ALTHEADER = -
%ALTHEADER WITH $PAD( %COLNAME, ' ', %ALTWIDTH(%I) )
END FOR
END FOR
* Loop around a $DB_NEXTROW call:
ROWLOOP:
%IMAGE = ''
%RC    = $DB_NEXTROW(%SRV)
IF %RC LT 0 THEN
JUMP TO WAITLOOP
```

```
END IF
IF %RC EQ Ø THEN
* Print the header the first time through:
IF %PRINT_HDR THEN
%PRINT_HDR = Ø
%IMAGE     = $PAD( ,'-',$LEN(%HEADER))
PRINT %HEADER
PRINT %IMAGE
END IF
* Print each row:
FOR %I FROM 1 TO %MAXCOL
%COL(%I) = $DEBLANK(%COL(%I))
%IMAGE = %IMAGE WITH $PAD( %COL(%I), ' ', %WIDTH(%I) )
END FOR
PRINT %IMAGE
END IF
IF %RC GT Ø THEN
* Print alternate row header first time through:
IF %PRINT_ALTHDR THEN
%PRINT_ALTHDR = Ø
%IMAGE        = $PAD( ,'-',$LEN(%ALTHEADER))
PRINT %ALTHEADER
PRINT %IMAGE
END IF
* Print each alternate row:
FOR %I FROM 1 TO %MAXALTCOL
%ALTCOL(%I) = $DEBLANK(%ALTCOL(%I))
%IMAGE = -
%IMAGE WITH $PAD( %ALTCOL(%I), ' ', %ALTWIDTH(%I) )
END FOR
PRINT %IMAGE
END IF
JUMP TO ROWLOOP
* Ignore return data:
RETURN:
%RC =  $DB_RETSTATUS(%SRV, %RETSTATUS)
IF %RC NE Ø THEN
CALL CLIENT_ERR( '$DB_RETSTATUS', %RC )
END IF
JUMP TO WAITLOOP
* Jump here when connection is lost:
ONCLOSE_ERROR:
PRINT -
'*-------------------------------------------------------*'
PRINT -
'* Remote server connection unexpectedly lost.  Cancelling   *'
PRINT '*   procedure execution. *'
PRINT -
'*-------------------------------------------------------*'
AUDIT -
'MSIR.AØØ1: Connection to remote server unexpectedly lost.'
STOP
* Jump here when finished.
DONE:
%RC = $DB_CLOSE
PRINT
PRINT -
```

```
'*-------------------------------------------------------------*'
PRINT -
   '* Session complete.  Number of connections closed: ' -
   WITH %RC
PRINT -
'*-------------------------------------------------------------*'
AUDIT -
   'MSIR.A00Ø2: Client run complete.  Connections closed = ' -
   WITH %RC
STOP
* Subroutine to handle $function error return codes.
SUBROUTINE CLIENT_ERR ( %FUNCTION IS STRING LEN 12, -
             %RC IS FLOAT )
PRINT -
'*-------------------------------------------------------------*'
PRINT -
   '*  Error return code from ' WITH %FUNCTION
PRINT '*  Return code = ' WITH %RC
PRINT -
'*-------------------------------------------------------------*'
AUDIT 'MSIR.A00Ø3: Application error: ' -
   WITH %FUNCTION WITH ' = ' WITH %RC
STOP
END SUBROUTINE
* Subroutine to handle messages returned by remote server.
SUBROUTINE MSGHANDLE( %CONNECT_IDENTIFIER IS FIXED, -
%SYBASE_MSG_NO           IS FIXED,          -
%SYBASE_MSG_STATE        IS FIXED,          -
%SEVERITY                IS FIXED,          -
%MSG_TEXT                IS STRING LEN 132, -
%SERVER_NAME             IS STRING LEN  3Ø, -
%PROCEDURE_NAME          IS STRING LEN  3Ø, -
%LINE_ERROR_NO           IS FIXED           )
PRINT
PRINT -
'*-------------------------------------------------------------*'
PRINT '* Remote server ' WITH %SERVER_NAME WITH -
      'reports the following:'
PRINT '*'
PRINT '*  Connection  ==> ' WITH %CONNECT_IDENTIFIER
PRINT '*  Procedure   ==> ' WITH %PROCEDURE_NAME
PRINT '*  Line number ==> ' WITH %LINE_ERROR_NO
PRINT '*  Severity    ==> ' WITH %SEVERITY
PRINT '*  MSG state   ==> ' WITH %SYBASE_MSG_STATE
PRINT '*  MSG number  ==> ' WITH %SYBASE_MSG_NO
PRINT '*  MSG         ==> ' WITH %MSG_TEXT
PRINT -
'*-------------------------------------------------------------*'
PRINT
AUDIT -
   'MSIR.A00Ø4: Message from remote server on connection ' -
   WITH %CONNECT_IDENTIFIER
AUDIT 'MSIR.A00Ø5: Message     = ' WITH %MSG_TEXT
AUDIT 'MSIR.A00Ø6: Procedure   = ' WITH %PROCEDURE_NAME
AUDIT 'MSIR.A00Ø7: Line number = ' WITH %LINE_ERROR_NO
END SUBROUTINE
END
```

APPENDIX B Datetime Processing Considerations

This chapter presents date processing issues, including usage of *Janus Open Client* past the year 1999, an explanation of its processing of dates, and any rules and restrictions you must follow to achieve correct results using date values with *Janus Open Client*.

Janus Open Client uses dates in the following ways:

- To examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *Janus Open Client* is under a rental or trial agreement
- As values communicated between an Open Client and an Open Server, using the Sybase DATETIME and SMALLDATETIME types
- As arguments to various other $functions, and returned values from them — in addition to the $DB_ functions, you may use the *Sirius Functions* to develop *Janus Open Client* applications. The *Sirius Functions* are incorporated into *Model 204* version 7.5 and later. See **http://m204wiki.rocketsoftware.com/index.php/List_of_$functions** for descriptions of those $functions, some of which process date values.

Please note that in addition to the above date processing performed by *Janus Open Client*, you can use it to transmit values between an Open Server and an Open Client, and you can use any number of Sirius $functions to manipulate values; any of these values might contain two digit year date values. The customer must ensure that any application using that data has an algorithm or rule for unambiguously determining the correct century for the values.

For headers on pages or rows that occur on printed pages or displayed screens, Sirius Software products generally use a full four-digit year format, although they may display dates with two-digit years in circumstances where the proper century can be inferred from the context.

You must examine all uses of date values in your applications to ensure that each of your applications produces correct results. Furthermore, both the operating system and *Model 204* must correctly process and transmit dates beyond 1999 in order for *Janus Open Client* to operate properly.

When a value of type **DATETIME** or **SMALLDATETIME** is transmitted between an Open Client and an Open Server, it is represented as a numeric value in units of 1/300 second since January 1, 1900 at 12:00 AM. When one of these values is sent from, or received by, *Janus Open Client*, it is converted from, or converted to, a representation containing

calendar date components such as year, month, and day. This is the only kind of explicit date value manipulation performed by the $DB_ functions; you may, however, use a number of datetime $functions (described in the documentation wiki at **http://m204wiki.rocketsoftware.com/index.php/List_of_$functions**) that have different modes of operation.

The rest of this chapter contains a discussion of datetime formats, valid datetime strings, and processing of two-digit year values. It also contains example datetime formats and corresponding example datetime strings.

B.1 Datetime Formats

The representation of a date is determined by a *datetime format*. This value is a character string, composed of the concatenation of tokens (for example, "YYYY" for a four-digit year, and "MI" for minutes) and separator characters (for example, "/" in "MM/DD/YY" for two-digit month, day, and year separated by slashes).

These *datetime format* strings are used in many products in addition to *Janus Open Client*. The products using datetime format strings are:

- *Fast/Unload*
- *Janus Open Client*
- *Janus Open Server*
- *Janus Specialty Data Store*
- *Janus Web Server*
- *SirDBA*
- *Sirius Functions*
- *Sir2000 Field Migration Facility*
- *Sir2000 User Language Tools*

The rules for these *datetime format* strings are consistent throughout all these products, though certain uses of these strings might impose extra restrictions. For example, a leading blank may match an HH, DD, or MM token in $DB_ function arguments, but it may not in some cases in other products.

There are certain rules applied to determine if a format is valid. The basic rules are:

1. If a format string contains a numeric datetime token (that is "ND", "NM", or "NS"), then the format string must consist of only one token. Numeric datetime tokens are only supported in format strings for the *Sir2000 Field Migration Facility*.

2. You must specify at least one time, weekday, or date token.

3. Except for "weekday", you can't specify redundant information. More specifically this means

    - Except for "I", no token can be specified twice.

- At most one year format (contains Y) can be specified.
- At most one month format (contains MON, Mon, or MM) can be specified.
- At most one day format (DD or Day) can be specified.
- At most one weekday format (WKD, Wkd, WKDAY, or Wkday) can be specified.
- If AM is specified, then PM can not be specified.
- At most one fractions-of-a-second format (contains X) can be specified.
- If DDD is specified, then neither a day nor month format can be.

4. If ZYY is specified in a format string, no other token that denotes a variable-length value may be used.

5. If a format string contains other tokens that denote variable length values, then an * token may only appear as the last character of the format string.

6. The DAY token may not be immediately followed by another token whose value may be numeric, regardless of whether the following token repsents a variable length value. Thus, DAY may not be followed by *, I, YY, YYYY, CYY, MM, HH, MI, SS, X, XX, or XXX; DAY may not be followed by a decimal digit separator, and DAY may not be followed by a quote followed by a decimal digit.

7. The maximum length of a format string is 100 characters.

Note: A common mistake is to use "MM" for minutes; it should be "MI".

The valid tokens in a date format are shown in the following list. In general, the **output format rule** for a token is shown, that is, the result when a DATETIME or SMALLDATETIME numeric value is converted to a datetime character string in a User Language %variable. The **input format rules** for $SIR_ functions are less strict; for example, all of the tokens that convert ***from*** an alphabetic string (for example, "MON") will allow any case string (for example, "jan" or "JAN" or "Jan").

NM numeric datetime value containing the number of milliseconds (1/1000 of a second) since January 1, 1900 at 12:00 AM. (This token is allowed only in the *Sir2000 Field Migration Facility*.)

NS numeric datetime value containing the number seconds since January 1, 1900 at 12:00 AM. (This token is allowed only in the *Sir2000 Field Migration Facility*.)

ND numeric date value containing the number of days since January 1, 1900. (This token is allowed only in the *Sir2000 Field Migration Facility*.)

***** Ignore entire variable-length substring matching pattern, if any. See "Datetime and format examples" on page 80.

I	Ignore corresponding input character. See “Datetime and format examples” on page 80.
"	Following character is "quoted", that is, it acts as a separator character. See “Datetime and format examples” on page 80.
YYYY	Four-digit year
YY	Two-digit year
CYY	Year minus 1900 (three digits, including any leading zero). See “Datetime and format examples” on page 80.
ZYY	Year minus 1900, two-digit or three-digit year number (variable length data). See “Datetime and format examples” on page 80.
MONTH	Full-month name (uppercase variable length). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as Month.
Month	Full-month name (mixed-case variable length). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as MONTH.
MON	Three-character month abbreviation (uppercase). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as Mon.
Mon	Three-character month abbreviation (mixed case). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as MON.
MM	Two-digit month number. When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as BM (leading blank is allowed). See “Datetime and format examples” on page 80.
BM	Two-character month number; when used as an argument to a $DB_ function for converting ***from*** a string, this is the same as MM. See “Datetime and format examples” on page 80.
DDD	Three-digit Julian day number
DD	Two-digit day number. When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as BD (leading blank is allowed). See “Datetime and format examples” on page 80.
BD	Two-character day number; when used as an argument to a $DB_ function for converting ***from*** a string, this is the same as DD. See “Datetime and format examples” on page 80.
DAY	One-digit or two-digit day number (variable length data). See “Datetime and format examples” on page 80.
WKDAY	Full day-of-week name (uppercase variable length). when used as an argument to a $DB_ function for converting ***from*** a string, this is the same as Wkday.
Wkday	Full day-of-week name (mixed-case variable length). when used as an argument to a $DB_ function for converting ***from*** a string, this is the same as WKDAY.
WKD	Three-character day-of-week abbreviation (uppercase). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as Wkd.
Wkd	Three-character day-of-week abbreviation (mixed case). When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as WKD.

HH Two-digit hour number. When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as BH (leading blank is allowed). See “Datetime and format examples” on page 80.

BH Two-character hour number; When used as an argument to a $DB_ function for converting ***from*** a string, this is the same as HH. See “Datetime and format examples” on page 80.

MI Two-digit minute number

SS Two-digit second number

X Tenths of a second

XX Hundredths of a second

XXX Thousandths of a second (milliseconds)

AM AM/PM indicator

PM AM/PM indicator

The valid separators in a date format are:

blank (" ")
apostrophe ("'")
slash ("/")
colon (":")
hyphen ("-")
back slash ("\")
period (".")
comma (",")
underscore ("_")
left parenthesis ("(")
right parenthesis (")")
plus ("+")
vertical bar ("|")
equals ("=")
ampersand ("&")
at sign ("@")
sharp ("#")
the decimal digits ("0" - "9").

In addition, any character may be a separator character if preceeded by the quoting character (").

See “Datetime and format examples” on page 80 for examples which include use of various separator characters.

B.2 Valid Datetimes

For a datetime string to be valid it must meet the following criteria:

- Its length must be less than 128 characters.
- It must be compatible with its corresponding format string.
- It must represent a valid date and/or time. For example, at most 23:59:59.999 for a time, 01-12 for a month, 01-31 or less (depending on the month) for a day, February 29 is only valid in leap years (only centuries divisible by 4 are leap years: 2000 is but neither 1800, 1900, nor 2100 are). Note: weekdays are not checked for consistency against the date; for example, both Saturday, 02/15/97 and Friday, 02/15/97 are valid.
- It must be within the date range allowed for the corresponding format. A datetime string used with a CYY or ZYY format can only represent dates from 1900 to 2899, inclusive. A datetime string used with a YY format can only represent dates in a range of 100 or less years, as determined by CENTSPAN and SPANSIZE. The valid range of dates for all other formats is from 1 January 1753 thru 31 December 9999.

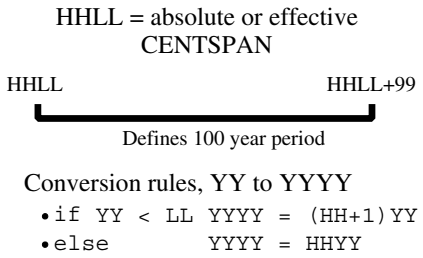
B.3 Processing Dates With Two-Digit Year Values

A date field with only two digits for the year value is capable of representing a range of up to one hundred years. When we compare a pair of two-digit year values we are accustomed to thinking of the century as fixed, so that all dates are either "19xx" or "20xx". However, a date field with two-digit year values can actually represent dates from two different centuries, provided that the *range* of dates does not exceed 100 years.

B.3.1 CENTSPAN

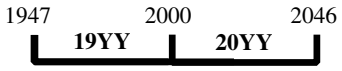
CENTSPAN provides a mechanism for unambiguously converting dates with two-digit year values into dates with four-digit year values. The CENTSPAN mechanism allows two-digit year values to span two centuries without confusion. CENTSPAN identifies the four-digit year value that is the *start* of a range of years represented by the two-digit year values.

CENTSPAN may be specified as an *absolute* unsigned four digit value between 1753 and 9999, or it may be specified as a *relative* signed value between -99 and +99, inclusive. A relative CENTSPAN value is dynamically converted to an *effective* absolute value before it is used to perform a YY to YYYY conversion. The effective CENTSPAN value is formed by adding the relative CENTSPAN to the current four-digit year value at the time the relative value is converted.



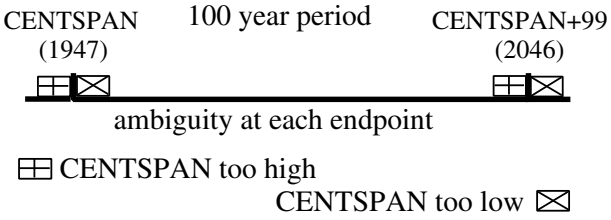
Example:

```
CENTSPAN = -50
current date = 1997
effective CENTSPAN = 1947
```



A simple algorithm is used to convert a two-digit year value (YY) to a four-digit year value, using a four-digit absolute or effective CENTSPAN value (HHLL). If the two-digit year value is less than the low-order two digits of the CENTSPAN value, then the resulting century is one greater than the high-order two digits of the CENTSPAN value. Otherwise the resulting century is the same as the high-order two digits of the CENTSPAN value.

Using all one hundred available years for mapping two-digit year values can cause significant confusion and result in data integrity errors: dates just above and just below the 100-year window are mapped to the other end of the window. From the previous example, the date "47" will be intepreted as 1947, when it could have conceivably been 2047. Similarly, the date "46" will be intepreted as 2046, when it might have been 1946.



If CENTSPAN is set to a value that is too high, dates that are just prior to CENTSPAN will appear to occur 100 years hence. If CENTSPAN is set to a value that is too low, dates that fall just after CENTSPAN+99 will appear to have occured 100 years earlier. A full one-hundred year window also can not detect attempts to represent more than one hundred years of values with a two-digit year.

B.3.2 SPANSIZE

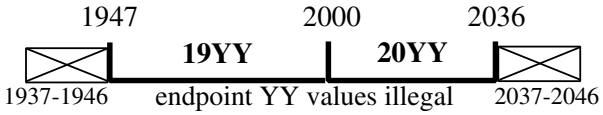
There is a method to protect from the ambiguities that can occur at each end of the 100-year window defined by CENTSPAN. SPANSIZE is used to restrict the size of the window used for mapping two-digit year values. The effect is to create two *guard bands*, one just below the date window and one just above. An attempt to represent a date value that lands in a guard band produces an error.

Each guard band contains CENTSPAN-SPANSIZE years, hence a SPANSIZE of 100 removes the protection. The default SPANSIZE is 90, which provides protection for two ten year windows: one below the CENTSPAN setting and one starting at

CENTSPAN+90. From our previous example:

Example:

```
CENTSPAN = -50
SPANSIZE = 90
current date = 1997
```



An attempt to represent the values "37" through "46" will be rejected. This protects the range 1937 through 1946 as well as the range 2037 through 2046. Note that an intended value of 2047, expressed as "47" will be accepted and interpreted as 1947. In general a smaller SPANSIZE provides the highest assurance of correct mappings. However, any setting of SPANSIZE less than 100 will probably detect the case where a range greater than one hundred years is being used.

B.4 Datetime and format examples

There is an extensive set of format tokens, as shown in "Datetime Formats" on page 74. These tokens and the various separator characters can be combined in almost limitless possibility, giving rise to an extremely large set of datetime formats. This section provides examples of some common datetime formats, and also tries to explain the use of some of the format tokens which might not be obvious. Each example format is explained and also presented with some matching datetimes; again, bear in mind that these tokens can be combined in very many ways and only a very few are shown here. It is assumed that these examples are invoked sometime between the years 1998-2040, as the basis for relative CENTSPAN calculations.

Note that in addition to the $DB_ functions, you may use the *Sirius Functions* to develop *Janus Open Client* applications (see **http://m204wiki.rocketsoftware.com/index.php/List_of_$functions**). The datetime handling in the *Sirius Functions* is more diverse than that that used in communicating values between an Open Client and an Open Server.

YYMMDD This is the common six-digit date format which supports sort order if all dates are within a single century. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '960229', 0, 'DATETIME YYMMDD')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of February 29th, 1996.

YYYYMMDD

This is the common eight-digit date format which supports sort order with dates in two centuries. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '19921212', 0, 'DATETIME YYYYMMDD')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of December 12th, 1992.

MM/DD/YY

This is the U.S. six-digit date format for display. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '12/14/94', 0, 'DATETIME MM/DD/YY')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of December 14th, 1994.

Notes - in the $DB_ functions:

- The leading zero corresponding to an MM token may be given as a blank, thus allowing " 7/15/98".
- The BM token can be used in an **input** format instead of MM.

DD.MM.YY

This is a European six-digit date format for display. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '14.12.94', 0, 'DATETIME DD.MM.YY')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of December 14th, 1994.

Notes — in the $DB_ functions:

- The leading zero corresponding to a DD token may be given as a blank, thus allowing " 7.04.89".
- The BD token can be used in an **input** format instead of DD.

Wkday, DAY Month YYYY "A"T HH:MI

This is a format which could be used for report headers. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   'Friday, 7 February 1998 ' WITH -
   'AT 21:33', Ø, 'DATETIME ' WITH -
   'Wkday, DAY Month YYYY "A"T HH:MI')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of February 7th, 1998, 9:33 PM.

Notes — in the $DB_ functions:

- If a format contains AM or PM, then the time (HH:MI) must be between 00:01 and 12:00 and must be accompanied by either AM or PM.
- If a format contains DAY (for example, "DAY MON YY"), the string matching it may have a leading zero, thus allowing "06 MAY 98".
- If a format contains HH, the string matching it may have a leading blank, thus allowing " 8:30".
- The BH token can be used in an **input** format instead of HH.

YYIIII This is a format which could be used for data which contains a 2-digit year prefixing other information, such as a sequence number. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '92ABCD', Ø, 'DATETIME YYIIII')
```

will set the DATETIME parameter named “ADD_DATE” to the Sybase representation of January 1st, 1992.

YY* This is a format which could be used for data which contains a 2-digit year prefixing other information, such as a sequence number, when the other information is variable length. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '92', Ø, 'DATETIME YY*')
%RC = $DB_RPCPARAM(%CON, 'EXP_DATE', -
   'ABC1992', Ø, 'DATETIME *YYYY')
```

will set both of the DATETIME parameters named "ADD_DATE" and "EXP_DATE" to the Sybase representation of January 1st, 1992.

Notes:

- At most one occurrence of the asterisk (*) token may appear in a datetime format.

CYYDDD This is a compact six-digit date format with explicit century information, from 1900 through and including 2899. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '097031', 0, 'DATETIME CYYDDD')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of January 31st, 1997.

ZYYMMDD

This is a compact six- or seven-digit date format with explicit century information, from 1900 through and including 2899, that can often be used with "old" YYMMDD date values in the 1900s. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '971201', 0, 'DATETIME ZYYMMDD')
%RC = $DB_RPCPARAM(%CON, 'EXP_DATE', -
   '1001201', 0, 'DATETIME ZYYMMDD')
```

will set the DATETIME parameters named "ADD_DATE" and "EXP_DATE", respectively, to the Sybase representations of December 1st, 1997 and December 1st, 2000.

Notes — in the $DB_ functions:

- A three digit number with a leading zero may correspond to a ZYY token, thus allowing "0971201".

YY0000 Decimal digits can be used as separator characters. The following User Language fragment

```
%RC = $DB_RPCPARAM(%CON, 'ADD_DATE', -
   '92000', 0, 'DATETIME YY000')
```

will set the DATETIME parameter named "ADD_DATE" to the Sybase representation of January 1st, 1992.

Notes:

- Numeric separators, unlike alphabetic separators, do not need to be preceeded by a quote character (").
- Numeric separators are available starting with version 5.2 of *Janus Open Client*.

B.5 $DB_ Functions CENTSPAN Argument

Some of the $DB_ functions accept an optional argument containing a CENTSPAN value to be used for the call. The default value of these CENTSPAN arguments is -50. The default value should be adequate in most cases; if you have carefully determined it should be different in some application, code the value on the relevant $function invocations.

For a different approach, see the description of the CENTSPLT and DEFCENT parameters (for example, **http://m204wiki.rocketsoftware.com/index.php/CENTSPLT_parameter**) and $function arguments.

Index

A

ADDCA, JANUS subcommand ... 12
ALLOCC parameter, JANUS DEFINE ... 16
Alternate rows ... 34
  defined ... 34

B

BINDADDR parameter, JANUS DEFINE ... 16
binding ... 34, 37, 42
  assigning column values to %variables ... 34
  defined ... 37, 42
BSIZE parameter, JANUS DEFINE ... 16

C

CENTSPAN ... 78, 84
CENTSPLT argument ... 84
CENTSPLT parameter ... 84
Certificate, SSL ... 20
  in SSL cache ... 22, 25
  requested by server ... 23
CHARSET parameter
  JANUS DEFINE ... 17
CHARSET, JANUS subcommand ... 12
Client certificate ... 20, 23
CLSOCK, JANUS subcommand ... 12
Compute rows ... 34
  defined ... 34
CONFIGURATION, JANUS subcommand ... 12
Connection limit ... 15
Content type
  client Post data ... 19-20

D

data types ... 42, 46
  returned by remote servers ... 42, 46
Date processing ... 73, 84
DEFCENT argument ... 84
DEFCENT parameter ... 84
DEFINE, JANUS subcommand ... 12
DEFINEIPGROUP, JANUS subcommand ... 12
DEFINEREMOTE, JANUS subcommand ... 12
DEFINEUSGROUP, JANUS
    subcommand ... 12
DELCA, JANUS subcommand ... 12
DELETE, JANUS subcommand ... 12
DELETEIPGROUP, JANUS
    subcommand ... 12
DELETEREMOTE, JANUS subcommand ... 12
DELETEUSGROUP, JANUS
    subcommand ... 13
DISPLAY, JANUS subcommand ... 13
DISPLAYCA, JANUS subcommand ... 13
DISPLAYREMOTE, JANUS
    subcommand ... 13
DISPLAYSOCK, JANUS subcommand ... 13
DISPLAYWEB, JANUS subcommand ... 13
DISPLAYXT, JANUS subcommand ... 13
DOMAIN, JANUS subcommand ... 13
DRAIN, JANUS subcommand ... 13

E

Encoding
  HTML form ... 19-20
Environment definition (overview) ... 10
Error messages ... 48
  handling messages from the remote
      server ... 48

F

FORCE, JANUS subcommand ... 13
FTP, JANUS subcommand ... 13

I

IBSIZE parameter, JANUS DEFINE ... 17
Installation ... 4
Introduction to JANUS ... 1

J

JANCAT ... 2
Janus Client $functions ... 31

Janus commands ... 34
introduction to ... 11
JANUS DEFINE ... 34
ERRCLOSE -- handling lost connections ... 34
wildcards used with ... 11
JANUS concepts ... 9
Janus functions ... 37-42, 44-47, 49-52, 54-61, 63-64
$DB_ALTBIND ... 37
$DB_ALTCOLID ... 38
$DB_ALTCOLLEN ... 39
$DB_ALTCOLNAME ... 40
$DB_ALTCOLOP ... 40
$DB_ALTCOLTYPE ... 41
$DB_BIND ... 42
$DB_CLOSE ... 44
$DB_COLLEN ... 44
$DB_COLNAME ... 45
$DB_COLTYPE ... 46
$DB_LANGPUT ... 47
$DB_MSGHANDLE ... 47
$DB_NEXTROW ... 49
$DB_NUMALT ... 50
$DB_NUMALTCOL ... 50
$DB_NUMCOL ... 51
$DB_NUMRET ... 52
$DB_ONCLOSE ... 52
$DB_OPEN ... 54
$DB_RESULTS ... 55
$DB_RETDATA ... 55
$DB_RETLEN ... 56
$DB_RETNAME ... 57
$DB_RETSTATUS ... 58
$DB_RETTYPE ... 59
$DB_RPCINIT ... 60
$DB_RPCPARAM ... 61
$DB_SEND ... 63
$DB_SQL ... 63
$DB_TEST ... 64
$DB_WAIT ... 64
JANUS parameters ... 10
TCPSERV ... 10
TCPTYPE ... 10
JANUS, introduction to
Janus IFDIAL Library ... 1
Janus TCP/IP Base ... 1

K

Keepalive connection, TCP ... 28

L

LANGUAGE parameter
JANUS DEFINE ... 17
LANGUAGE, JANUS subcommand ... 13
LIMITS, JANUS subcommand ... 13
LOADXR, JANUS subcommand ... 13

M

MASTER parameter, JANUS DEFINE ... 18
defining OPEN CLIENT ports ... 18
Mixed-case User Language ... 31
Model 204 resource requirements ... 16
buffer space requirements ... 16

N

NAMESERVER, JANUS subcommand ... 13
NOUPCASE parameter ... 18
Converting client data to upper case ... 18

O

OBSIZE parameter, JANUS DEFINE ... 18

P

Performance ... 17-18
Port, Janus
definition ... 14
Ports ... 9
PRELOGINUSER parameter, JANUS DEFINE ... 19

R

RAWINPUT parameter, JANUS DEFINE ... 19
RAWINPUTONLY parameter, JANUS DEFINE ... 20
RELOAD, JANUS subcommand ... 13
Remote server ... 31
return data ... 34
defined ... 34
row data ... 34
defined ... 34
RPC positional parameters ... 61

S

Sample code ... 67
  Open Client 1 (Language Request) ... 67
SDAEMON ... 10
  defined ... 10
SDAEMONs ... 10
  setting NSUBTKS ... 10
Server ports ... 9
SIRIUS file ... 4
Sirius Mods ... 4
SRVSOCK, JANUS subcommand ... 13
SSL certificate ... 23
  See also Certificate, SSL
SSL parameter, JANUS DEFINE ... 20
SSLBSIZE parameter, JANUS DEFINE ... 21
SSLCACHE parameter, JANUS DEFINE ... 22
SSLCIPH parameter, JANUS DEFINE ... 23
SSLCLCERT parameter, JANUS DEFINE ... 23
SSLCLCERTR parameter, JANUS DEFINE ... 23
SSLIBSIZE parameter, JANUS DEFINE ... 24
SSLMAXAGE parameter, JANUS DEFINE ... 25
SSLMAXCERTL parameter, JANUS DEFINE ... 25
SSLOBSIZE parameter, JANUS DEFINE ... 26
SSLPROT parameter, JANUS DEFINE ... 26
SSLSTATUS, JANUS subcommand ... 13
SSLUNENC parameter, JANUS DEFINE ... 27
START, JANUS subcommand ... 13
STATUS, JANUS subcommand ... 13
STATUSCA, JANUS subcommand ... 14
STATUSREMOTE, JANUS subcommand ... 14

T

TCP keepalives ... 28
TCPKEEPALIVE parameter, JANUS DEFINE ... 28
TCPLOG, JANUS subcommand ... 14
TIMEOUT parameter
  JANUS DEFINE ... 28
Timeouts, session ... 28
TNSERV port type
  TCPKEEPALIVE processing ... 28
TRACE parameter ... 29
TRACE, JANUS subcommand ... 14
Translation, character set ... 30
TSTATUS, JANUS subcommand ... 14

U

UL/SPF ... 4
UPCASE parameter ... 29
  Converting client data to upper case ... 29
User Language coding considerations ... 32, 34
  ERRCLOSE ... 34
    automatic handling of lost connections ... 34
  Open Client applications ... 32
Userid and password ... 18, 29
  Converting to upper case ... 18, 29

W

WEB, JANUS subcommand ... 14

X

XTAB parameter
  JANUS DEFINE command ... 30