
Janus Open Server Reference Manual

Model 204 Interoperability Products



Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, MA 02139

Telephone: (617) 876-6677

FAX: (617) 234-1200

E-mail: support@sirius-software.com

World Wide Web: <http://sirius-software.com>

August 3, 2010

© 2010 Sirius Software, Inc.



Proprietary Notices

The following products:

- *Fast/Backup*
- *Fast/Reload*
- *Fast/Unload*
- *Fast/Unload User Language Interface*
- *Janus Network Security*
- *Janus Open Client*
- *Janus Open Server*
- *Janus Sockets*
- *Janus Specialty Data Store*
- *Janus TCP/IP Base*
- *Janus Web Server*
- *SirDBA*
- *Sirius Functions*
- *Sirius Mods*
- *SirMon*
- *SirPro*
- *SirScan*
- *Sir2000 Field Migration Facility*
- *Sir2000 User Language Tools*
- *UL/SPF*

are proprietary products of Sirius Software, Inc.:

Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, Massachusetts 02139
USA

Model 204® is a proprietary product of Computer Corporation of America, a wholly-owned subsidiary of Rocket Software, Inc., which owns the trademark:

Rocket Software Corporate Office
M204 Division
275 Grove Street
Suite 3-410
Newton, Massachusetts 02466-2272
USA

Sybase, DB-Library, Omni SQL Server and Adaptive Server are registered trademarks of Sybase Inc.:

Sybase Inc.
6475 Christie Ave.
Emeryville, California 94608
USA

Contents

Proprietary Notices	iii
Contents	v
Summary of Changes	ix
Sirius Mods Version 6.0	ix
Sirius Mods Version 4.6	ix
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
Language Requests and RPCs	10
EXEC2RPC	11
Parameter checking	12
Environment Definition	13
Chapter 3: Janus Commands	15
JANUS command overview	16
JANUS DEFINE	18
ALLOCC	20
AUDTERM	20
BINDADDR xxx	20
BSIZE xxx	21
CHARSET xxx	21
CMD 'xxx'	21
EXEC2RPC	22
IBSIZE xxx	22
LANGUAGE xxx	23
MASTER	23
MSG204 xxx	23
MSG204L xxx	24
NOAUDTERM	24
NOUPCASE	24

OBSIZE xxx	25
OPEN list	25
PRELOGINUSER userid	26
RAWINPUT	26
RAWINPUTONLY	27
RPCONLY	27
SSL	27
SSLBSIZE xxxx	29
SSLCACHE xxxx	29
SSLCIPH xxx	30
SSLCLCERT and SSLCLCERTR	31
SSLIBSIZE xxxx	32
SSLMAXAGE xxx	32
SSLMAXCERTL xxx	33
SSLOBSIZE xxxx	33
SSLPROT xxx	34
SSLUNENC	35
TCPKEEPALIVE	35
TIMEOUT xxxx	36
TRACE xxx	36
UPCASE	37
XTAB table	37
Chapter 4: Janus Open Server \$Functions	39
\$SRV_BIND	41
\$SRV_CLOSE	43
\$SRV_DATA	44
\$SRV_DONE	47
\$SRV_LANGGET	48
\$SRV_MSG	48
\$SRV_NUMPARM	49
\$SRV_PARMGET	49
\$SRV_PARMLEN	51
\$SRV_PARMNAME	52
\$SRV_PARMNUM	52
\$SRV_PARMSET	53
\$SRV_PARMTYPE	54
\$SRV_RPCNAME	55
\$SRV_SENDRROW	56
\$SRV_SETROW	56
\$SRV_WAIT	58
Chapter 5: Open Server User Language Coding Considerations	59
Open Server User Language debugging	59

Appendix A: Sample Sybase Client Application	61
Appendix B: Sample Open Server Programs	69
Retrieve/Send data with \$SRV functions	69
Generic RPC router	72
Send U.L. procedure to client with \$SRV functions	74
Appendix C: Datetime Processing Considerations	77
Datetime Formats	78
Valid Datetimes	82
Processing Dates With Two-Digit Year Values	82
CENTSPAN	82
SPANSIZE	83
Datetime and format examples	84
\$SRV_ Functions CENTSPAN Argument	88
Index	89
Figures	
Figure 1: Model 204 TCP/IP Connectivity using Janus	2
Figure 2: JANUS DEFINE command syntax	18
Figure 3: \$SRV_BIND function.	41
Figure 4: \$SRV_BIND return codes	42
Figure 5: \$SRV_CLOSE function.	43
Figure 6: \$SRV_DATA function	44
Figure 7: \$SRV_DONE function.	47
Figure 8: \$SRV_DONE return codes	47
Figure 9: \$SRV_LANGGET function.	48
Figure 10: \$SRV_MSG function.	48
Figure 11: \$SRV_MSG return codes	48
Figure 12: \$SRV_NUMPARAM function.	49

Figure 13:	\$SRV_NUMPARAM return codes	49
Figure 14:	\$SRV_PARMGET function.	49
Figure 15:	\$SRV_PARMGET return codes	50
Figure 16:	\$SRV_PARMLEN function.	51
Figure 17:	\$SRV_PARMLEN return codes	51
Figure 18:	\$SRV_PARMNAME function.	52
Figure 19:	\$SRV_PARMNUM function.	52
Figure 20:	\$SRV_PARMNUM return codes	52
Figure 21:	\$SRV_PARMSET function.	53
Figure 22:	\$SRV_PARMSET return codes	54
Figure 23:	\$SRV_PARMTYPE function.	54
Figure 24:	\$SRV_PARMTYPE return codes	55
Figure 25:	\$SRV_RPCNAME function.	55
Figure 26:	\$SRV_SENDRROW return codes	56
Figure 27:	\$SRV_SETROW function.	56
Figure 28:	\$SRV_SETROW return codes	57
Figure 29:	\$SRV_WAIT function.	58
Figure 30:	\$SRV_WAIT return codes	58

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 Server* in version 6.0 of the *Sirius Mods*.

- DEBUG keyword replaced by TRACE
- AUDTERM replaced by NOAUDTERM as port default (JANUS DEFINE), and now applicable to OPENSERV ports

Sirius Mods Version 4.6

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

- Manual converted for layout changes

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.

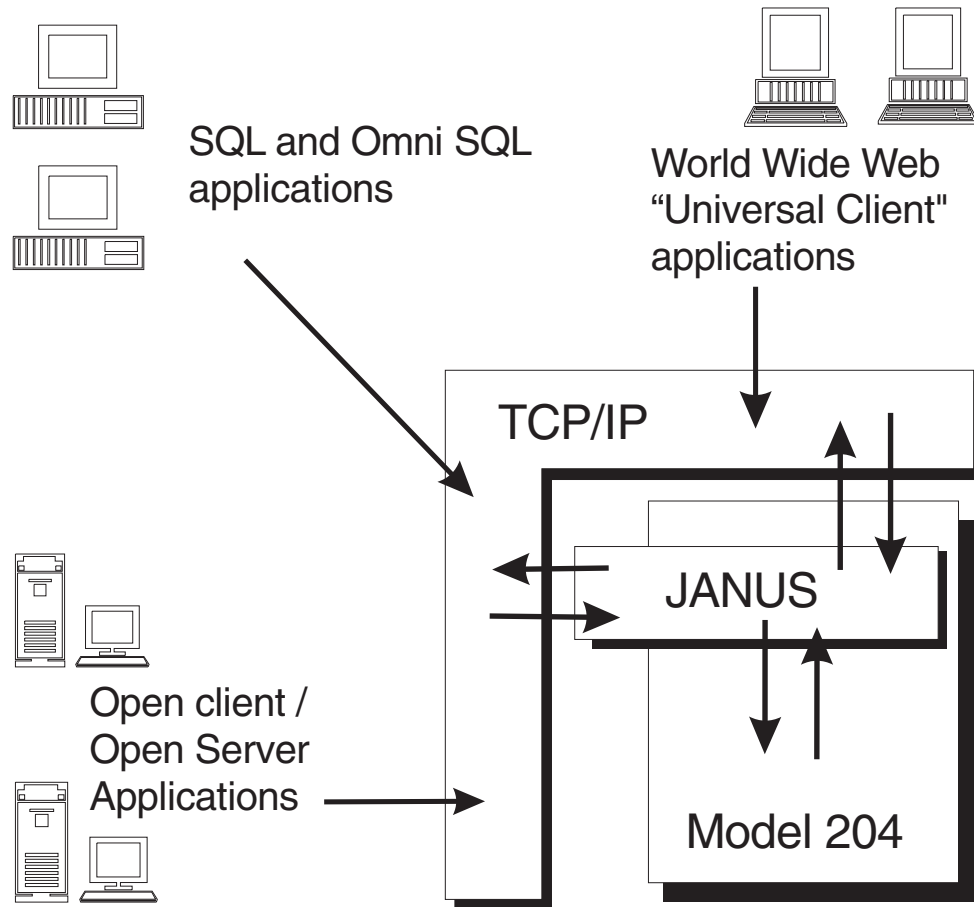
Janus Web Server

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.

The use of all Janus products involves the **JANUS *Model 204*** command, which is documented in the ***Janus TCP/IP Base Reference Manual***. 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 Server is part of the Janus family of products that provides connectivity to the *Model 204* database. A site that has *Janus Open Server* must have the *Janus TCP/IP Base* because without it, it is impossible to use *Janus Open Server*. A *Janus Open Server* 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 ***Janus Web Server Reference Manual***.

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.

These enhancements are distributed as components of the **Sirius Mods** and make up a collection of products including those in the Janus family. The *Sirius Mods* include many non-connectivity related products (such as *Fast/Backup*, *Fast/Reload*, and the *Fast/Unload User Language Interface*) that are not part of the Janus family. No *Sirius Mods* products are required to run *Janus Open Server* 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 (as of *Sirius Mods* version 6.8). which also contains code and data useful to Janus users including *Janus Open Server* 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 Server*, or any other Janus product, for that matter.

Thus, to install *Janus Open Server*, both the *Sirius Mods* and *UL/SPF* must be installed, following the instructions in the ***Sirius Mods Installation Guide*** and the ***UL/SPF Installation and Maintenance Guide***, respectively. When 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. And, as of version 6.8, 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.

Sirius Software has a strong commitment to backward compatibility with the *Sirius Mods*. This means that any User Language application (including *UL/SPF*) that uses the *Sirius Mods* will run 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.

This document, the ***Janus Open Server Reference Manual***, assumes that a site is running *Sirius Mods* version 6.7 or later and has installed *UL/SPF* version 6.2 or later. Any documentation that requires a later 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 of the *Sirius Mods* 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 Server*; that is, all versions since version 6.7 of the *Sirius Mods* and version 6.2 of *UL/SPF*.

1.3 Related manuals

As mentioned in “[Janus, the Sirius Mods, and UL/SPF](#)” on page 3, *Janus Open Server* requires the installation of both the *Sirius Mods* and *UL/SPF*. As such, the person responsible for the installation of *Janus Open Server* should refer to the ***Sirius Mods Installation Guide*** and the ***UL/SPF Installation and Maintenance Guide***. Also, the ***Sirius Messages Manual*** contains documentation on *Sirius Mods* error messages, so it might be useful to application programmers.

Also, as stated in [on page 3](#), *Janus Open Server* depends on the *Janus TCP/IP Base* product. Hence, there is much useful information in the ***Janus TCP/IP Base Reference Manual***.

You are authorized to use a number of Sirius \$functions along with *Janus Open Server*, 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 ***Sirius Functions Reference Manual***.

1.4 Related products

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

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 Server* (as well as other products in the Janus family).

One of the convenient debugging features available with *Janus Open Server* is a TRACE facility which logs Janus request/response information to the *Model 204* journal. In addition, most application debugging in a *Model 204* environment is done on a 3270 or 3270 emulator — but since *Janus Open Server* applications are not associated with a 3270, debugging techniques geared toward a 3270 will not work for them, so most of their debugging information goes 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 Server* 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 *Janus Open Server*, Horizon, BATCH2 and other Janus server applications.

1.5 System requirements

The current release of *Janus Open Server* 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 client communicating with *Janus Open Server*:

Sybase DB-Library Release 4 or later

any client software that uses TDS release 4 or later

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 Language Requests and RPCs

Janus Open Server is designed to communicate with clients that use the Sybase Open Client protocol. These clients are typically designed to work with a Sybase SQL Server or its equivalent. Because of some significant differences between Sybase SQL Server and *Janus Open Server*, the *Janus Open Server* programmer should be aware of some issues that can be ignored by a Sybase SQL Server programmer.

Specifically, the *Janus Open Server* programmer must be aware of the difference between *Language Requests* and *RPCs*. When a Sybase Open Client is communicating with a server it sends requests to the server. These requests will be either *Language Requests* or *RPCs*.

A *Language Request* is simply a stream of character data to be compiled by the server. These *Language Requests* are almost always SQL or Transact-SQL commands and statements. *Janus Open Server* provides no general facility to interpret *Language Requests*. It does, however, interpret a small subset of Transact-SQL statements when the EXEC2RPC parameter is specified in a port definition. *Language Requests* can also be retrieved in a User Language procedure (using \$SRV_LANGGET) and then interpreted by the User Language.

An *RPC* (Remote Procedure Call) is a stream of binary data that contains the name of a remote procedure and parameters for that remote procedure. *RPCs* require no parsing by a server and (generally) no compilation. Because of this, *RPCs* are the recommended way of issuing requests when performance is an issue, whether the server is a Sybase SQL Server or a *Janus Open Server*.

RPCs can come from one of three places. One, they can be explicitly coded in a C program using DB/Library. Second, they can be created by a third-party client package. Finally, they can be created from the compilation of an EXEC (or EXECUTE) Transact-SQL statement on a Sybase SQL Server. Because most third party clients do not generate *RPCs* and most applications are not coded in C, the most common source of *RPCs* is EXECUTE statements. Because of this, the term *RPC* is often used interchangeably to refer to EXECUTE statements. This is, in fact, incorrect. When a client sends an EXECUTE statement to a server that EXECUTE statement is actually part of a *Language Request*. When the server compiles this *Language Request* the EXECUTE statement is converted to an *RPC* that might be sent to another server at execution time.

The distinction between *Language Requests* and *RPCs* and the superior support for *RPCs* in *Janus Open Server* means that if possible, a client should be coded to send *RPCs* to the server. If this is not possible (because of limitations in the client software), then the server port should be set up with the EXEC2RPC parameter and EXECUTE statements should be used in the client requests. In any case, Transact-SQL statements not supported in the EXEC2RPC conversion of *Language Requests* should not be sent by the client unless User Language is written to parse and process these statements.

2.3 EXEC2RPC

The EXEC2RPC parameter for *Janus Open Server* ports allows a certain subset of language requests to be mapped to RPCs. The key to this mapping is the similarity between the Transact-SQL EXEC (or EXECUTE) statement and RPCs. This similarity is not, of course, coincidence. EXEC statements are converted into RPCs by the Transact-SQL compiler. For a language request to be convertible into an RPC, it must contain an implicit or explicit EXEC statement. In addition to the EXEC statement a few other Transact-SQL statements are acceptable in a language request that can be converted into an RPC. These statements are DECLARE, SELECT and SET. In addition, *Janus Open Server* accepts implied EXEC statements if the language request begins with an RPC. That is, the request

```
exec ulproc 18, 19
```

could be written

```
ulproc 18, 19
```

The primary limitation on the EXEC (or EXECUTE) statement is that only one is allowed per language request, whether implied or explicit. The following language request would be rejected by a *Janus Open Server* EXEC2RPC port because it contains two EXEC statements.

```
exec ulproc 18, 19
exec retrieve "elmo", 178945
```

All forms of the DECLARE statement are accepted, with the exception that certain datatypes are currently unsupported by *Janus Open Server*. The unsupported datatypes are *bit*, *nchar*, and *nvarchar*.

All forms of the SET statement are accepted, with the limitation that most set parameters have no effect on *Janus Open Server*. In fact, the only parameters that do anything on a *Janus Open Server* are *noexec*, *parseonly*, *nocount* and *rowcount*. The *noexec* and *parseonly* parameters are equivalent on a *Janus Open Server*.

The only form of the SELECT statement that is accepted on an EXEC2RPC port is the form that assigns a constant value to a variable. The following SELECT statements are all valid on an EXEC2RPC port :

```
select @grover = 22.17, @ernie = $12
select @count = "Transylvania"
select @snuffleupagus = 'Cabbages', @bert = 'oatmeal'
```

Any form of the SELECT statement that would result in data being returned to the client or data being retrieved from a database is invalid on an EXEC2RPC port. In addition, assignments from variables or expressions is also invalid on an EXEC2RPC port. The statement

```
select field1, field2
```

is invalid because it requires a retrieval of data from a database. The statement

```
select "Praire Dawn"
```

is invalid because it results in the return of data ("Praire Dawn") to the client. The statement

```
select @bert = @ernie
```

is invalid because the source of the assignment is a variable. The statement

```
select @cyranose = 12 + 18 * 2
```

is invalid because the source of the assignment is an expression (rather than a constant value).

In addition to the above limitations, no statements can follow an EXEC statement in an EXEC2RPC language request. That is, the following program is invalid because there is a set statement after the EXEC statement.

```
declare @cookie varchar(18)
select @cookie = 'allistair'
exec monster @cookie
set rowcount 999999
```

Finally, all limitations to Transact-SQL that apply when running against a SQL Server also apply when running against a Janus Open Server port. This means that a language request that runs against a *Janus Open Server* could be run against a Sybase SQL Server if the RPC specified on the EXEC statement is implemented on the Sybase SQL Server.

2.4 Parameter checking

Validity checking for RPC parameters in a *Janus Open Server* is the responsibility of the User Language application running on the EXEC2RPC port. For example, if a User Language application expects a return parameter called @kermit as the first parameter to an RPC called "bigbird", the following request would still be treated as valid by *Janus Open Server*.

```
exec bigbird @grover = 18 output
```

Whether the User Language running on a *Janus Open Server* port should do parameter validity checking is an issue that must be addressed on a site by site (and maybe application by application) basis. One philosophy might be that the individual(s) coding the client application want them to work correctly so it is up to these individuals to determine the correct parameters for any RPC and to suffer the consequences of any miscoding. This is reasonable as long as an RPC is only used by one application that is

written at the same time as the client, or if the same programmer or programming team is responsible for both the client application and the RPCs. If, on the other hand, an RPC is to be reused by many different client applications it might be worth the programming overhead to validate parameters.

2.5 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 Sirius functions and 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 ***Sirius Mods Installation Guide***.

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.

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 as of *Sirius Mods* version 6.3 are **JANUS** and **JANUSDEBUG**. For a full description of the Janus commands, see the *Janus TCP/IP Base Reference Manual*. The manual you are reading describes the commands and \$functions that are specific to *Janus Open Server*.

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. Exceptions are the **JANUSDEBUG** command, which can be issued by any logged-in user, and commands that are executed through the **JANMAN** subsystem. **JANMAN** is an optional application subsystem that ships with Janus and is described in the *Janus TCP/IP Base Reference Manual*. As of *Sirius Mods* version 6.2, 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?, WHERESTHEBEEF?, 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 the *Janus TCP/IP Base Reference Manual*.

Subcommand execution requires System Manager privileges, unless the command is executed through the JANMAN subsystem.

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 <i>Janus Open Client</i> , 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 <i>Janus Open Server</i> 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 <i>Janus Sockets</i> CLSOCK applications and <i>Janus Open Client</i> applications; only used with the IBM TCP/IP interfaces.
RELOAD	Reloads the <i>Model 204</i> -to-SQL mappings from the JANCAT file for a <i>Janus Specialty Data Store</i> 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 <i>Janus Web Server</i> processing rules.

3.2 JANUS DEFINE

The JANUS DEFINE command is used to specify the characteristics of a *Janus Open Server* 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 *Sirius 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. There is a discussion of server ports in the *Janus TCP/IP Base Reference Manual*.

type Port type. For *Janus Open Server* ports, specify OPENSERV.

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.

Before *Sirius Mods* version 6.8, the *maxcon* value had to be less than or equal to the number of sdaemons defined to the online. If you are defining multiple ports for your site, however, 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 Server*, 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.

Under *Sirius Mods* version and later, restrictions on the allowed values for *maxcon* are no longer present, 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

Once again, all the parameters allowed on the JANUS DEFINE command are documented in the *Janus TCP/IP Base Reference Manual*. For your convenience, only

those applicable to port definitions used with *Janus Open Server* 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 AUDTERM

This parameter is used to control whether the server thread sends “non-compiler terminal output” to the audit trail. Compiler terminal output is always sent to the audit trail. Any terminal output sent to the audit trail is sent as RK lines.

AUDTERM specifies that terminal output **is sent** to the audit trail; NOAUDTERM (“NOAUDTERM” on page 24), which is the default port setting, specifies that (non-compiler) terminal output **is not sent** to the audit trail.

Note that some “print output” can be “captured” on a Sirius \$list, a Janus Socket, or a USE output stream, and thus it would not be sent as terminal output -- to the audit trail or anywhere else. For further description of terminal output, Starting with version 6.0, this parameter applies to all Janus “server” port types, and the default setting is NOAUDTERM. Prior to this, the parameter only applied to WEB ports, and the default setting was AUDTERM.

This introduces a small incompatibility. Starting with version 6.0, compared to earlier versions, any WEB port connection without an explicit AUDTERM or NOAUDTERM will probably generate fewer audit trail lines, as will any SDS or OPENSERV port. This should be a benefit, since most of this output is either uninteresting or already logged to the audit trail as ER, AD or MS lines. Logging these messages as RK lines as well is just a waste of journal space and I/O and makes application diagnosis and debugging from the audit trail more difficult because of the extra noise data. For WEB, OPENSERV, or SRVSOCK applications that wish to explicitly audit information, the User Language AUDIT statement should be used, not the PRINT statement.

3.2.3 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.

3.2.4 **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.

3.2.5 **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.

3.2.6 **CMD 'xxx'**

This parameter specifies the *Model 204* commands to be executed after the files and groups specified in the OPEN parameter (“OPEN list” on page 25) are opened. Multiple commands must be separated by the word “AND”, and any command that contains blanks must be enclosed in quotes. Multiple commands in the CMD clause are only supported in version 6.0 and later of *Sirius Mods*.

CMD may span more than one line — continued with a hyphen (-) — but the total length of commands plus one overhead byte per command cannot exceed 255 bytes.

For an OPENSERV, SRVSOCK, or TNSERV port, the commands specified by CMD specify the processing performed for each connection to the port. If the user logs off and logs on again during the same connection, the CMD command(s) are not executed.

Because of this, using AUTOSYS is probably preferable to using CMD parameters for most TNSERV applications.

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.

3.2.7 EXEC2RPC

This parameter indicates that language requests should be converted to RPCs. The types of language requests that can be translated into an RPC is described in the *Janus Open Server Reference Manual*. EXEC2RPC implies RPCONLY. That is, it is not possible to set up a port so that Janus attempts to convert language requests into RPCs but if the attempt fails the language request is then made available to the User Language via \$SRV_LANGGET. If a language request is converted to an RPC, \$SRV_WAIT returns a 1 indicating that an RPC has been received. Thus, it is impossible for a User Language application on an EXEC2RPC port to tell if the current request was a “true” RPC or one that was generated via EXEC2RPC.

3.2.8 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.

3.2.9 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.

3.2.10 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 OPENSERVICES 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.

3.2.11 MSG204 xxx

This parameter specifies the Sybase message number to be used to return terminal output to the client. If this parameter is not specified, all terminal output for the open server will be sent to the *Model 204* audit trail. Because the Sybase client server communications protocol is a half duplex protocol, only terminal output that occurs while the client is in the “receive” state will be sent to the client. Terminal output that occurs while the client is in the “send” state will be sent to the audit trail regardless of the setting of the MSG204 parameter. A client is in the “receive” state after it has sent an RPC to the server and before it has received the DONE message for the request. The error state and error class for terminal output messages are both always set to 0.

3.2.12 **MSG204L xxx**

This parameter specifies the Sybase message number to be used to return terminal output to the client. This parameter is identical to the MSG204 parameter except that it indicates that the *Model 204* messages during logon will also be sent to the client. If MSG204 is specified instead of MSG204L, only the messages that occur after logon will be sent to the client.

3.2.13 **NOAUDTERM**

This parameter is used to control whether the server thread sends “non-compiler terminal output” to the audit trail. Compiler terminal output is always sent to the audit trail. Any terminal output sent to the audit trail is sent as RK lines.

NOAUDTERM, which is the default port setting, specifies that (non-compiler) terminal output **is not sent** to the audit trail; AUDTERM specifies that terminal output **is sent** to the audit trail.

Note that some “print output” can be “captured” on a Sirius \$list, a Janus Socket, or a USE output stream, and thus it would not be sent as terminal output -- to the audit trail or anywhere else. For further description of terminal output, Starting with version 6.0, this parameter applies to all Janus “server” port types, and the default setting is NOAUDTERM. Prior to this, the parameter only applied to WEB ports, and the default setting was AUDTERM.

This introduces a small incompatibility. Starting with version 6.0, compared to earlier versions, any WEB port that does not specify either AUDTERM or NOAUDTERM will probably generate fewer audit trail lines, as will any SDS or OPENSERV port. This should be a benefit, since most of this terminal output is either uninteresting or already logged to the audit trail as ER, AD or MS lines. Logging these messages as RK lines as well is just a waste of journal space and I/O and makes application diagnosis and debugging from the audit trail more difficult because of the extra noise data. For WEB, OPENSERV, or SRVSOCK applications that wish to explicitly audit information, the User Language AUDIT statement should be used, not the PRINT statement.

3.2.14 **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.

3.2.15 **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 maximum 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.

3.2.16 **OPEN list**

This parameter specifies the name of one or more *Model 204* files or groups to be opened when a server session is initiated.

If you specify multiple files or groups in an OPEN clause, they must be separated by an AND keyword. You can also specify individual file open privileges; if not, they default to X'0221'. Multiple files or groups and explicitly specified privileges in the OPEN clause are supported only in version 6.0 and later of *Sirius Mods*.

The syntax of each file or group specification is:

```
[FILE | GROUP] name [ [WITH] privs]
```

The first file or group listed in an OPEN clause is set as the default file or group context for the thread. If neither the keyword FILE nor the keyword GROUP is specified, OPEN looks first for a permanent group, then for a file, to open. It does not look for a temporary group, since one cannot yet exist at open time.

You can use the CMD parameter to specify a command to execute just after a file or group opens. If the CMD parameter specifies an INCLUDE command, the included procedure is assumed to come from the first file or group specified in the OPEN clause.

Examples of valid OPEN clauses follow:

```
JANUS DEFINE WEBXXX 80 WEBSERV 20 -
      OPEN WEBPROC AND -
      FILE DATAPROC WITH X'0761'
JANUS DEFINE OPENDOOR 1234 OPENSERV 40 -
      OPEN GROUP DOORPROC AND -
      FILE DOORDATA X'BFFF' -
      CMD 'I DRIVER'
```

3.2.17 PRELOGINUSER userid

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

This parameter is only available in version 6.0 and later of the *Sirius Mods*. Before this version, these were all true about processing that occurred before user login:

- It ran under “NO USERID”.
- It was not visible with *SirMon*, the MONITOR command, or LOGWHO.
- It was not BUMP'able.

After version 6.0 of the *Sirius Mods*, 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.

3.2.18 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.

This parameter is only available in version 6.7 and later of *Sirius Mods*.

3.2.19 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 26), 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 *Janus Web Server Reference Manual*.

This parameter is only available in version 6.8 and later of *Sirius Mods*.

3.2.20 RPCONLY

This parameter indicates that only RPC requests are allowed on the port. All non-RPC (language) requests are rejected. If the port is also defined as EXEC2RPC, an attempt is made to convert the language request to an RPC. If this is successful, the request is treated as a valid RPC request. If a port is defined as RPCONLY, the only valid return code from \$SRV_WAIT (on that port) is 1 meaning an RPC was received.

3.2.21 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 Sirius Software.

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 and described in the ***Janus TCP/IP Base Reference Manual*** 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 ***Janus Web Server Reference Manual*** include:

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

3.2.22 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 27](#)) 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.

Under version 6.0 and later of the *Sirius Mods*, 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 32](#)) and ([“SSLOBSIZE xxxx” on page 33](#)).

3.2.23 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 27](#)) 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 32](#)), 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 33](#)) 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 31](#)).

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.

3.2.24 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 27) 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.

3.2.25 SSLCLCERT and SSLLCERTR

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 27) 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.
- SSLLCERTR 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

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, which is described in the *Janus TCP/IP Base Reference Manual*.

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 SSLLCERTR 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 10 HTTPVERSION 1.1 -
      SSL JANSSL TM2008.PKEY SSLLCERTR

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'
```

The SSLCLCERT and SSLLCERTR parameters are only available in version 6.0 and later of *Sirius Mods*.

3.2.26 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 27) 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.

Before version 6.0 of the *Sirius Mods*, the size of the SSL input buffer was specified with the SSLBSIZE parameter (“SSLBSIZE xxxx” on page 29) because the SSLIBSIZE parameter was not 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.

3.2.27 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 27) 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. Before

version 6.0 of the *Sirius Mods*, only SSL V2 was supported and the SSLMAXAGE parameter was not available, so an implicit SSLMAXAGE value of 2 was always used.

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.

3.2.28 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 29).

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.

The SSLMAXCERTL parameter is only available in version 6.0 and later of the *Sirius Mods*.

3.2.29 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 27) 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.

Before version 6.0 of the *Sirius Mods*, the size of the SSL output buffer was always 256 bytes and the SSLOBSIZE parameter was not available.

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 32) on the other side.

3.2.30 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.

Before version 6.0 of the *Sirius Mods*, only SSL V2 was supported and the SSLPROT parameter was not available.

3.2.31 SSLUNENC

This parameter indicates that an unencrypted private key is being used in the certificate specified by the SSL parameter (“SSL” on page 27) on this Janus server port definition.

As of *Sirius Mods* version 6.2, this parameter is obsolete — as of this version, *Janus Network Security* 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.

Regardless of the *Sirius Mods* version, the use of unencrypted private keys is discouraged.

Before *Sirius Mods* version 6.2, SSLUNENC must have been specified 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 **is** used in the certificate specified on the SSL parameter, the SSLUNENC parameter **must not** be specified. Specifying SSLUNENC will 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*.

3.2.32 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.

This parameter is only available in *Sirius Mods* version 6.9 and later.

3.2.33 **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.

3.2.34 **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 the JANUS TRACE command in the *Janus TCP/IP Base Reference Manual*.

Note: The TRACE keyword was introduced in version 6.0 of the *Sirius Mods*. Before that, trace operations were controlled by the DEBUG keyword, which is no longer available as of version 6.5.

3.2.35 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.

3.2.36 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 was the only available option before *Sirius Mods* version 6.0.

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

Valid for all port types, the XTAB parameter is available in *Sirius Mods* version 6.0 and later.

Janus Open Server \$Functions

Once a *Janus Open Server* port is defined and started, Sybase clients can establish a connection to the port via Sybase DB-Library Open Client function calls. The Janus Server Functions (those prefixed with `$SRV_`) provide the facility by which the User Language server programs communicate back to the client.

When a connection is established, it is associated with an SDAEMON thread (if one is available). A *Model 204* logon is then attempted for the userid and password passed by the Sybase client. If the logon succeeds, the file specified in the OPEN clause of the JANUS DEFINE command, if any, is opened. After this the command specified by the CMD clause of the JANUS DEFINE command is executed as if the user had typed this command at a terminal.

At this point the Janus thread acts much the same as any other *Model 204* thread, with a few exceptions. First, because the thread has no terminal associated with it, it cannot issue any terminal input requests. Any terminal input requests result in a user restart due to a “lost connection.” Terminal output, on the other hand gets routed to the audit trail (as type RK journal entries). The other major difference between a Janus thread and any other thread is its ability to issue Janus server function calls.

Once a Janus thread is logged on, the thread is in a “receive” state. That is, it is up to the server to receive data from the client. The first server function a Janus thread should call is `$SRV_WAIT`. This function indicates that the server should wait for a request (an RPC or language request) from the client. `$SRV_WAIT` returns when a request is received. If the client request is an RPC, `$SRV_WAIT` reads the RPC input parameters into the thread's RPC buffer, and the thread switches to “send” state upon return from the `$SRV_WAIT`. This means that the server is now responsible for sending data back to the client until a `$SRV_DONE` is issued. After the `$SRV_DONE`, the server should issue another `$SRV_WAIT` if another RPC is to be handled.

The *Model 204* \$functions that work with and respond to Sybase RPCs are listed below and described in individual sections thereafter. Although *Sirius Mods* version 6.5 made mixed-case User Language available for use with Sirius Janus products, the *Janus Open Server Reference Manual* retains the all-uppercase presentation for \$function names and User Language entities. For more information about mixed-case User Language, see the *Janus SOAP Reference Manual*.

<code>\$SRV_BIND</code>	Bind a variable or literal to a column.
<code>\$SRV_CLOSE</code>	Close a connection.
<code>\$SRV_DATA</code>	Get the value of a parameter.
<code>\$SRV_DONE</code>	Send a “done” message to client.

\$SRV_LANGGET	Retrieve text of language request from client.
\$SRV_MSG	Send a message to the client.
\$SRV_NUMPARM	Return the number of RPC parameters.
\$SRV_PARMGET	Return the value of an RPC parameter.
\$SRV_PARMLEN	Return the length of an RPC parameter.
\$SRV_PARMNAME	Return the parameter name of a numbered RPC parameter.
\$SRV_PARMNUM	Return the parameter number of a named RPC parameter.
\$SRV_PARMSET	Set the value of an RPC parameter.
\$SRV_PARMTYPE	Return the datatype of an RPC parameter.
\$SRV_RPCNAME	Return the name of the RPC.
\$SRV_SENDRROW	Send a row of data (an image) to the client.
\$SRV_SETROW	Set the image for subsequent rows.
\$SRV_WAIT	Wait for a client request or action.

These functions can only be called by User Language programs running on a Janus server thread. If any of the Janus server functions are called by a user who is not running as a Janus server thread, the results of the function will be meaningless. (See the section on Open Server User Language Coding Considerations for instructions on how to debug a *Janus Open Server* application).

If any of the Janus server functions are called after the client has been disconnected, a user restart will occur. Because \$SRV_CLOSE causes the client to be disconnected, care should be taken that no further calls are placed to \$SRV_xxxxxx functions after any call to \$SRV_CLOSE.

Furthermore, as a user restart in APSY context will cause the server session to execute its APSY error procedure, care should be taken that no calls to \$SRV_xxxxxx functions should ever be placed in the APSY error procedure on a logical path that is followed for user restarts resulting from a lost connection. This is the same consideration programmers would give to not coding full-screen I/O in an APSY error proc when the error proc is invoked because of a lost connection.

4.1 **\$SRV_BIND**

`$SRV_BIND` binds the next column to a `%variable`, image item, or literal; it determines the source of a column value sent by `$SRV_SENDROW`.

`$SRV_BIND` accepts one required and three optional arguments and returns a numeric result code.

```
%RESULT = $SRV_BIND( variable_or_lit, -  
                    column name, type, -  
                    length_or_centspan )
```

\$SRV_BIND function.

Either this function or `$SRV_SETROW` (but not both) is used to bind the columns of rows sent by `$SRV_SENDROW`. `$SRV_BIND` is called once for each column in the order they occur in a row.

The binding is only valid for the duration of the request.

A *PACKED*, *ZONED*, *EFORMAT* or *DBCS* image item may not be bound to a column.

The maximum length of a column name is 30. If the name is omitted, no name is given to the column.

The third argument specifies the type of the column; it may be one of the following:

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

If the third argument is omitted, the following defaults are used:

- Float `%variables` or image items
Type=FLOAT Default len=8
- Other numeric `%variables` or image items
Type=INT Default len=4
- Numeric literals
Type=FLOAT Default len=8
- Other
Type=VARCHAR Default len=len of var or literal

The fourth argument specifies the length of the column, unless the type is DATETIME or SMALLDATETIME. The default length depends on the type, as shown in the table above.

For date types:

- The third argument is either the keyword DATETIME or SMALLDATETIME, followed by the format of the datetime string values. See [“Datetime Formats” on page 78](#) for an explanation of datetime formats.
- The fourth argument is a CENTSPAN value; the default is -50. See [“CENTSPAN” on page 82](#) for an explanation of CENTSPAN processing.

\$SRV_BIND may be called only before \$SRV_SENDRROW is called, or after \$SRV_DONE('MORE') is called and before \$SRV_SENDRROW is called.

<pre>0 - Bind successful 1 - Invalid call, RPC/language request not active 2 - Invalid call, already bound to row image by \$SRV_SETROW 3 - First argument missing, or invalid type of first argument, or invalid third or fourth argument 4 - Insufficient storage, as determined by RBSIZE. Each column requires 33 bytes plus the length of the column name; for a date type, an additional 1 byte plus the length of the date format is required. -100 - Connection lost</pre>
--

\$SRV_BIND return codes

A null value is sent by \$SRV_SENDRROW if the source value cannot be converted to the client representation. For example, the following User Language fragment will send a null date value:

```
%X = $SRV_BIND(%S, 'DUE', 'DATETIME MMDDYY')
%S = 'PIZZA'
%X = $SRV_SENDRROW
```

Example: In order to send rows consisting of a company name and fax number, the following User Language fragment binds columns to %COMPANY and %FAX and sends them to the client:

```
%RESULT = $SRV_BIND(%COMPANY, 'COMPANY')
%RESULT = $SRV_BIND(%FAX, 'FAX_NUMBER')
LBBL: FIND ALL RECORDS FOR WHICH ...
FRBL: FOR EACH RECORD IN LBBL
%COMPANY = COMPANY NAME
%FAX = FAX
%RESULT = $SRV_SENDRROW
END FOR
%RESULT = $SRV_DONE
%RESULT = $SRV_CLOSE
```

4.2 **\$SRV_CLOSE**

\$SRV_CLOSE terminates a client connection.

\$SRV_CLOSE accepts no arguments and always returns a 0.

<pre>%RESULT = \$SRV_CLOSE</pre>

\$SRV_CLOSE function.

Applications may be written so that at the completion of processing the server closes the connection with the client, as in the example code fragment that follows.

```
%Y = $SETROW('TEXASTEAS')
LBBL: FIND ALL RECORDS FOR WHICH ...
FRBL: FOR EACH RECORD IN LBBL
%TEXASTEAS:WELL.IDENT = WELL.IDENT
...
%X = $SRV_SENDRROW
END FOR
%X = $SRV_DONE
%X = $SRV_CLOSE
RETURN
```

No calls to any \$SRV_xxxxx function should ever be made after a call to \$SRV_CLOSE. Calls placed to any \$SRV_xxxxx function (including another call to \$SRV_CLOSE) after the client connection has been closed, will cause a user restart on the Janus server thread.

4.3 **\$SRV_DATA**

This retrieves the value of parameters associated with a connection, and optionally resets the value of certain parameters.

\$SRV_DATA accepts two arguments and returns a null string for any errors, or a string containing the value of the parameter.

<code>%RESULT = \$SRV_DATA(parm-name, new-value)</code>

\$SRV_DATA function

The first argument is the name of a parameter which might either have been passed by the Sybase client application or might be locally produced by the Janus server. This is a required argument.

The second argument is a value to which the first parameter should be set. This argument is optional.

Following is a list of valid parameters that may be viewed.

APPNAME

Name of application.

CHARNAME

Type of character.

HOSTNAME

Host processor identifier.

IPADDR

Client internet address in 'dotted' format (e.g. 204.0.0.1).

LANGNAME

Initial language.

NAME

The server name formatted as '*m204.jobname.portname*'. This is the name that is returned to the client message handling routine.

PROCESSID

Host process identification.

PROCNAME

Name of application.

PROGNAME

Client DB-Library name.

PROGVER

Client DB-Library version.

ROWCOUNT

The setting of the ROWCOUNT option (set via the SET statement) in an EXEC2RPC request. This option indicates the maximum number of rows to be sent by the server for any client request. A value of 0 indicates that there is no maximum.

RPCNAME

Remote Procedure Call name.

SERVNAME

Name of server.

USERNAME

Name of user.

Following is a list of parameters that may be viewed or set.

EXEC2RPC

When this parameter is set to 'ON', language requests are automatically converted into RPC's (where possible). The initial setting for the EXEC2RPC parameter is determined by the JANUS DEFINE command for the port. When EXEC2RPC is 'ON', the only possible return code from a \$SRV_WAIT is 1 (indicating RPC has been received).

MSG204

When this parameter is set to a non-zero value it indicates the Sybase message number to be used to return terminal output to the client. If this parameter is set to 0, all terminal output for the open server will be sent to the Model 204 audit trail. The initial setting for the MSG204 parameter is determined by the JANUS DEFINE command for the port. Because the Sybase client server communications protocol is a half duplex protocol,

only terminal output that occurs while the client is in the “receive” state will be sent to the client. Terminal output that occurs while the client is in the “send” state will be sent to the audit trail regardless of the setting of the MSG204 parameter. A client is in the “receive” state after it has sent an RPC to the server and before it has received the DONE message for the request. The error state and error class for terminal output messages are both always set to 0.

RPCONLY

When this parameter is set to 'ON', only RPC's are accepted on the connection. Any language requests converted because of the EXEC2RPC parameter are treated as RPC's for the purposes of the RPCONLY parameter, in fact EXEC2RPC effectively implies RPCONLY. The initial setting for the RPCONLY parameter is determined by the JANUS DEFINE command for the port. When RPCONLY is 'ON', the only possible return code from a \$SRV_WAIT is 1 (indicating RPC has been received).

UPCASE

When this parameter is set to 'ON', identifiers passed from the client are translated to upper case. This includes RPC names, parameter names and data returned by \$SRV_DATA. Parameter values are not translated regardless of the setting of the UPCASE parameter. This parameter is useful for writing applications that are not dependent on the case of client identifiers. It also helps bridge the gap between Model 204 where the natural representation of identifiers is in upper case and workstations where the natural representation is in lower case. Value is either 'ON' or 'OFF' and the initial value is always 'OFF'.

```
%RESULT = $SRV_DATA('IPADDR')
```

In the previous example, %RESULT is set to the value of the client's internet address.

```
%RESULT = $SRV_DATA('UPCASE', 'ON')
```

In the previous example, 'UPCASE' is set to 'ON', translating all parameter names to upper case.

```
%RESULT = $SRV_DATA('MSG204', 20001)
```

In the previous example, 'MSG204' is set to 20001 meaning that all terminal output on the open server (when the client is in receive mode) is to be sent to the client as Sybase message number 20001.

4.4 **\$SRV_DONE**

This tells the client that the server has completed its processing of the current request.

`$SRV_DONE` accepts one argument and returns a numeric code.

```
%RESULT = $SRV_DONE( parm1 )
```

\$SRV_DONE function.

The only argument is optional, and may contain either of the following strings, which are passed back to the client application:

ERROR

indicating that an error was encountered in server processing. Processing continues.

MORE

indicating that the server has completed sending the current table, and data for another table has yet to be sent.

```
0 - Successful termination/Server continues running  
1 - Invalid call, not in valid state to send data
```

\$SRV_DONE return codes

An RPC continues to be active until a `$SRV_DONE` is issued without 'MORE' specified. Once the `$SRV_DONE` is issued, a `$SRV_WAIT` must again be issued before another RPC can be processed.

```
%RESULT = $SRV_DONE
```

In the previous example, the client is informed that the current RPC is finished and no errors were encountered. After this call, the server must issue a `$SRV_WAIT` to wait for the next request.

```
%RESULT = $SRV_DONE('MORE')
```

In the previous example, the client is informed that the current set of rows (table) is finished, no errors were encountered and more data is to follow.

```
%RESULT = $SRV_DONE('MORE ERROR')
```

In the previous example, the client is informed that the current set of rows (table) is finished, there was an error processing the previous set of rows and more data is to follow.

4.5 **\$SRV_LANGGET**

This retrieves language input from client.

`$SRV_LANGGET` accepts one argument and returns the language request sent by a client.

```
%RESULT = $SRV_LANGGET( num_bytes )
```

\$SRV_LANGGET function.

The only argument is the number of bytes to retrieve. This parameter is required and must be less than 256.

```
%RESULT = $SRV_LANGGET(255)
```

In the previous example, `%RESULT` is set to the first 255 bytes of language data sent by the client.

4.6 **\$SRV_MSG**

This sends a message to the client.

`$SRV_MSG` accepts three arguments and returns a numeric code.

```
%RESULT = $SRV_MSG( msg_num, -  
                    error_class, msg_text )
```

\$SRV_MSG function.

The first argument is a message number which may be used when the message text or handling is performed in the client application. This is a required argument. Messages 1 through 9999 are reserved by Sybase, and users should not reuse these message numbers unless they replace them with identical meanings as they would have in a Sybase server.

The second argument is the error class of the message. The values of this parameter are application specific, and may be used by the client code to determine severity level or other handling characteristics. This argument is optional.

The third argument is message text. This is an optional argument.

```
0 - Message send successful.
1 - Invalid call, not in valid message state.
2 - Message number or error class invalid.
```

\$SRV_MSG return codes

The message number, error class and message text are passed to the client message handler.

```
%RESULT = $SRV_MSG(10472, 6, 'Invalid customer ID passed')
```

In the previous example, the client message handler is entered with the message 'Invalid customer ID passed', with message number 10472 and message class 6. The server name passed to the client is 'm204.jobname.portname'.

4.7 \$SRV_NUMPARAM

This returns the number of RPC parameters.

\$SRV_NUMPARAM accepts no arguments and returns a numeric code.

```
%RESULT = $SRV_NUMPARAM
```

\$SRV_NUMPARAM function.

In the above example, %RESULT is set to the number of parameters in the RPC.

```
>= 0 - Successful return.
-1 - Invalid call, RPC not active
```

\$SRV_NUMPARAM return codes

4.8 \$SRV_PARMGET

This gets the value of an RPC parameter.

\$SRV_PARMGET accepts four arguments and returns a numeric code.

```
%RESULT = $SRV_PARMGET( parm_no, parm_name, -
                        dest_variable, date_time_format )
```

\$SRV_PARMGET function.

The first argument is the number of the RPC parameter for which the function should return the value. This is an optional argument, but must be specified if the second argument is null.

The second argument is the name of the RPC parameter for which the function should return the value. This is an optional argument, but must be specified if the first argument is null. If both the first and second arguments are specified, the RPC parameter number specified by argument 1 is used if the indicated parameter number and all parameters before it are unnamed, that is, are positional parameters. Otherwise, the parameter name is used to locate the parameter.

The third argument is the name of a destination Model 204 %variable which will be assigned the value of the retrieved RPC parameter. This argument cannot be a constant, a field name or an expression. For example, 'TEST', 22 and %VAR1 + %VAR2 are examples of invalid values for the third argument.

The fourth argument is for parameters that are Sybase DATETIME or SMALLDATETIME types; it specifies the string datetime format that will be used for storing the result into the destination %variable specified as the third argument. The default format is 'MON DD YYYY HH:MI:SS'. See [“Datetime Formats” on page 78](#) for an explanation of datetime formats.

The fourth parameter is ignored if the incoming parameter is not a DATETIME or SMALLDATETIME type.

<pre>0 - Function successful. 1 - Invalid call, RPC not active. 2 - Parameter not found. 3 - Argument 3 is an invalid target. 4 - Data conversion error. 5 - Invalid datetime format.</pre>

\$SRV_PARMGET return codes

```
%RESULT = $SRV_PARMGET(1,, %PARM1)
```

In the previous example, the value of the first RPC parameter passed by the client is placed in %PARM1.

```
%RESULT = $SRV_PARMGET('CLIENT.ID', %CLIENT.ID)
```

In the previous example, the value of the RPC parameter passed by the client under the name 'CLIENT.ID' is placed in %CLIENT.ID.

```
%RC = $SRV_PARMGET(4,'START_DATE',%START_DATE,'YYYYMMDD')
```

In this example, the client passes the fourth parameter, named START_DATE; its value will be placed into %START_DATE as a string in YYYYMMDD format.

4.9 **\$SRV_PARMLEN**

This gets the length of an RPC parameter.

\$SRV_PARMLEN accepts three arguments and returns a numeric code.

```
%RESULT = $SRV_PARMLEN( parm_no, -  
                        parm_name, dest_variable )
```

\$SRV_PARMLEN function.

The first argument is the number of the RPC parameter for which the function should return the length. This is an optional argument, but must be specified if the second argument is null.

The second argument is the name of the RPC parameter for which the function should return the length. This is an optional argument, but must be specified if the first argument is null. If both the first and second arguments are specified, the RPC parameter number specified by argument 1 is used if the indicated parameter number and all parameters before it are unnamed, that is, are positional parameters. Otherwise, the parameter name is used to locate the parameter.

The third argument is the name of a destination Model 204 %variable which will be assigned the length of the retrieved RPC parameter. This argument cannot be a constant, a field name or an expression. For example, 'TEST', 22 and %VAR1 + %VAR2 are examples of invalid values for the third argument.

```
0 - Function successful.  
1 - Invalid call, RPC not active.  
2 - Parameter not found.  
3 - Argument 3 is an invalid target.
```

\$SRV_PARMLEN return codes

```
%RESULT = $SRV_PARMLEN(1,, %PARM1LEN)
```

In the previous example, the length of the first RPC parameter passed by the client is placed in %PARM1LEN.

```
%RESULT = $SRV_PARMLEN('CLIENT.ID', %LEN)
```

In the previous example, the length of the RPC parameter passed by the client under the name 'CLIENT.ID' is placed in %LEN.

For *char* or *varchar* datatypes, **\$SRV_PARMLEN** returns the maximum length of the string regardless of the current length. That is, if the specified parameter is defined as *varchar(18)* but currently is set to 'small', **\$SRV_PARMLEN** would return 18.

For all other datatypes, `$SRV_PARMLEN` simply returns the length of the binary representation of the datatype. For example, if the datatype is *smallint*, `$SRV_PARMLEN` returns 2; if the datatype is *float*, `$SRV_PARMLEN` returns 8.

4.10 **\$SRV_PARMNAME**

This returns the parameter name of a numbered RPC parameter. `$SRV_PARMNAME` accepts one argument and returns a string value.

```
%PARAM.NAME = $SRV_PARMNAME( parm_num )
```

\$SRV_PARMNAME function.

The first and only argument is the number of an RPC parameter for which the name is requested. Returned is either the name of the parameter, or null ("") to indicate no such parameter number, not a server or no RPC, or that the specified parameter is being passed by number and has no name.

```
%NAME1 = $SRV_PARMNAME(1)
```

In the previous example, the name of the first RPC parameter passed by the client is placed in `%NAME1`.

```
%RESULT = $SRV_PARMGET(, $SRV_PARMNAME(1), %VALUE1)
```

is the same as

```
%RESULT = $SRV_PARMGET(1, , %VALUE1)
```

4.11 **\$SRV_PARMNUM**

This returns the parameter number of a named RPC parameter.

`$SRV_PARMNUM` accepts one argument and returns a numeric code.

```
%RESULT = $SRV_PARMNUM( parm_name )
```

\$SRV_PARMNUM function.

The first and only argument is the name of an RPC parameter for which the parameter number is to be returned.

Returned is one of the following codes

<p>>0 - Parameter number (successful completion). 0 - Name missing or not found. 1 - Invalid call, RPC not active.</p>

\$SRV_PARMNUM return codes

```
%NUM1 = $SRV_PARMNUM('ADDRESS')
```

In the previous example, the number of the RPC parameter passed by the client under the name 'ADDRESS' is placed in %NUM1.

```
%RESULT = $SRV_PARMGET($SRV_PARMNUM('FNAME'), , %VALUE1)
```

is the same as

```
%RESULT = $SRV_PARMGET(, 'FNAME', %VALUE1)
```

4.12 \$SRV_PARMSET

This sets the value of an RPC parameter. \$SRV_PARMSET accepts five optional arguments and returns a numeric code.

<pre>%RESULT = \$SRV_PARMSET(parm_no, parm_name, - value, date_time_format, centspan)</pre>

\$SRV_PARMSET function.

The first argument is the number of the RPC parameter for which the function should set the value. This is an optional argument, but must be specified if the second argument is null.

The second argument is the name of the RPC parameter for which the function should set the value. This is an optional argument, but must be specified if the first argument is null. If both the first and second arguments are specified, the RPC parameter number specified by argument 1 is used if the indicated parameter number and all parameters before it are unnamed, that is, are positional parameters. Otherwise, the parameter name is used to locate the parameter.

The third argument is the value to which the RPC parameter should be set. This is an optional argument; if omitted the parameter is set to null. The parameter being set must have been defined in the client RPC call as an output parameter.

The fourth argument is the datetime format of the value being set, for parameters that are Sybase DATETIME or SMALLDATETIME data types. The default format is 'MON DD YYYY HH:MI:SS'. See [“Datetime Formats” on page 78](#) for an explanation of datetime formats.

The fifth argument is a CENTSPAN value; the default is -50. See “CENTSPAN” on page 82 for an explanation of CENTSPAN processing.

The fourth and fifth arguments are ignored if the parameter is not a Sybase DATETIME or SMALLDATETIME type.

`$$SRV_PARMSET` returns one of the following codes

<pre>0 - Function successful. 1 - Invalid call, RPC not active. 2 - Parameter not found. 3 - Parameter is not an output parameter. 4 - Conversion error. 5 - Invalid datetime format.</pre>

\$\$SRV_PARMSET return codes

```
%RESULT = $$SRV_PARMSET(1,, 22)
```

In the previous example, the first RPC parameter passed by the client is set to 22. This first parameter must have been indicated to be an output parameter by the client for this to work. When the RPC is completed (as indicated by `$$SRV_DONE`) the value 22 will be returned as the output value of the first RPC parameter.

```
%RESULT = $$SRV_PARMSET('ACTION', %ACTION)
```

In the previous example, the value of the RPC parameter passed by the client under the name 'ACTION' is set to the value in `%ACTION`.

4.13 \$\$SRV_PARMTYPE

This gets the datatype of an RPC parameter.

`$$SRV_PARMTYPE` accepts three arguments and returns a numeric code.

<pre>%RESULT = \$\$SRV_PARMTYPE(parm_no, - parm_name, dest_variable)</pre>
--

\$\$SRV_PARMTYPE function.

The first argument is the number of the RPC parameter for which the function should return the datatype. This is an optional argument, but must be specified if the second argument is null.

The second argument is the name of the RPC parameter for which the function should return the datatype. This is an optional argument, but must be specified if the first argument is null. If both the first and second arguments are specified, the RPC parameter number specified by argument 1 is used if the indicated parameter number and all parameters before it are unnamed, that is, are positional parameters. Otherwise, the parameter name is used to locate the parameter.

The third argument is the name of a destination Model 204 %variable which will be assigned the datatype of the retrieved RPC parameter. This argument cannot be a constant, a field name or an expression. For example, 'TEST', 22 and %VAR1 + %VAR2 are examples of invalid values for the third argument.

<p>0 - Function successful. 1 - Invalid call, RPC not active. 2 - Parameter not found. 3 - Argument 3 is an invalid target.</p>

\$SRV_PARMTYPE return codes

```
%RESULT = $SRV_PARMTYPE(1,, %PARM1TYPE)
```

In the previous example, the datatype of the first RPC parameter passed by the client is placed in %PARM1TYPE.

```
%RESULT = $SRV_PARMTYPE('CLIENT.ID', %TYPE)
```

In the previous example, the datatype of the RPC parameter passed by the client under the name 'CLIENT.ID' is placed in %TYPE.

The possible datatypes returned by \$SRV_PARMTYPE are *char*, *varchar*, *int*, *smallint*, *tinyint*, *float*, *real*, *money*, *smallmoney*, *text*, *image*, *binary* and *varbinary*. For the *char*, *varchar*, *text*, *image*, *binary* and *varbinary* datatypes, the length can be retrieved using the \$SRV_PARMLEN function.

4.14 \$SRV_RPCNAME

This returns the current Remote Procedure Call (RPC) name.

\$SRV_RPCNAME accepts no arguments and returns either the RPC name, or null (") to indicate that no RPC is active.

<pre>%RESULT = \$SRV_RPCNAME</pre>

\$SRV_RPCNAME function.

In the above example %RESULT is set to the name of the active RPC.

4.15 **\$SRV_SENDROW**

This sends a row of data to the client.

\$SRV_SENDROW accepts no arguments and returns a numeric code.

```
%RESULT = $SRV_SENDROW
```

\$SRV_SENDROW function.

%RESULT is set to a numeric code indicating the success of sending the row.

<pre>0 - Row sent successfully 1 - Invalid call, RPC/language request not active 2 - Invalid call, no current row image. 3 - Image not found. 100+n - Conversion error occurred for `n' columns in the row. Each such column was sent as a null value.</pre>
--

\$SRV_SENDROW return codes

\$SRV_SENDROW must be preceded by a \$SRV_SETROW or \$SRV_BIND to indicate the source of the data. \$SRV_SENDROW must be invoked from the same procedure as \$SRV_SETROW or \$SRV_BIND unless \$SRV_SETROW specified an image defined as PERM GLOBAL.

```
%RESULT = $SRV_SETROW('OUTREC') FOR EACH RECORD IN FIND1
%OUTREC:FNAME = FNAME %OUTREC:LNAME = LNAME %OUTREC:SSN = SSN
%RESULT = $SRV_SENDROW END FOR
```

In the previous example \$SRV_SENDROW sends the contents of the image 'OUTREC' to the client as a row of data. A row is sent for each record in the found set 'FIND1'.

4.16 **\$SRV_SETROW**

This sets the Model 204 image name for subsequent rows.

\$SRV_SETROW accepts one argument and returns a numeric code.

```
%RESULT = $SRV_SETROW( imagename )
```

\$SRV_SETROW function.

The image 'imagename' becomes the Model 204 IMAGE which contains the row of data to be sent to the client for all subsequent \$SRV_SENDRROW function calls.

The Model 204 image specified in \$SRV_SETROW must have been defined with the NAMESAVE option. This option is only available if Janus is installed and is, therefore, not documented in the *Model 204 User Language Reference Manual*.

```
IMAGE STIMPY NAMESAVE PART.NO IS BINARY LEN 4 COST IS BINARY LEN  
4 DESC IS STRING LEN 20 END FOR
```

In the previous example, the image named 'STIMPY' is defined with the NAMESAVE option and hence can be specified by the \$SRV_SETROW function.

If the Model 204 image specified in \$SRV_SETROW does not conform to certain rules, the \$SRV_SETROW will fail with a return code of 4. The rules for \$SRV_SETROW images are :

- Images cannot contain a DEPENDING clause (an image array element whose number of occurrence depends upon a value within the image itself).
- Images cannot contain an array whose number of occurrences is specified as UNKNOWN.
- Images cannot contain *PACKED*, *ZONED*, *EFORMAT* or *DBCS* datatypes.

%RESULT is set to one of the following values.

```
0 - Row successfully set to image name.  
1 - Invalid call, RPC/language request not active.  
2 - Invalid call, already have current row image or column binding.  
3 - Image invalid or not found.  
4 - Image has invalid format.
```

\$SRV_SETROW return codes

```
%RESULT = $SRV_SETROW('OUTREC1') %RESULT = $SRV_SENDRROW %RESULT  
= $SRV_DONE('MORE') %RESULT = $SRV_SETROW('OUTREC2') FOR EACH  
RECORD IN FIND1 %OUTREC2:FNAME = FNAME %OUTREC2:LNAME = LNAME  
%OUTREC2:SSN = SSN %RESULT = $SRV_SENDRROW END FOR %RESULT =  
$SRV_DONE
```

In the previous example \$SRV_SETROW sets the current output image to 'OUTREC1'. \$SRV_SENDRROW is then used to send the contents of image OUTREC1 to the client as

a row. `$$SRV_DONE` is issued to indicate that the server is done sending rows of `OUTREC1` but is not done with the RPC. The server then issues a `$$SRV_SETROW` to set the current output image to 'OUTREC2'. `$$SRV_SENDRROW` is used to send the contents of image `OUTREC2` to the client for each record in the found set called 'FIND1'. Finally, a `$$SRV_DONE` is issued to indicate that the RPC is done.

4.17 `$$SRV_WAIT`

This causes the server application to wait for a call from the client.

`$$SRV_WAIT` accepts no arguments and returns a numeric code.

```
%RESULT = $$SRV_WAIT
```

`$$SRV_WAIT` function.

The following `$$SRV_WAIT` return codes are possible

```
1 - RPC request.  
2 - Language request.
```

`$$SRV_WAIT` return codes

The example Model 204 server application at the end of this manual shows how `$$SRV_WAIT` is used in a driver routine that processes a number of RPCs within a single procedure.

`$$SRV_WAIT` must be called before an RPC or language request can be processed. The `$$SRV_WAIT` is satisfied when a client request is received. Processing then continues until a `$$SRV_DONE` is issued. After a `$$SRV_DONE` is issued, the server *must* again issue a `$$SRV_WAIT` before it will be able to process another RPC request.

Open Server User Language Coding Considerations

Coding Janus server applications is not very different from coding any other User Language application. The main difference is that Janus applications run on TCP/IP threads that are not attached to a full screen I/O device. Instead, the client application performs all screen formatting, and the Janus server application communicates with the client using \$SRVxxx functions rather than READ SCREEN statements.

Any attempt to perform full screen I/O in a Janus session will result in a user restart. No READ SCREENS should be employed in Janus applications, and programmers should make sure that all \$READs and dummy string substitutions are satisfied without prompts being sent to the (non-existent) screen.

```
JANUS DEFINE REGION1 1001 OPENSERV 20 OPEN FILE PRODPROC -  
CMD 'I REGIONAL.QUERIES,1,HIGH'  
JANUS START REGION1  
  
PROCEDURE REGIONAL.QUERIES  
BEGIN  
...  
FDX: IN DATAFILE FD REGION = ??REGNUM  
PRIVILEGE = ??PRIV  
...
```

In the previous example, the two global FIND criteria are satisfied by values concatenated onto the INCLUDE statement specified in the DEFINE for the port.

When coding ON units in Janus server applications, programmers should keep in mind that any “DO YOU REALLY WANT TO...” messages will cause a user restart of the server session because they send a prompt to the non-existent full screen device. All error, enqueueing and locking conditions should be handled so as not to cause these messages.

5.1 Open Server User Language debugging

Because full screen I/O cannot be performed in a Janus application, the primary source of information for debugging User Language server programs is the *Model 204* journal. A journal formatting and scanning tool, like *SirScan*, is highly recommended.

User Language PRINT and AUDIT statements are routed to the *Model 204* journal, along with any other messages that would ordinarily appear at a user's terminal.

APPENDIX A *Sample Sybase Client Application*

The following Sybase client application program demonstrates the use of DB-LIBRARY open client code to pass a request to a *Janus/Model 204 Janus Open Server* example that follows.

```
#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>
#define USER      "sa"
#define PASSWORD  ""
#define LANGUAGE  "us_english"
#define SQLBUFLN  255
#define ERR_CH    stderr
#define OUT_CH    stdout
extern void      error();
extern int       err_handler();
extern int       msg_handler();
/* forward declarations of the error/message handler routines. */
int             err_handler( );
int             msg_handler( );

char null_str[10] = "nuttin" ;
main()
{
LOGINREC        *login;
DBPROCESS       *dbproc;

int             i, j, numcols, collen, totlen;
int             rc;
char            data[20][80];
int             sel_lat1, sel_lat2, sel_long1, sel_long2;
int             min_lat, max_lat, min_long, max_long;
char            *ptr;
char            display_buf[512];
char            *display_ptr;
DBCHAR          database[31];
DBCHAR          username[31];
DBINT           indicator ;
DBCHAR          unit_id[81];
DBCHAR          unit_name[80] ;

/* Initialize DB-Library. */

/* Install the user-supplied error-handling and message-handling
 * routines. They are defined at the bottom of this source file.
 */
dberrhandle(err_handler);
dbmsghandle(msg_handler);

/* init dblib */
if (dbinit() == FAIL)
exit(ERREXIT);

/* Allocate and initialize the LOGINREC structure to be used
 * to open a connection to a server.
 */

if ((login = dblogin( )) == NULL)
{
printf("dblogin failed\n") ;
fflush(stdout) ;
dbexit() ;
```

```

exit(-1) ;
}

/* print the version of dblib */
printf("%s\n", dbversion()) ;

/* set the username and pwd for logging in to the server */
DBSETLUSER(login, "homer");
DBSETLPWD(login, "homer");

if ((dbproc = dbopen(login, (BYTE *)NULL)) == NULL)
{
printf("dbopen failed\n") ;
fflush(stdout) ;
dbexit() ;
exit(-1) ;
}

dbrpcinit(dbproc, "WELL_RANGE", 0);
dbrpcparam(dbproc, "sel.lat.min", DBRPCRETURN, SYBINT4, -1, -1,
&min_lat);
dbrpcparam(dbproc, "sel.lat.max", DBRPCRETURN, SYBINT4, -1, -1,
&max_lat);
dbrpcparam(dbproc, "sel.long.min", DBRPCRETURN, SYBINT4, -1, -1,
&min_long);
dbrpcparam(dbproc, "sel.long.max", DBRPCRETURN, SYBINT4, -1, -1,
&max_long);
dbrpcsend(dbproc);
dbsqlok(dbproc);

if (rc = dbresults(dbproc) != SUCCEED) {
printf("Unexpected result from WELL_RANGE\n");
Goto close;
}

if (rc = dbresults(dbproc) != NO_MORE_RESULTS) {
printf("Unexpected result from WELL_RANGE\n");
Goto close;
}

min_lat = * (int *) dbretdata(dbproc, 1);
max_lat = * (int *) dbretdata(dbproc, 2);
min_long = * (int *) dbretdata(dbproc, 3);
max_long = * (int *) dbretdata(dbproc, 4);

printf("Min latitude = %9d   Max latitude = %9d\n", min_lat ,
max_lat );
printf("Min longitude = %9d   Max longitude = %9d\n", min_long,
max_long);

for (;;) {

dbrpcinit(dbproc, "WELL_LIST", 0);

printf("Bottom left corner latitude = ");
rc = getn(&sel_lat1, min_lat, max_lat);
if (rc < 0) break;

```

```
if (rc > 0)
dbrpcparam(dbproc, "sel.lat1", 0, SYBINT4, -1, -1, &sel_lat1);
else sel_lat1 = min_lat;

printf("Bottom left corner longitude = ");
rc = getn(&sel_long1, min_long, max_long);
if (rc < 0) break;
if (rc > 0)
dbrpcparam(dbproc, "sel.long1", 0, SYBINT4, -1, -1, &sel_long1);
else sel_long1 = min_long;

printf("Top right corner latitude   = ");
rc = getn(&sel_lat2, sel_lat1, max_lat);
if (rc < 0) break;
if (rc > 0)
dbrpcparam(dbproc, "sel.lat2", 0, SYBINT4, -1, -1, &sel_lat2);

printf("Top right corner longitude   = ");
rc = getn(&sel_long2, sel_long1, max_long);
if (rc < 0) break;
if (rc > 0)
dbrpcparam(dbproc, "sel.long2", 0, SYBINT4, -1, -1, &sel_long2);
dbrpcsend(dbproc);
dbsqllok(dbproc);

/* Process the results */

while ((rc = dbresults(dbproc)) != NO_MORE_RESULTS)
{
if (rc == FAIL)
{
printf("ERROR ON DBRESULTS!!!!!!!!!!!!!!!!!!!!\n");
Goto close;
}
printf("\n");
numcols = dbnumcols(dbproc);
if (numcols > 20) numcols = 20;
totlen = 0;
for (i = 1; i <= numcols; i++) {
collen = dbcollen(dbproc, i);
dbbind(dbproc, i, STRINGBIND, 80, &data[i - 1]) ;
ptr = dbcname(dbproc, i);
for (j = 1; j <= collen; j++) {
if (*ptr == '\0') putchar(' ');
else {
putchar(*ptr);
ptr++;
}
}
totlen = totlen + collen;
putchar(' ');
totlen = totlen + 1;
}
printf("\n");
for (i = 1; i <= totlen; i++) putchar('-');
printf("\n");
```

```

while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
for (i = 1; i <= numcols; i++) {
collen = dbcollen(dbproc, i);
ptr = &data[i - 1][0];
for (j = 1; j <= collen; j++) {
if (*ptr == '\0') putchar(' ');
else {
putchar(*ptr);
ptr++;
}
}
putchar(' ');
}
printf("\n");
}
printf("\n%d rows affected\n", DBCOUNT(dbproc)) ;
}
close:
fflush(stdout);
dbclose(dbproc) ;
dbexit();
exit(STDEXIT);
}

int getn(idata, min, max)
int      *idata;
int      min, max;

{
char      data[80];
char      *digit;
char      negative;

for (;;) {
if (gets(data) == NULL) return(-1);
if (strlen(data) == 0) return(0);
*idata = 0;
digit = data;
negative = 0;
if (*digit == '-') {
negative = 1;
digit++;
}
for (;;) {
if ((*digit < '0') || (*digit > '9') ||
(*idata > 1000000000)) {
digit = data;
while (*digit > '\0') {
*digit = toupper(*digit);
digit++;
}
if (strcmp(data, "EXIT") == 0) return(-2);
if (strcmp(data, "STOP") == 0) return(-2);
if (strcmp(data, "END") == 0) return(-2);
if (strcmp(data, "QUIT") == 0) return(-2);
}
}
}

```

```
printf("Invalid integer");
break;
}
*idata = *idata * 10 + (*digit - '0');
if (*++digit == '\0') {
if (negative) *idata = -*idata;
if ((*idata >= min) && (*idata <= max)) return(1);
printf("Value must be in the range %d to %d",
min, max);
break;
}
}
printf(" - please reenter - ");
}
}

int err_handler(dbproc, severity, dberr, oserr, dberrstr, oserrstr)
DBPROCESS      *dbproc;
int             severity;
int             dberr;
int             oserr;
char            *dberrstr;
char            *oserrstr;
{
if (dberr == SYBESMSG)
return (INT_CANCEL);

if ((dbproc == NULL) || (DBDEAD(dbproc)))
return(INT_EXIT);
else
{
fprintf(stderr, "DB-Library error:\n\t%s\n", dberrstr);
if (oserr != DBNOERR)
fprintf(stderr, "Operating-system error:\n\t%s\n",
oserrstr);
return(INT_CANCEL);
}
}

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
srvname, procname, line)

DBPROCESS      *dbproc;
DBINT          msgno;
int            msgstate;
int            severity;
char           *msgtext;
char           *srvname;
char           *procname;
DBUSMALLINT    line;

{
fprintf (stderr, "Msg %ld, Level %d, State %d\n",
msgno, severity, msgstate);
if (strlen(srvname) > 0)
fprintf (stderr, "Server '%s', ", srvname);
if (strlen(procname) > 0)
```

```
fprintf (stderr, "Procedure '%s', ", procname);  
if (line > 0)  
    fprintf (stderr, "Line %d", line);  
fprintf(stderr, "\n\t%s\n", msgtext);  
return(0);
```

 APPENDIX B *Sample Open Server Programs*

B.1 Retrieve/Send data with \$SRV functions

The following Janus/*Model 204* server application program communicates with the previous Sybase client code. The following JANUS DEFINE and START statements would have been previously executed to allow clients to access this server program:

```
JANUS DEFINE SRVWELL 2001 OPENSERV 50          -
OPEN FILE SIRGENR CMD 'INCLUDE SRVWELL' -
TIMEOUT 300
JANUS START SRVWELL
```

The define statement above causes the procedure SRVWELL to be automatically included from file SIRGENR as soon as the client connection is established. Processing within the procedure is based on the RPC name, which is found with \$SRV_RPCNAME and processed with the IF statement at label RPC.

Once the server program is included, it waits at \$SRV_WAIT until the client sends the RPC. After either of the two RPC is handled, a \$SRV_DONE is issued, and processing returns to the \$SRV_WAIT to wait for another RPC. If the value sent by the client is not a valid RPC, a \$SRV_CLOSE is executed, disconnecting the client.

Note that \$SRV_WAIT *must* be called before an RPC can be processed, and \$SRV_DONE *must* be called without its parameter set to 'MORE', to terminate the RPC. As this program sits in a loop, processing repeated RPCs from the same client, \$SRV_WAIT is called again after the \$SRV_DONE, to ready the server program for the next RPC.

```
PROCEDURE SRVWELL
B

%RC          IS FLOAT
%I           IS FLOAT
%SEL.LAT1    IS STRING LEN 50
%SEL.LAT2    IS STRING LEN 50
%SEL.LONG1   IS STRING LEN 50
%SEL.LONG2   IS STRING LEN 50

IMAGE LIST TEMP GLOBAL NAMESAVE
API.NBR      IS STRING LEN 14
SEL.LAT      IS STRING LEN 7
SEL.LONG     IS STRING LEN 9
FNL.CLS      IS STRING LEN 8
OPER.NM      IS STRING LEN 23
TOT.DEPTH    IS STRING LEN 9
END IMAGE
```

```
%RC = $SRV_DATA('UPCASE', 'ON')

RPC_LOOP:

%RC = $SRV_WAIT
IF %RC NE 1 THEN
JUMP TO ERROR
END IF

RPC:

IF $SRV_RPCNAME EQ 'WELL_RANGE' THEN
JUMP TO WELL_RANGE
ELSEIF $SRV_RPCNAME EQ 'WELL_LIST' THEN
JUMP TO WELL_LIST
END IF

%RC = $SRV_MSG(20001, 7, 'Invalid RPC')
JUMP TO ERROR

WELL_RANGE:

FV1:    IN FILE SIRGENR FOR 1 VALUE OF SEL.LAT IN ASCENDING
%RC = $SRV_PARMSET(, 'SEL.LAT.MIN', VALUE IN FV1)
END FOR

FV2:    IN FILE SIRGENR FOR 1 VALUE OF SEL.LAT IN DESCENDING
%RC = $SRV_PARMSET(, 'SEL.LAT.MAX', VALUE IN FV2)
END FOR

FV3:    IN FILE SIRGENR FOR 1 VALUE OF SEL.LONG IN ASCENDING
%RC = $SRV_PARMSET(, 'SEL.LONG.MIN', VALUE IN FV3)
END FOR

FV4:    IN FILE SIRGENR FOR 1 VALUE OF SEL.LONG IN DESCENDING
%RC = $SRV_PARMSET(, 'SEL.LONG.MAX', VALUE IN FV4)
END FOR

%RC = $SRV_DONE

JUMP TO RPC_LOOP

*****
*WELL_LIST - Handle well list RPC                                     *
*****

WELL_LIST:

PREPARE IMAGE LIST
%RC    = $SRV_SETROW('LIST')

%RC    = $SRV_PARMGET(, 'SEL.LAT1'  , %SEL.LAT1)
%RC    = $SRV_PARMGET(, 'SEL.LAT2'  , %SEL.LAT2)
%RC    = $SRV_PARMGET(, 'SEL.LONG1' , %SEL.LONG1)
%RC    = $SRV_PARMGET(, 'SEL.LONG2' , %SEL.LONG2)

*****
```

```

*-> Locate well data.
*****

IN SIRGENR CLEAR LIST PUMPERS
G1:   IN SIRGENR FDWOL
P1:   PLACE RECORDS IN G1 ON LIST PUMPERS
RELEASE RECORDS IN G1

IF %SEL.LAT1 NE '' THEN
G2:   FD ON LIST PUMPERS FOR WHICH SEL.LAT IS GE %SEL.LAT1
C2:   CLEAR LIST PUMPERS
PLACE RECORDS IN G2 ON LIST PUMPERS
END IF

IF %SEL.LAT2 NE '' THEN
G3:   FD ON LIST PUMPERS FOR WHICH SEL.LAT IS LE %SEL.LAT2
C3:   CLEAR LIST PUMPERS
PLACE RECORDS IN G3 ON LIST PUMPERS
END IF

IF %SEL.LONG1 NE '' THEN
G4:   FD ON LIST PUMPERS FOR WHICH SEL.LONG IS GE %SEL.LONG1
C4:   CLEAR LIST PUMPERS
PLACE RECORDS IN G4 ON LIST PUMPERS
END IF

IF %SEL.LONG2 NE '' THEN
G5:   FD ON LIST PUMPERS FOR WHICH SEL.LONG IS LE %SEL.LONG2
C5:   CLEAR LIST PUMPERS
PLACE RECORDS IN G5 ON LIST PUMPERS
END IF

%I = 0
FOR EACH RECORD ON LIST PUMPERS
%I = %I + 1
IF (%I GT 200) THEN
%RC = $SRV_MSG(20002, 1, -
'More rows available but not sent')
LOOP END
END IF
%LIST:API.NBR = API.NBR
%LIST:SEL.LAT = SEL.LAT
%LIST:SEL.LONG = SEL.LONG
%LIST:FNL.CLS = FNL.CLS
%LIST:OPER.NM = FLD.NM
%LIST:TOT.DEPTH = TOT.DEPTH
%RC = $SRV_SENDROW
IF %RC NE 0 THEN
PRINT '$SRV_SENDROW = ' WITH %RC
JUMP TO ERROR
END IF
END FOR

%RC = $SRV_DONE

JUMP TO RPC_LOOP

```

```
ERROR:

%RC = $SRV_DONE('ERROR')
%RC = $SRV_CLOSE

END
END PROCEDURE SRVWELL
```

B.2 Generic RPC router

This Janus/*Model 204* server program looks in JANUS for a User Language procedure of the same name as the client-supplied RPC. If the procedure is found it is executed. The JANUS DEFINE command causes the driver routine to be automatically invoked when the client connects.

```
JANUS DEFINE ADHOC 3001 Janus 10 - OPEN FILE OPENSERV CMD 'INCLUDE
ADHOC_RPC_DRIVER' JANUS START ADHOC
```

This procedure is a good example of a driver routine that allows the client to request the execution of any User Language procedure in a designated procedure file.

```
PROCEDURE ADHOC_RPC_DRIVER
*-----*
* Description:      Driver routine which will look for and include
*
*                  any stored procedure in file JANUS.
*
*-----*
B

* Variable declarations.

%PROC      IS STRING LEN 255
%RC        IS FLOAT          COMMON
%X         IS FLOAT

* Force upper case translation of RPC and parameter names.

%RC = $SRV_DATA('UPCASE','ON')

* $SRV_WAIT must be called before an RPC is processed.

%RC = $SRV_WAIT
IF %RC NE 1 THEN
CALL ERROR( 20001, 7, '$SRV_WAIT function failed.' )
END IF

* Retrieve the name of the RPC (the stored User Language procedure).

%PROC = $SRV_RPCNAME
IF %PROC EQ '' THEN
CALL ERROR(20002, 7, 'Null RPC specified.')
END IF
```

```

* Verify the RPC exists in file JANUS.

%X    = $RDPROC('OPEN','JANUS',%PROC)
IF $STATUS THEN
%X = $RDPROC('CLOSE',%X)
CALL ERROR(20003, 7, 'Invalid RPC. Procedure does not exist.')
END IF
%X    = $RDPROC('CLOSE',%X)

* Build a temporary procedure to INCLUDE the RPC.

%PROC = 'IN FILE JANUS INCLUDE ' WITH %PROC
%RC   = $BLDPROC(-3,'*', 'OPEN')
%RC   = $BLDPROC(-3,%PROC,'CLOSE')

* Error routine to null-out the INCLUDE procedure and close
* connection.

SUBROUTINE ERROR ( %MSG.NO  IS STRING LEN 10, -
%MSG.CLS IS STRING LEN  2, -
%MSG.TXT IS STRING LEN 255 )
%RC IS FLOAT COMMON

IF %MSG.NO NE '' THEN
%RC = $SRV_MSG( %MSG.NO, %MSG.CLS, %MSG.TXT )
%RC = $SRV_DONE('ERROR')
ELSE
%RC = $SRV_DONE('')
END IF
%RC   = $SRV_CLOSE
%RC   = $BLDPROC(-3,'*', 'OPEN')
%RC   = $BLDPROC(-3,'*', 'CLOSE')
STOP
END SUBROUTINE ERROR
END

* INCLUDE the user-selected RPC.

INCLUDE -3
END PROCEDURE ADHOC_RPC_DRIVER

```

B.3 Send U.L. procedure to client with \$SRV functions

This Janus/Model 204 server program sends a user-selected User Language procedure to the client. This procedure could be invoked as an RPC from the driver routine in the previous example.

```
PROCEDURE ADHOC_PROC_SEND
*-----*
* Description:      Send the lines of a user-specified
*
*                   User Language procedure to the client.
*-----*
B

*/ Variable declarations.

%FILE          IS STRING LEN  8
%HOLD          IS STRING LEN 132
%PASS          IS STRING LEN  8
%PROC          IS STRING LEN 255

%RC            IS FLOAT          COMMON
%X             IS FLOAT

%CURPRIV       IS FIXED  DP 0
%CURPRIV.0200 IS FIXED  DP 0

IMAGE PROC NAMESAVE
LINE          IS STRING LEN 132
END IMAGE

*/ Retrieve the procedure name, the file and the optional password,
*/ which should have been passed by client as named parameters.

%RC = $SRV_PARMGET('FILE',  %FILE)
%RC = $SRV_PARMGET('PASSWORD',%PASS)
%RC = $SRV_PARMGET('PROC',  %PROC)

IF %FILE EQ '' THEN
CALL ERROR( 20005, 7, 'No source procedure file specified.' )
END IF

IF %PROC EQ '' THEN
CALL ERROR( 20006, 7, 'No procedure specified.' )
END IF

*/ Open the source file.

IF %PASS EQ '' THEN
OPENC FILE %FILE
ELSE
OPENC FILE %FILE PASSWORD %PASS
END IF

*/ See if we got access to the file.
```

```
%CURPRIV = $VIEW('CURPRIV',%FILE)
IF $STATUS AND NOT %CURPRIV THEN
CALL ERROR( 20007, 7, 'Unable to open specified source file.' )
END IF

*/ See if we have adequate privileges to read procedures.

%CURPRIV.0200 = %CURPRIV / 512
IF NOT $MOD(%CURPRIV.0200,2) THEN
CALL ERROR( 20008, 7, 'Insufficient privileges to read proc.' )
END IF

*/ Open the procedure.

%X = $RDPROC('OPEN',%FILE,%PROC)
IF $STATUS THEN
CALL ERROR( 20009, 7, 'Specified procedure does not exist.' )
END IF

*/ Retrieve the procedure's LINEND character.

%HOLD = $RDPROC('LINEND',%X)
IF $STATUS THEN
CALL ERROR( 20010, 7, '$RDPROC LINEND error.' )
END IF

*/ Set the image for passing data back to client.

PREPARE IMAGE PROC
%RC = $SRV_SETROW('PROC')

*/ Send each line of the procedure to the client.

REPEAT WHILE $STATUS = 0
%PROC:LINE = $RDPROC('GET',%X)
%RC      = $SRV_SENDRROW
IF %RC THEN
%HOLD   = '$SRV_SENDRROW Failure. Return Code = ' WITH %RC
CALL ERROR( 20011, 7, %HOLD)
END IF
END REPEAT
%RC = $SRV_DONE('')
%RC = $SRV_CLOSE

*/ Error routine: optionally sends client a message & closes
connection.

SUBROUTINE ERROR ( %MSG.NO  IS STRING LEN 10, -
%MSG.CLS IS STRING LEN  2, -
%MSG.TXT IS STRING LEN 255 )
%RC IS FLOAT COMMON

IF %MSG.NO NE '' THEN
%RC = $SRV_MSG( %MSG.NO, %MSG.CLS, %MSG.TXT )
%RC = $SRV_DONE('ERROR')
ELSE
```

```
%RC = $SRV_DONE('')
END IF
%RC = $SRV_CLOSE
STOP
END SUBROUTINE ERROR
END
END PROCEDURE ADHOC_PROC_SEND
```

APPENDIX C *Datetime Processing Considerations*

This chapter presents date processing issues, including usage of *Janus Open Server* 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 Server*.

Janus Open Server 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 Server* 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 \$SRV_ functions, you may use the *Sirius Functions* to develop *Janus Open Server* applications. See the ***Sirius Functions Reference Manual*** 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 Server*, 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.

To correctly use *Janus Open Server* past the year 1999, version 4.6 of the *Sirius Mods*, or later, is required. 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.

Above and beyond the post-1999 requirements specific to *Janus Open Server*, 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 Server* 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 Server*, 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 \$SRV_ functions; you may, however, use a number of datetime \$functions, described in the *Sirius Functions Reference Manual*, which 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.

C.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 (e.g., “YYYY” for a 4 digit year, and “MI” for minutes) and separator characters (e.g., “/” in “MM/DD/YY” for two-digit month, day, and year separated by slashes).

These *datetime format* strings are used in many Sirius Software products in addition to *Janus Open Server*. 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 Sirius 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 \$SRV_ function arguments, but it may not in some cases in other Sirius 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 (i.e. “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 “l”, 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 represents 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 which convert *from* an alphabetic string (e.g., “MON”) will allow any case string (e.g., “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 <i>Sir2000 Field Migration Facility</i> .) |
| NS | numeric datetime value containing the number seconds since January 1, 1900 at 12:00 AM. (This token is allowed only in the <i>Sir2000 Field Migration Facility</i> .) |
| ND | numeric date value containing the number of days since January 1, 1900. (This token is allowed only in the <i>Sir2000 Field Migration Facility</i> .) |
| * | Ignore entire variable-length substring matching pattern, if any. See “Datetime and format examples” on page 84. |

I	Ignore corresponding input character. See “Datetime and format examples” on page 84.
"	Following character is “quoted”, that is, it acts as a separator character. See “Datetime and format examples” on page 84. This token is available starting with version 5.2 of the <i>Sirius Mods</i> .
YYYY	4 digit year
YY	2 digit year
CYY	Year minus 1900 (3 digits, including any leading zero). See “Datetime and format examples” on page 84.
ZYY	Year minus 1900, two-digit or three-digit year number (variable length data). See “Datetime and format examples” on page 84.
MONTH	Full month name (upper case variable length). When used as an argument to a \$SRV_ 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 \$SRV_ function for converting from a string, this is the same as MONTH.
MON	Three character month abbreviation (upper case). When used as an argument to a \$SRV_ 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 \$SRV_ function for converting from a string, this is the same as MON.
MM	Two-digit month number. When used as an argument to a \$SRV_ function for converting from a string, this is the same as BM (leading blank is allowed). See “Datetime and format examples” on page 84.
BM	Two-character month number; when used as an argument to a \$SRV_ function for converting from a string, from a string, this is the same as MM. See “Datetime and format examples” on page 84. This token is available starting with version 5.2 of the <i>Sirius Mods</i> .
DDD	Three digit Julian day number
DD	Two-digit day number. When used as an argument to a \$SRV_ function for converting from a string, this is the same as BD (leading blank is allowed). See “Datetime and format examples” on page 84.
BD	Two-character day number; when used as an argument to a \$SRV_ function for converting from a string, this is the same as DD. See “Datetime and format examples” on page 84. This token is available starting with version 5.2 of the <i>Sirius Mods</i> .
DAY	One-digit or two-digit day number (variable length data). See “Datetime and format examples” on page 84.
WKDAY	Full day of week name (upper case variable length). when used as an argument to a \$SRV_ 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 \$SRV_ function for converting from a string, this is the same as WKDAY.
WKD	Three character day of week abbreviation (upper case). When used as an argument to a \$SRV_ 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 \$SRV_ function for converting <i>from</i> a string, this is the same as WKD.
HH	Two-digit hour number. When used as an argument to a \$SRV_ function for converting <i>from</i> a string, this is the same as BH (leading blank is allowed). See “Datetime and format examples” on page 84 .
BH	Two-character hour number; When used as an argument to a \$SRV_ function for converting <i>from</i> a string, this is the same as HH. See “Datetime and format examples” on page 84 . This token is available starting with version 5.2 of the <i>Sirius Mods</i> .
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 84](#) for examples which include use of various separator characters.

Decimal digits as separator characters, and the quoting character are available starting with version 5.2 of the *Sirius Mods*.

C.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; e.g., 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 SPAN SIZE. The valid range of dates for all other formats is from 1 January 1753 thru 31 December 9999.

C.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.

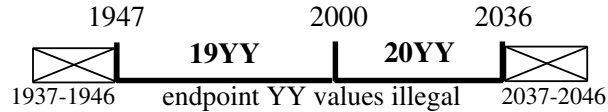
C.3.1 CENTSPAN

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.

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.

C.4 Datetime and format examples

There is an extensive set of format tokens, as shown in [“Datetime Formats” on page 78](#). 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 \$SRV_ functions, you may use the *Sirius Functions* to develop *Janus Open Server* applications. 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. See the ***Sirius Functions Reference Manual*** for descriptions of its \$functions.

YYMMDD This is the common 6-digit date format which supports sort order if all dates are within a single century. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -  
                  '960229', 'YYMMDD')
```

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

YYYYMMDD

This is the common 8-digit date format which supports sort order with dates in 2 centuries. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -  
                  '19921212', '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. 6-digit date format for display. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -  
                  '12/14/94', 'MM/DD/YY')
```

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

Notes - in the \$SRV_ 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. The BM token is available starting with version 5.2 of *Janus Open Server*.

DD.MM.YY

This is a European 6-digit date format for display. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -  
                  '14.12.94', 'DD.MM.YY')
```

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

Notes - in the \$SRV_ 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. The BD token is available starting with version 5.2 of *Janus Open Server*.

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 = $SRV_PARMSET( , 'ADD_DATE', -  
  'Friday, 7 February 1998 ' WITH -  
  'AT 21:33', -  
  '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 \$SRV_ 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 (e.g., “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. The BH token is available starting with version 5.2 of *Janus Open Server*.

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 = $SRV_PARMSET( , 'ADD_DATE', -  
  '92ABCD', '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 = $SRV_PARMSET( , 'ADD_DATE', -  
  '92', 'YY*')  
%RC = $SRV_PARMSET( , 'EXP_DATE', -  
  'ABC1992', '*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 * token may appear in a datetime format.

CYYDDD This is a compact 6-digit date format with explicit century information, from 1900 through and including 2899. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -
                    '097031', 'CYYDDD')
```

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

ZYYMDD

This is a compact 6- or 7-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 1900's. The following User Language fragment

```
%RC = $SRV_PARMSET( , 'ADD_DATE', -
                    '971201', 'ZYYMDD')
%RC = $SRV_PARMSET( , 'EXP_DATE', -
                    '1001201', 'ZYYMDD')
```

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 \$SRV_ 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 = $SRV_PARMSET( , 'ADD_DATE', -
                    '92000', 'YY0000')
```

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 preceded by a quote character (“”).
- Numeric separators are available starting with version 5.2 of *Janus Open Server*.

C.5 \$SRV_ Functions CENTSPAN Argument

Some of the \$SRV_ 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.

In *Model 204 V4R1.0*, CCA has taken a different approach to the default 100 year period is used; see the *Model 204* documentation for a description of the CENTSPLT and DEFCENT parameters and \$function arguments.

Index

A

ADDCA, JANUS subcommand ... 16
 ALLOCC parameter, JANUS DEFINE ... 20
 AUDTERM parameter
 JANUS DEFINE command ... 20

B

BINDADDR parameter, JANUS DEFINE ... 20
 BSIZE parameter, JANUS DEFINE ... 21

C

CENTSPAN ... 82, 88
 CENTSPLT argument ... 88
 CENTSPLT parameter ... 88
 Certificate, SSL ... 27
 in SSL cache ... 29, 33
 requested by server ... 31
 CHARSET parameter
 JANUS DEFINE ... 21
 CHARSET, JANUS subcommand ... 16
 Client certificate ... 28, 31
 Client IP address ... 44
 CLSOCK, JANUS subcommand ... 16
 CMD parameter
 JANUS DEFINE command ... 21
 CONFIGURATION, JANUS subcommand ... 16
 Connection limit ... 19
 Content type
 client Post data ... 26-27

D

Datatype of parameter ... 54
 \$SRV_PARMTYPE ... 54
 Date processing ... 77, 88
 DEFCENT argument ... 88
 DEFCENT parameter ... 88
 DEFINE, JANUS subcommand ... 16
 DEFINEIPGROUP, JANUS subcommand ... 16
 DEFINEREMOTE, JANUS subcommand ... 16
 DEFINEUSGROUP, JANUS
 subcommand ... 16
 DELCA, JANUS subcommand ... 16

DELETE, JANUS subcommand ... 16
 DELETEIPGROUP, JANUS
 subcommand ... 16
 DELETEREMOTE, JANUS subcommand ... 16
 DELETEUSGROUP, JANUS
 subcommand ... 17
 DISPLAY, JANUS subcommand ... 17
 DISPLAYCA, JANUS subcommand ... 17
 DISPLAYREMOTE, JANUS
 subcommand ... 17
 DISPLAYSOCK, JANUS subcommand ... 17
 DISPLAYWEB, JANUS subcommand ... 17
 DISPLAYXT, JANUS subcommand ... 17
 DOMAIN, JANUS subcommand ... 17
 DRAIN, JANUS subcommand ... 17

E

Encoding
 HTML form ... 26-27
 Ending a request ... 47
 \$SRV_DONE ... 47
 Environment definition (overview) ... 13
 EXEC2RPC ... 11
 EXEC2RPC parameter ... 45
 EXEC2RPC parameter, JANUS DEFINE ... 22

F

FORCE, JANUS subcommand ... 17
 FTP, JANUS subcommand ... 17

I

IBSIZE parameter, JANUS DEFINE ... 22
 Installation ... 4
 Introduction to JANUS ... 1
 IP address of client ... 44

J

JANUS ... 2
 Janus commands
 introduction to ... 15
 wildcards used with ... 15
 JANUS concepts ... 9

Janus functions ... 41, 43-44, 47-49, 51-56, 58
 \$SRV_BIND ... 41
 \$SRV_CLOSE ... 43
 \$SRV_DATA ... 44
 \$SRV_DONE ... 47
 \$SRV_LANGGET ... 48
 \$SRV_MSG ... 48
 \$SRV_NUMPARAM ... 49
 \$SRV_PARMGET ... 49
 \$SRV_PARMLEN ... 51
 \$SRV_PARMNAME ... 52
 \$SRV_PARMNUM ... 52
 \$SRV_PARMSET ... 53
 \$SRV_PARMSTYPE ... 54
 \$SRV_RPCNAME ... 55
 \$SRV_SENDRW ... 56
 \$SRV_SETROW ... 56
 \$SRV_WAIT ... 58
JANUS parameters ... 13
 TCPSERV ... 13
 TCPTYPE ... 13
JANUS, introduction to
 Janus IFDIAL Library ... 1
 Janus TCP/IP Base ... 1

K

Keepalive connection, TCP ... 35

L

LANGUAGE parameter
 JANUS DEFINE ... 23
Language requests ... 10
LANGUAGE, JANUS subcommand ... 17
Length of parameter ... 51
 \$SRV_PARMLEN ... 51
LIMITS, JANUS subcommand ... 17
LOADXR, JANUS subcommand ... 17

M

MASTER parameter, JANUS DEFINE ... 23
 defining OPEN CLIENT ports ... 23
Mixed-case User Language ... 39
Model 204 resource requirements ... 20
 buffer space requirements ... 20
MSG204 parameter ... 23, 45
 Sending terminal output to client ... 23, 45
MSG204L parameter ... 24
 Sending terminal output to client ... 24

N

Name of current RPC ... 55
 \$SRV_RPCNAME ... 55
Name of parameter ... 52
 \$SRV_PARMNAME ... 52
NAMESERVER, JANUS subcommand ... 17
NOAUDTERM parameter
 JANUS DEFINE command ... 24
NOUPCASE parameter ... 24
 Converting client data to upper case ... 24
Number of parameter with name ... 52
 \$SRV_PARMNAME ... 52
Number of RPC parameters ... 49
 Retrieving with \$SRV_NUMPARAM ... 49

O

OBSIZE parameter, JANUS DEFINE ... 25
OPEN parameter
 JANUS DEFINE command ... 25
Open Server User Language Coding ... 40
OPENSERV port type ... 19

P

Parameter ... 49, 51-54
 Datatype of ... 54
 \$SRV_PARMSTYPE ... 54
 Length of ... 51
 \$SRV_PARMLEN ... 51
 Name of ... 52
 \$SRV_PARMNAME ... 52
 Number of name ... 52
 \$SRV_PARMNUM ... 52
 Retrieving value ... 49
 \$SRV_PARMGET ... 49
 Setting return value ... 53
 \$SRV_PARMSET ... 53
Parameters ... 49
 Number of in RPC ... 49
 \$SRV_NUMPARAM ... 49
Performance ... 22, 25
Port, Janus
 definition ... 18
Ports ... 9
PRELOGINUSER parameter, JANUS
 DEFINE ... 26

R

RAWINPUT parameter, JANUS DEFINE ... 26
 RAWINPUTONLY parameter, JANUS
 DEFINE ... 27
 RELOAD, JANUS subcommand ... 17
 Retrieving language input ... 48
 \$SRV_LANGGET ... 48
 Retrieving value of parameter ... 49
 \$SRV_PARMGET ... 49
 Rows ... 56
 Sending row image to client ... 56
 \$SRV_SENDRROW ... 56
 Setting current row image ... 56
 \$SRV_SETROW ... 56
 RPCONLY parameter ... 46
 RPCONLY parameter, JANUS DEFINE ... 27
 RPCs ... 10

S

Sample code ... 61, 69, 72, 74
 Sybase client ... 61
 Sybase server 1 (send/receive data) ... 69
 Sybase server 2 (Generic RPC router) ... 72
 Sybase server 3 (send proc to client) ... 74
 SDAEMON ... 13
 defined ... 13
 Sdaemon threads
 limit per Online ... 19
 SDAEMONS ... 13
 setting NSUBTKS ... 13
 SDS port type ... 19
 Sending a message to the client ... 48
 \$SRV_MSG ... 48
 Sending a row to a client ... 56
 \$SRV_SENDRROW ... 56
 \$SRV_SETROW ... 56
 Server ports ... 9
 Setting an output parameter ... 53
 \$SRV_PARMSET ... 53
 SIRIUS file ... 4
Sirius Mods ... 4
 SRVSOCK, JANUS subcommand ... 17
 SSL certificate ... 31
 See also Certificate, SSL
 SSL parameter, JANUS DEFINE ... 27
 SSLBSIZE parameter, JANUS DEFINE ... 29
 SSLCACHE parameter, JANUS DEFINE ... 29
 SSLCIPH parameter, JANUS DEFINE ... 30
 SSLLCERT parameter, JANUS DEFINE ... 31

SSLCLCERTR parameter, JANUS
 DEFINE ... 31
 SSLIBSIZE parameter, JANUS DEFINE ... 32
 SSLMAXAGE parameter, JANUS
 DEFINE ... 32
 SSLMAXCERTL parameter, JANUS
 DEFINE ... 33
 SSLOBFSIZE parameter, JANUS DEFINE ... 33
 SSLPROT parameter, JANUS DEFINE ... 34
 SSLSTATUS, JANUS subcommand ... 17
 SSLUNENC parameter, JANUS DEFINE ... 35
 START, JANUS subcommand ... 17
 STATUS, JANUS subcommand ... 17
 STATUSCA, JANUS subcommand ... 18
 STATUSREMOTE, JANUS subcommand ... 18

T

TCP keepalives ... 35
 TCPKEEPALIVE parameter, JANUS
 DEFINE ... 35
 TCPLOG, JANUS subcommand ... 18
 Terminal output ... 20, 23-24, 45
 Sending to client ... 23-24, 45
 TIMEOUT parameter
 JANUS DEFINE ... 36
 Timeouts, session ... 36
 TNSERV port type
 TCPKEEPALIVE processing ... 36
 TRACE parameter ... 36
 TRACE, JANUS subcommand ... 18
 Translation, character set ... 37
 TSTATUS, JANUS subcommand ... 18

U

UL/SPF ... 4
 UPCASE parameter ... 37, 46
 Converting client data to upper case ... 37,
 46
 User Language coding considerations ... 59
 AUDIT ... 59
 debugging JANUS UL applications ... 59
 full screen I/O ... 59
 ON units ... 59
 Open Server applications ... 59
 PRINT ... 59
 Userid and password ... 24, 37
 Converting to upper case ... 24, 37

W

Waiting for client requests ... 58
 \$SRV_WAIT ... 58
WEB, JANUS subcommand ... 18
WEBSERV port type ... 19

X

XTAB parameter
 JANUS DEFINE command ... 37

